

A Cautionary Evaluation of LLMs for TLS Normative Requirement Classification

Majid Mollaeefar

`mmollaeefar@fbk.eu`

Fondazione Bruno Kessler (FBK)

Riccardo Germani

Fondazione Bruno Kessler (FBK)

Salvatore Manfredi

Fondazione Bruno Kessler (FBK)

Silvio Ranise

Fondazione Bruno Kessler (FBK)

Research Article

Keywords: LLMs, Normative Requirements, RFC 2119 Classification, Technical Standards

Posted Date: August 24th, 2026

DOI: <https://doi.org/10.21203/rs.3.rs-10667541/v1>

License: © ⓘ This work is licensed under a Creative Commons Attribution 4.0 International License.

[Read Full License](#)

Additional Declarations: No competing interests reported.

A Cautionary Evaluation of LLMs for TLS Normative Requirement Classification

Majid Mollaeefar^{a,1}, Riccardo Germani^{b,1,2}, Salvatore Manfredi^{c,1}, Silvio Ranise^{d,1,2}

¹Center for Cybersecurity, Fondazione Bruno Kessler (FBK), Trento, Italy

²University of Trento, Trento, Italy

Abstract Extracting and classifying normative requirements from technical compliance standards is a critical task in cybersecurity auditing, where misclassification carries direct security consequences. This work evaluates the suitability of large language models (LLMs)—both individually and in ensemble configurations—for automating TLS normative requirement classification from technical standards. Using NIST SP 800-52 Revision 2 as a case study, we apply structured chain-of-thought prompting and accuracy-weighted multi-model ensemble voting across four LLMs to classify 144 TLS cipher suite patterns. The ensemble achieves 93.75% accuracy—a 0.69 percentage point improvement over the best individual model—and an ablation study confirms this improvement is robust across model configurations. However, with 8 out of 144 patterns still misclassified, the overall error rate remains prohibitive for compliance contexts where every classification carries accountability. Error analysis reveals that over 51% of failures stem from systematic confusion between SHOULD NOT (discouraged-but-conditionally-allowed) and MAY (unconditionally permissive), exposing a fundamental limitation in LLMs’ ability to interpret conditional discouragement language. We argue that current LLMs, even with structured reasoning and ensemble aggregation, are insufficiently reliable for standalone compliance classification, and we identify specific linguistic structures that must be addressed before such automation can be responsibly deployed.

Keywords LLMs, Normative Requirements, RFC 2119 Classification, Technical Standards

1 Introduction

Transport Layer Security (TLS) is the most widely deployed cryptographic protocol on the internet, securing communications across billions of endpoints in web, email, and messaging infrastructure. Its standardization has been a continuous, multi-decade effort spanning TLS 1.0 through 1.3, with each revision deprecating weaker configurations such as removing cipher suites without forward secrecy, mandating Authenticated Encryption with Associated Data (AEAD) encryption, and, most recently, preparing for post-quantum cryptographic migration. Compliance standards such as NIST SP 800-52 Revision 2 [25] codify these evolving best practices into normative requirements expressed in natural language prose. Every revision of the protocol or the standard forces security practitioners to manually re-extract and re-codify the applicable requirements, which is a process fundamentally at odds with the pace of TLS evolution.

In prior work [21], we developed TLSAssistant, an automated compliance checker that detects TLS misconfigurations against NIST SP 800-52. The tool is effective, but it depends on a manually curated audit dataset: domain experts spent months translating the standard’s natural language requirements into machine-readable rules. This translation step remains the critical bottleneck in the compliance automation pipeline. It is time-consuming, error-prone, and must be repeated whenever the standard is updated.

TLS cipher suites follow a deterministic naming convention (e.g., TLS_AES_128_GCM_SHA256) that enables pattern-based context extraction. While this structuredness is unusual compared to less regimented standards (e.g., OAuth, OpenID Connect), TLS’s ubiquity and criticality make the investment worthwhile. Even if the approach is not immediately transferable to every protocol, automating compliance for TLS alone has significant practical impact.

^aCorresponding author. e-mail: mmollaeefar@fbk.eu

^bCorresponding author. e-mail: rgermenia@fbk.eu

^ce-mail: smanfredi@fbk.eu

^de-mail: ranise@fbk.eu

This automation is far from trivial. These standards specify requirements using RFC 2119¹ normative keywords (MUST, SHOULD, MAY, etc.), which define precise compliance obligations, but their interpretation is complicated by extensive use of conditional language, cross-references across sections, and context-dependent exceptions. For example, Appendix D.1 of the NIST SP 800-52 Revision 2 [25] says:

“While these guidelines do not recommend cipher suites using RSA key transport, there may be circumstances in practice where RSA key transport is needed. For example, if an agency uses a network appliance for regulatory or enterprise security purposes that only functions with these cipher suites, then these cipher suites may need to be enabled.”

which discourages the use of RSA cipher suites unless they are required by a necessary service. The subsequent paragraph defines the list of allowed cipher suites with the following text:

“If RSA key transport is needed while a new traffic inspection strategy is being developed, only RSA key transport cipher suites from the following list may be used.”

By reading this phrase without considering the one above, the reader could interpret the usage of RSA cipher suites as neither encouraged nor discouraged.

This example illustrates that correct classification requires reasoning over multiple paragraphs and their interdependencies—a capability that rule-based extractors, which operate on isolated sentences, fundamentally lack. Large language models (LLMs) offer the promise of automating this classification through their ability to process broader context and interpret nuanced language. However, RFC 2119 requirement-based classification presents unique challenges beyond conventional text understanding, where it requires models to accurately interpret nuanced distinctions between mandatory obligations (MUST), recommendations (SHOULD), permissions (MAY), discouragements (SHOULD NOT), and prohibitions (MUST NOT), while handling negations, conditional clauses, and context-dependent exceptions. In compliance auditing, even a single misclassification can lead to deploying prohibited cipher suites or overlooking mandatory requirements—raising the question of whether current LLMs are sufficiently trustworthy for this domain.

In this work, we present an automated framework for extracting and classifying normative requirements from structured technical compliance documents, and evaluate its suitability for TLS compliance auditing applications. For documents with well-defined pattern identifiers (e.g., TLS cipher suite names, API endpoints), we employ deterministic pattern matching to locate relevant textual contexts. The core

challenge addressed in this work is *accurate requirement classification* through two complementary mechanisms:

1. **Structured Reasoning Chain Prompting:** Multi-component prompts that embed RFC 2119 definitions, enforce step-by-step logical reasoning through a six-stage decision tree, and produce outputs in structured JSON schemas.
2. **Weighted Multi-Model Ensemble:** Accuracy-based weighted voting across multiple LLMs to mitigate model-specific biases. Each model contributes votes proportional to its validation accuracy, with the ensemble selecting requirement levels through majority aggregation.

We validate this framework on a list of TLS cipher suite patterns from NIST SP 800-52 Revision 2, achieving 93.75% ensemble accuracy with four LLMs—a 0.69 percentage point improvement over the best individual model. Our error analysis reveals that over 51% of classification failures stem from confusion between SHOULD NOT (discouraged but conditionally allowed) and MAY (unconditionally permissive), highlighting a systematic limitation in LLM interpretation of conditional requirement language. These results indicate that current LLMs, even with structured reasoning and ensemble aggregation, are not yet sufficiently reliable for standalone deployment in compliance auditing. We argue that the structured reasoning chains, while providing useful traceability, are insufficient to compensate for the fundamental semantic confusion LLMs exhibit at the SHOULD NOT–MAY boundary.

The contributions of this work are: (1) a systematic evaluation of LLM-based TLS normative requirement classification revealing that 93.75% accuracy is insufficient for compliance-critical applications, (2) empirical demonstration that weighted multi-model ensembles provide robust but modest improvements, (3) detailed error analysis identifying conditional discouragement language as the dominant and persistent failure mode, and (4) concrete recommendations for standardization bodies and practitioners on the limitations of current LLM-based compliance automation.

The rest of this paper is organized as follows. Section 2 provides relevant background and related work. Section 3 presents our proposed framework. Section 4 describes the experimental setup and analyzes results, including the full classification outcomes. Section 5 discusses limitations and future directions, and Section 6 concludes with implications for practice.

2 Background & Related Work

This section reviews the relevant background for our work. We first describe the challenges of translating technical specifications written in natural language into machine-readable

¹<https://www.rfc-editor.org/rfc/rfc2119.html>

formats (Section 2.1). We then discuss AI techniques relevant to addressing these challenges, namely prompt engineering and ensemble learning (Section 2.2). Finally, we review approaches that combine these techniques to implement the mapping from natural language specifications to structured formats (Section 2.3). For each, we review relevant state-of-the-art approaches and identify open challenges that motivate our work.

2.1 Requirements Extraction from Technical Standards

Technical standards documents, such as NIST guidelines, RFCs, and industry specifications, define compliance requirements using structured normative language. RFC 2119 [3] standardizes this vocabulary through five requirement levels: MUST (mandatory), MUST NOT (prohibited), SHOULD (recommended), SHOULD NOT (not recommended), and MAY (optional).²

These keywords establish unambiguous obligations critical for security auditing, regulatory compliance, and system certification.

Manual extraction of normative requirements from lengthy standards is resource-intensive and error-prone. Regulatory texts impose complex requirements that necessitate collaboration between requirements engineers and legal domain experts to resolve ambiguity and ensure accurate interpretation [22]. Moreover, empirical studies demonstrate that even domain experts exhibit significant disagreement when identifying ambiguous requirements in natural language specifications [17], motivating automated extraction methods. Such automated methods can be beneficial for works like [12] in which the onerous manual extraction process resulted in the creation of an auditable dataset [11], then employed by an actionable tool able to automatically perform compliance analyses.

Early automated approaches relied on rule-based systems and pattern-matching techniques. Regular expression-based extractors identify modal verbs combined with syntactic patterns to locate requirement statements [29]. Dependency parsing methods analyze sentence structure to extract subject-obligation-condition triples, leveraging linguistic analysis of modality and negation [23]. While computationally efficient, these approaches suffer from brittleness: they fail when requirements use non-standard phrasing (“it is required that”), implicit obligations, or domain-specific terminology [9].

A parallel development is the emergence of machine-readable standards formats that aim to eliminate the natural language interpretation problem altogether. The Open Security Controls Assessment Language (OSCAL) [15], developed by NIST, provides a standardized XML/JSON/YAML

representation for security controls, assessment plans, and compliance artifacts. OSCAL enables automated compliance checking without NLP-based requirement extraction by encoding controls in a structured, queryable format. However, the vast majority of existing standards including most NIST Special Publications, ISO standards, and industry specifications remain published in natural language prose, with machine-readable representations available only after costly manual translation, if at all. This gap between existing prose standards and future machine-readable formats motivates the need for reliable automated extraction in the interim, while also suggesting that the most sustainable long-term solution may be to mandate machine-readable formats at the standards publication stage rather than relying on NLP post-processing.

More sophisticated methods employ semantic role labeling to identify actors, actions, and constraints within requirement statements [35]. Information extraction pipelines combine techniques such as named entity recognition, relation extraction, and semantic role labeling to build structured requirement databases [24]. However, these techniques require extensive feature engineering and domain-specific training data, limiting generalizability across different standards [6].

2.2 Prompt Engineering and Ensemble Learning

Large language models (LLM) output quality heavily depends on prompt design [26]. Zero-shot prompting provides task descriptions without examples, while few-shot learning includes demonstrations of desired input-output pairs [4]. Chain-of-thought (CoT) prompting instructs models to generate intermediate reasoning steps before final answers, significantly improving performance on multi-step reasoning tasks [33], even in zero-shot settings [18].

Recent advances explore structured prompting strategies: decomposed prompting breaks complex queries into subtask sequences [16], least-to-most prompting decomposes problems into subproblems solved sequentially from simplest to most complex [36], and self-consistency decoding samples multiple reasoning paths then aggregates via majority vote [32]. For classification tasks, format-constrained generation uses finite-state machines to enforce structured output formats [34].

In machine learning, ensemble learning aggregates predictions from multiple classifiers to improve robustness and accuracy [7]. For LLMs, ensemble strategies show consistent gains over single-model baselines. Examples include mixture-of-agents architectures that combine outputs from diverse LLMs through layered aggregation [31]; multi-model debate, which refines answers through iterative critique [8]; and self-consistency sampling, which aggregates multiple independent samples from the same model [32].

²RFC 2119 defines two keywords for each requirement level (e.g., SHOULD NOT and NOT RECOMMENDED) which are equivalent.

Scaling the number of sampling agents with majority voting yields consistent gains over single-model baselines [20]. Multi-round discussion among diverse LLMs converges toward consensus through confidence-weighted voting [5]. Pairwise ranking followed by generative fusion provides another effective strategy for combining outputs from heterogeneous LLMs [14]. However, these strategies have predominantly been evaluated on general NLP benchmarks; their effectiveness for domain-specific normative classification with structured output requirements remains underexplored.

2.3 LLMs for Document Understanding and Classification

LLMs have demonstrated remarkable zero-shot and few-shot capabilities in document understanding tasks without task-specific fine-tuning [4, 30]. Recent work applies LLMs to requirements engineering domains such as assessing LLM capabilities for requirements elicitation, analysis, and specification [1], and compliance analysis of regulatory texts [13].

For technical document analysis, retrieval-augmented generation (RAG) architectures combine LLM reasoning with deterministic document retrieval [19]. This hybrid approach addresses LLM hallucination risks by grounding responses in retrieved textual evidence, crucial for compliance contexts requiring verifiable citations [10]. However, most RAG implementations rely on dense vector retrieval, which may obscure the provenance of retrieved passages through embedding compression—a limitation in compliance contexts where full auditability of retrieved evidence is required.

Nevertheless, applying LLMs to technical documents such as RFC 2119 classification presents unique challenges. Unlike binary sentiment analysis or topic categorization, normative requirement classification requires: (1) distinguishing five normative strength levels with subtle semantic boundaries, (2) correctly interpreting negations and conditional clauses that invert requirement polarity, and (3) handling domain-specific deontic language—such as distinguishing obligation from permission—where fine-grained modality detection requires specialized modeling beyond general-purpose NLP [28]. These challenges motivate the structured prompting and multi-model ensemble approach we present in the following section.

3 Proposed Solution

While the requirements engineering community has explored various ML and NLP approaches for requirement extraction, our work focuses specifically on applying LLMs to cybersecurity compliance auditing. This domain imposes several demands that motivate our framework design:

1. Security auditors must distinguish between mandatory (MUST), recommended (SHOULD), and permissive (MAY) requirements—not merely detect whether a statement is a requirement. Misclassifying a MUST NOT as a MAY could lead to deploying prohibited cipher suites, with direct security consequences.
2. Security standards frequently attach conditions to requirements (e.g., “supported when using TLS 1.2 with RSA certificates”). An auditing tool must capture these conditions, as they determine whether a requirement applies to a given system configuration.
3. In regulatory compliance contexts, human auditors must be able to verify and justify automated classifications. Black-box predictions are insufficient; structured reasoning chains that expose the decision logic are essential for audit traceability.
4. When classifications carry compliance consequences, relying on a single model prediction introduces unacceptable risk. Ensemble aggregation across multiple models and inference rounds provides the reliability that security auditing demands.

In this work, we address these demands through a unified framework combining deterministic retrieval, structured prompting with explicit reasoning chains, and weighted multi-model ensemble aggregation, which is designed for automated security compliance auditing of technical standards.

The remainder of this section details the three phases of our framework, illustrated in Figure 1. While we instantiate it with contemporary LLMs for validation purposes (Section 4), the design is extensible to any LLM backend capable of structured output generation. Phase 1 prepares input data and extracts relevant textual contexts through deterministic pattern matching. Phase 2 constructs structured prompts that embed RFC 2119 definitions, enforce a six-step reasoning chain, and produce outputs to strict JSON schemas. Phase 3 queries multiple LLMs and combines their predictions through accuracy-weighted ensemble voting across multiple rounds, leveraging both model diversity and temporal consistency to produce final classifications.

3.1 Phase 1: Input Preparation and Context Extraction

The first phase establishes the foundation for requirement classification by preparing input data and extracting relevant textual contexts. This phase takes two primary inputs: (1) a compliance document in Markdown format (e.g., NIST SP 800-52 Revision 2), and (2) a list of target patterns to be classified (e.g., TLS cipher suite identifiers). For the former, Markdown is chosen for the sake of ease of parsing and machine readability. We also consider a preprocessing step to convert alternative document formats in PDF into Markdown

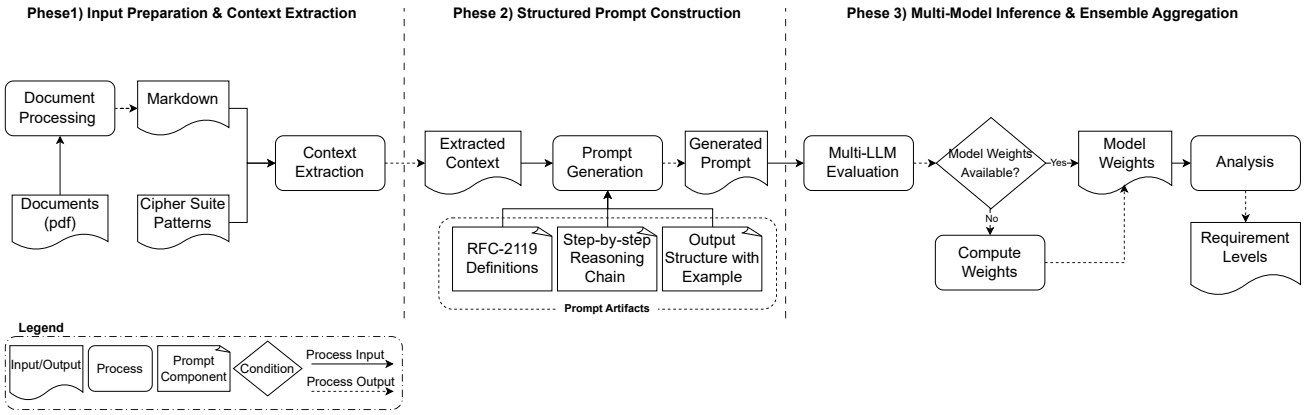


Fig. 1 Process view of our proposed framework.

using existing tools and libraries (e.g., Pandoc³, Docling⁴) to ensure uniformity.

For each target pattern from the input list (e.g., TLS_AES_256_GCM_SHA384), we perform case-insensitive regex matching across all document sections and once we find the pattern, retrieve its enclosing section. Here, a *section* corresponds to a top-level segment of the Markdown document delimited by header markers (#). This means the retrieved context includes not just the paragraph containing the pattern, but the entire section in which it appears—encompassing all subsections and paragraphs under that header. When a pattern matches multiple sections, all sections are aggregated to provide comprehensive coverage. This deterministic retrieval approach provides transparency and reproducibility while avoiding the semantic ambiguity of dense vector-based retrieval methods. Our framework implements a RAG approach, where pattern-based retrieval identifies relevant contexts, and LLM reasoning performs requirement classification over the retrieved text. This hybrid approach combines the precision of deterministic retrieval with the semantic understanding capabilities of large language models.

The output of this process is a structured dataset where each row contains: (1) a pattern identifier (cipher suite name), and (2) its complete textual context (all document sections mentioning that pattern). This dataset serves as input to Phase 2, where prompts are constructed for each pattern-context pair.

3.2 Phase 2: Structured Prompt Construction

The second phase transforms each pattern-context pair from Phase 1 into a structured prompt optimized for RFC 2119 classification. LLMs exhibit significant sensitivity to prompt design, with variations in prompt structure often yielding

inconsistent outputs [2]. To mitigate this variability and enforce domain-specific reasoning, as can be seen in Figure 1 (prompt artifacts dashed-box in phase 2), we employ a multi-component prompt template that combines 1) explicit RFC 2119 definitions, 2) a step-by-step reasoning chain, and 3) output structure with an example. These components, together with the extracted context from the previous phase, construct our final prompt for each pattern.

3.2.1 RFC 2119 Definition Grounding

The prompt explicitly embeds the complete RFC 2119 definitions for all five requirement levels (MUST, MUST NOT, SHOULD, SHOULD NOT, MAY) plus NOT MENTIONED⁵. This grounding prevents the model from hallucinating requirement semantics and ensures consistent interpretation across different model architectures and training distributions.

3.2.2 Six-Step Reasoning Chain

To encourage systematic analysis rather than pattern matching, we structure the reasoning process as six explicit steps:

1. **Sentence Identification:** Locate all sentences mentioning the target pattern or related concepts.
2. **Keyword Extraction:** Identify RFC 2119 keywords (or their synonyms) within these sentences.
3. **Negation Detection:** Check for negations or conditional phrases that modify requirement semantics (e.g., “may” vs. “shall not”).
4. **Contextual Analysis:** Evaluate the overall tone and strength of language used in the surrounding context.
5. **Hierarchical Decision Tree:** Apply a cascading decision logic:
 - Check for prohibitive language (MUST NOT, SHALL NOT) → MUST NOT

³<https://pandoc.org/>

⁴<https://www.docling.ai/>

⁵It is used to label patterns that do not appear in the document.

- If absent, check for mandatory language (MUST, SHALL, REQUIRED) → MUST
- Evaluate conditional acceptance with discouragement (SHOULD NOT, NOT RECOMMENDED) → SHOULD NOT
- If permissive, check for recommendatory language (SHOULD, RECOMMENDED) → SHOULD
- Check for optional/permissive language (MAY, OPTIONAL) → MAY
- If no explicit guidance appears → NOT MENTIONED

6. **Justification:** Provide explicit reasoning for the final classification.

This reasoning chain is templated directly into the prompt, requiring the model to provide intermediate outputs for each step. This transparency facilitates error analysis and debugging while reducing the likelihood of misclassifications.

3.2.3 Output Structure

Beyond requirement level classification, many requirements are conditional—they apply only under specific circumstances (e.g., “supported when using TLS 1.2 with RSA certificates”). The prompt therefore also requests identification of such conditions, encoded as a binary flag (Conditional: YES/NO) and a textual description of the triggering condition.

We enforce a strict five-field JSON schema as below to the end of the prompt, together with an explicit example to guide the model’s output format:

```
{
  "Pattern": "<cipher suite name>",
  "Requirement": "<MUST|MUST NOT|SHOULD|...>",
  "Conditional": "<YES|NO>",
  "Condition": "<condition description or 'None'>",
  "Reasoning": "<justification>"
}
```

3.3 Phase 3: Multi-Model Inference and Ensemble Aggregation

By querying multiple LLM backends with the structured prompts from Phase 2, this phase produces the final output containing predicted requirement levels for each input pattern, along with model weights.

3.3.1 Multi-Model LLM Inference

For each pattern’s structured prompt, we execute inference across multiple LLM backends. Each inference request is executed with a fresh model context to prevent contamination from previous queries, ensuring that each pattern is evaluated independently without carryover effects. For robustness, we consider to perform multiple independent inference runs

per model. This repetition captures model stochasticity and provides a basis for consistency measurement. Single-model predictions, even leveraging state-of-the-art LLMs, exhibit non-trivial error rates due to training data biases, prompt sensitivity, and stochasticity; therefore, multi-round inference helps quantify this variability.

3.3.2 Weighted Ensemble Voting Formalization

To improve robustness and accuracy, we employ a weighted ensemble methodology that combines predictions from multiple LLM backends across multiple inference rounds. We formally define the voting mechanism as follows:

Notation:

- Let $M = \{m_1, m_2, \dots, m_K\}$ be the set of K models
- Let $R = \{r_1, r_2, \dots, r_N\}$ be the set of N inference rounds per model
- Let $\mathcal{L} = \{l_1, l_2, \dots, l_L\}$ be the set of RFC 2119 requirement levels
- Let $w_i \in [0, 1]$ be the normalized weight for model m_i , where $\sum_{i=1}^K w_i = 1$
- Let $p_{i,j}^{(c)} \in \mathcal{L}$ be the prediction of model m_i in round r_j for cipher suite pattern c

Model Weight Computation: Model weights are derived empirically from validation performance on the same manually annotated ground truth dataset⁶ used for final evaluation (Section 4.1). For each model m_i , we compute its accuracy a_i on this dataset, then normalize:

$$w_i = \frac{a_i}{\sum_{k=1}^K a_k} \quad (1)$$

This ensures that higher-accuracy models exert proportionally greater influence while all models contribute to the ensemble decision.

Vote Accumulation: For each pattern c and each requirement level $\ell \in \mathcal{L}$, we accumulate votes across all models and all rounds:

$$V_\ell^{(c)} = \sum_{i=1}^K \sum_{j=1}^N w_i \cdot \mathbb{I}[p_{i,j}^{(c)} = \ell] \quad (2)$$

where $\mathbb{I}[\cdot]$ is the indicator function that equals 1 when the condition is true, 0 otherwise. This accumulation means that if a model consistently predicts the same requirement level across multiple rounds, its contribution to that level’s vote grows proportionally to both its weight and consistency.

Ensemble Decision: The final ensemble prediction for pattern c selects the requirement level with the highest accumulated vote:

$$\hat{\ell}^{(c)} = \arg \max_{\ell \in \mathcal{L}} V_\ell^{(c)} \quad (3)$$

⁶<https://github.com/stfbk/tls-compliance-dataset/blob/master/markdown/Cipher%20Suites.md>

Example: Consider a 3-model ensemble with weights $w_1 = 0.346$, $w_2 = 0.333$, $w_3 = 0.321$ running 3 rounds each. If for pattern c :

- Model 1 predicts MAY in all 3 rounds: contributes $3 \times 0.346 = 1.038$ votes to MAY
- Model 2 predicts MAY in rounds 1-2, SHOULD NOT in round 3: contributes $2 \times 0.333 = 0.666$ votes to MAY and $1 \times 0.333 = 0.333$ votes to SHOULD NOT
- Model 3 predicts MAY in all 3 rounds: contributes $3 \times 0.321 = 0.963$ votes to MAY

The accumulated votes are:

$$V_{\text{MAY}}^{(c)} = 1.038 + 0.666 + 0.963 = 2.667 \text{ and } V_{\text{SHOULD NOT}}^{(c)} = 0.333.$$

The ensemble selects MAY as the final prediction. We will further analyze this voting behavior in Section 4.4.2.

4 Validation and Experimental Results

To evaluate the effectiveness and limitations of our proposed framework, we instantiated the pipeline with four large language models and validated their performance against manually curated ground truth classifications. This section presents the experimental setup, quantitative results, error analysis, and an assessment of whether the observed accuracy is sufficient for compliance-critical applications.

To guide our investigation in evaluating how helpful LLMs could be in supporting compliance auditing, we formulated the following research questions:

- **RQ1:** How accurately can LLMs classify normative requirement levels in technical security standards, and is this accuracy sufficient for compliance auditing?
- **RQ2:** Does combining predictions from multiple LLMs through weighted ensemble voting improve classification accuracy?
- **RQ3:** Where do LLMs fail in requirement classification, and what characteristics of normative language cause persistent misclassification?
- **RQ4:** What are the practical implications of these failure patterns for deploying LLM-based compliance tools in production environments?

4.1 Experimental Setup

4.1.1 Ground Truth Dataset

Our evaluation dataset consists of 144 TLS cipher suite patterns extracted from NIST Special Publication 800-52 Revision 2 [25] and compiled from the TLS Compliance Dataset [11]. Each pattern was manually annotated by domain experts with its normative requirement level according

to RFC 2119 semantics. Table 1 presents the ground truth distribution.

4.1.2 Configuration

For our experimental analysis, we instantiated our framework with four open-weight LLMs (*gemma3: 27b*, *qwen3:32b*, *gpt-oss:20b*, and *phi4:14b*) via Ollama inference server on a dedicated machine equipped with an Intel(R) Xeon(R) Gold 5218R CPU, 64 GB of RAM, and an NVIDIA RTX A5000 GPU with 24 GB of VRAM. LLMs use a temperature parameter (0.0-1.0) controlling output variability: higher values increase creativity, while lower values produce more deterministic responses [27]. We configured temperature = 0.0 for deterministic decoding to maximize reproducibility, and performed three independent inference rounds per model, yielding twelve total validation runs across the dataset.

4.1.3 Evaluation Metrics

For each model, we computed:

- **Accuracy:** The percentage of correctly classified patterns, computed independently for each of the three inference rounds.
- **Standard Deviation (SD):** Measures the variability of accuracy across the three rounds, where lower values indicate more stable performance.
- **Consistency Score:** Defined as $100 - \sigma_{\text{accuracy}}$, where σ_{accuracy} is the standard deviation of accuracy across rounds, where a score of 100 indicates perfect reproducibility.
- **Confusion Matrix:** Per-class breakdown of (actual requirement $\rightarrow$ predicted requirement) to identify systematic error patterns.

4.2 Overall Performance Results

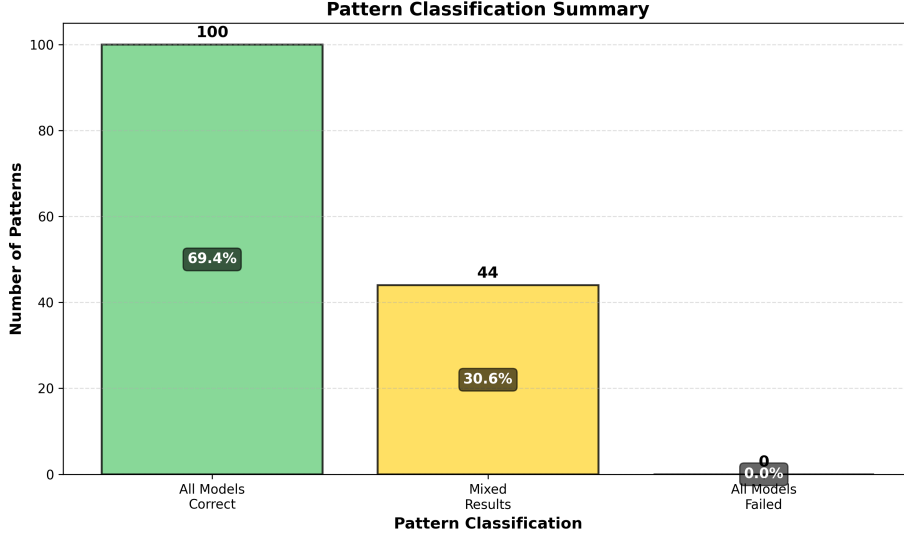
Table 2 summarizes the performance of individual models across three independent rounds, where sub-columns R1, R2, and R3 report the accuracy obtained in each respective round. Figure 2 visualizes the distribution of pattern difficulty across all models.

Table 1 Ground truth distribution of TLS cipher suite requirement classifications.

Requirement Level	Count	Percentage
MAY	84	58.3%
MUST NOT	49	34%
SHOULD NOT	10	6.9%
MUST	1	0.7%
SHOULD	0	0%
NOT MENTIONED	0	0%

Table 2 Model performance across cipher suite patterns (3 rounds per model).

Model	Accuracy (%)				SD	Consistency	Weight
	R1	R2	R3	Avg			
<i>gemma3</i>	93.06	93.06	93.06	93.06	0.00%	100.00	0.2585
<i>qwen3</i>	93.06	92.36	91.67	92.36	0.69%	99.31	0.2566
<i>gpt-oss</i>	88.89	88.89	88.89	88.89	0.00%	100.00	0.2469
<i>phi4</i>	87.50	85.42	84.03	85.65	1.75%	98.25	0.2379

**Fig. 2** Pattern classification summary across all models.

4.2.1 Key Observations

These results address **RQ1** and **RQ4**: the structured chain-of-thought prompting enables LLMs to achieve 85.65%–93.06% accuracy on RFC 2119 classification. However, the best model misclassifies approximately 10 out of 144 patterns, and the worst misclassifies 21. For compliance auditing, where each classification may determine whether a prohibited cipher suite is deployed or a mandatory requirement is overlooked, these error rates are concerning.

The model weights shown in Table 2 reflect each model’s contribution to the ensemble and are discussed in Section 4.4.

4.3 Pattern Difficulty Analysis

By looking at Figure 2, we can categorize patterns into two difficulty tiers based on cross-model agreement:

1. **All Models Correct** (100 patterns, 69.4%): Patterns with unambiguous textual guidance that all models classified correctly across all rounds. These typically feature explicit RFC 2119 keywords in clear syntactic contexts.
2. **Mixed Results** (44 patterns, 30.6%): Patterns where at least one model in at least one round disagreed with the ground truth. These often involve:

- Conditional requirements with complex nested conditions
- Implicit guidance requiring inference from surrounding context
- Subtle negations or discouragement phrasing (e.g., “not recommended” vs. “may be used in limited circumstances”)
- Context appearing in Appendix sections requiring broader document understanding

Notably, no patterns were universally misclassified by all models in all rounds, indicating that the ensemble approach captures at least one correct prediction for every pattern. However, this does not mean the ensemble corrects all errors—the ensemble still misclassifies 9 patterns (as discussed in Section 4.4).

Figure 3 visualizes the complete error distribution across all 44 cipher suite patterns showing which models made errors on specific patterns across different rounds.

4.3.1 Dominant Error: Over-Permissive Classification

Table 3 reveals a stark finding: 8 out of 10 SHOULD NOT patterns (80%) are misclassified as MAY, while every MUST NOT, MUST, and MAY pattern is correctly classified. This means the ensemble’s overall accuracy of 93.75% is entirely driven by correct classification of MAY and MUST NOT

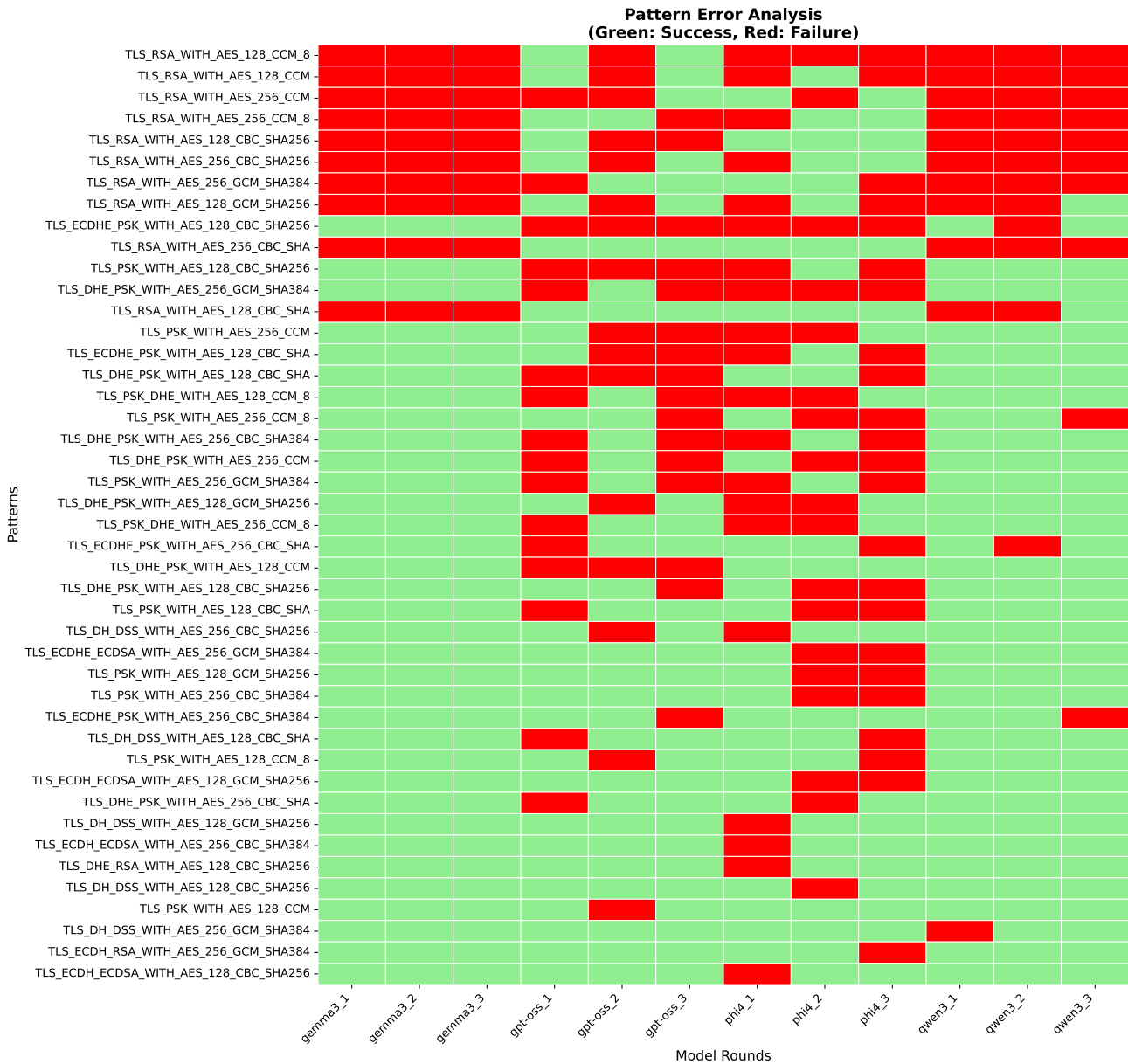


Fig. 3 Error matrix for the 44 patterns where at least one model made an error in at least one round.

Table 3 Confusion matrix for 4-model ensemble predictions (aggregated across all rounds). Rows show ground truth, columns show predicted labels.

Actual ↓ / Predicted →	MAY	MUST NOT	SHOULD NOT	MUST	Error Rate
MAY (84)	84	0	0	0	0%
MUST NOT (49)	0	49	0	0	0%
SHOULD NOT (10)	8	0	2	0	80%
MUST (1)	0	0	0	1	0%

patterns—the two classes that together comprise 92.3% of the dataset. On the only challenging class (SHOULD NOT), performance is near-random (20% accuracy, or 2 out of 10 correct).

This occurs when guidance phrases like “*not recommended but may be used in specific circumstances*” are inter-

preted as unconditional permission rather than conditional discouragement. For example:

“While these guidelines do not recommend cipher suites using RSA key transport, there may be circumstances in practice where it’s needed.”

Models correctly identify the permissive clause (“may be needed”) but underweight the discouragement (“do not recommend”), resulting in MAY instead of SHOULD NOT.

This finding directly addresses **RQ3**: LLMs systematically fail at the SHOULD NOT–MAY boundary, where conditional discouragement language creates persistent ambiguity that structured reasoning alone cannot resolve.

The six-step reasoning chain (defined in Section 3.2) enables precise diagnosis of where this failure occurs. Consider the pattern `TLS_RSA_WITH_AES_128_GCM_SHA256` (ground truth: SHOULD NOT). Three of four models classified it as MAY, with *gemma3* reasoning: “*The guideline does not recommend RSA key transport but lists specific cipher suites [...] that may be used temporarily during a transition period, implying permissive usage rather than a strong recommendation or prohibition.*” The model correctly executes Steps 1–3 (sentence identification, keyword extraction, negation detection), recognizing both the discouraging phrase “does not recommend” and the permissive clause “may be used.” However, at Step 4 (contextual analysis), it misjudges the overall language strength, treating the transitional allowance as the dominant signal. Consequently, at Step 5 (hierarchical decision tree), the model bypasses the higher-priority SHOULD NOT branch—which should match “not recommended”—and falls through to the MAY branch based on “may be used.” In contrast, *gpt-oss* correctly resolved the same passage, reasoning: “*The guidelines explicitly do not recommend RSA key transport [...] so the overall guidance is a conditional SHOULD NOT,*” demonstrating proper application of the priority ordering in Step 5.

This example illustrates both the value of structured reasoning chains for traceability and their fundamental limitation: providing a reasoning structure does not guarantee correct reasoning. All four models had access to the same reasoning template, yet three systematically misinterpreted the same passage. The structured chain reveals *where* reasoning goes wrong but does not prevent the error.

4.3.2 Model-Specific Error Patterns

Figure 4 decomposes errors by model and confusion type. *phi4* exhibits the highest error rate with a pronounced over-permissive bias (SHOULD NOT → MAY). *gemma3* shows the most balanced error distribution with the lowest error rate.

4.4 Ensemble Validation

The weighted ensemble achieved 93.75% accuracy, a 0.69 percentage point improvement over the best individual model (*gemma3* at 93.06%). Detailed analysis reveals that the ensemble corrected at least one model’s error in 25 patterns (17.4% of the dataset). Remarkably, *gemma3* was incorrect in

only 1 of these 25 patterns (4.0%), demonstrating its strong individual performance. In contrast, *gpt-oss* and *phi4* were incorrect in 14 (56.0%) and 16 (64.0%) of these patterns respectively, while *qwen3* failed in 2 patterns (8.0%).

The ensemble’s improvement over *gemma3* stems from correctly classifying 1 pattern that *gemma3* misclassified (`TLS_RSA_WITH_AES_128_CBC_SHA`), a SHOULD NOT requirement where conditional discouragement language was ambiguous. Specifically, the ensemble successfully leveraged complementary strengths: while *gemma3* over-interpreted the permissive clause as MAY, the ensemble’s aggregation of *qwen3*, *gpt-oss*, and *phi4* predictions provided corrective signals that restored the intended SHOULD NOT classification. This demonstrates that multi-model voting effectively mitigates model-specific blind spots, particularly for edge cases involving nuanced requirement distinctions.

4.4.1 Ablation Study: Ensemble without best performer model

To validate the robustness of our ensemble approach, we conducted an ablation study by excluding *gemma3* (the highest-accuracy model) and re-evaluating the ensemble with only the remaining three models (*qwen3*, *gpt-oss*, *phi4*). Results from this 3-model ensemble are as follows:

- **Model Weights:** *qwen3* (34.61%), *gpt-oss* (33.30%), *phi4* (32.09%)
- **3-Model Ensemble Accuracy:** 93.06% (134/144 correct)
- **Improvement over best in subset:** +0.69pp over *qwen3* (92.36%)

The 3-model ensemble achieved 93.06% accuracy, matching *gemma3*’s individual performance while maintaining the same 0.69 percentage point improvement margin over the best model in its subset (*qwen3*). Our takeaway from this ablation study is that our multi-model ensemble strategy provides value beyond simply deferring to the best individual model. Together, these results address **RQ2** where different LLMs exhibit varying accuracy, and combining their predictions through ensemble voting consistently improves over any single model, even when the best performer is removed.

4.4.2 Detailed Analysis: Weighted Voting Mechanism

To understand how the 3-model ensemble achieves its performance improvements, we analyzed the 23 patterns where the ensemble corrected at least one individual model’s error. Table 4 presents these patterns with each model’s accumulated weighted votes across all rounds, revealing the ensemble’s decision-making process.

The weighted voting mechanism demonstrates several key behaviors. First, the voting system aggregates predictions across all three rounds per model. Each model’s

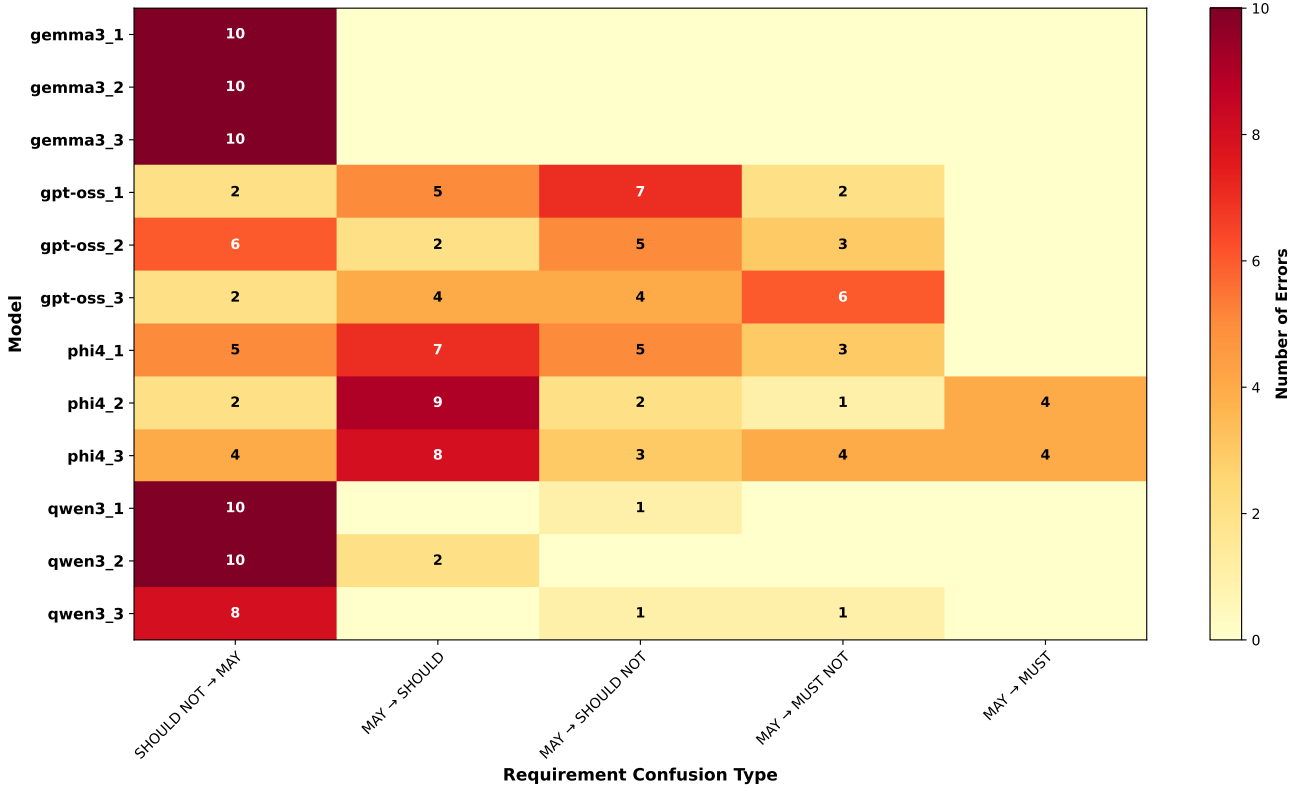


Fig. 4 Requirement prediction error breakdown by model.

contribution is calculated as its weight multiplied by the number of rounds in which it predicted that requirement. With model weights of approximately 0.346 (*qwen3*), 0.333 (*gpt-oss*), and 0.321 (*phi4*), the maximum contribution any single model can make is approximately 1.04, 1.00, and 0.96, respectively (weight $\times$ 3 rounds). For example, in TLS_PSK_WITH_AES_128_CBC_SHA256, *qwen3* contributed 1.04 votes to MAY by predicting it consistently across all 3 rounds.

Second, in 3 out of 23 correction patterns (highlighted in gray), the ensemble reached the correct answer despite the highest-weighted model (*qwen3* at 34.61%) voting incorrectly. For example, in TLS_PSK_WITH_AES_256_CCM_8, *qwen3* predicted MUST NOT (contributing 0.35 votes) while *gpt-oss* predicted MAY (contributing 0.67 votes) and *phi4* predicted MUST (contributing 0.32 votes). The ensemble correctly selected MAY because *gpt-oss*'s accumulated contribution (0.67) combined with other rounds created the winning vote total. This demonstrates that consistent predictions from lower-weighted models can collectively outweigh inconsistent predictions from higher-weighted models.

Third, when two or more models agree on the correct answer, their combined votes create a dominant signal. In TLS_RSA_WITH_AES_256_CBC_SHA, both *gpt-oss* (1.00 votes) and *phi4* (0.96 votes) correctly predicted SHOULD

NOT, while *qwen3* incorrectly predicted MAY (1.04 votes). The consensus between *gpt-oss* and *phi4* (combined 1.96 votes) decisively outweighed *qwen3*'s dissenting vote, ensuring the correct classification.

Fourth, the ensemble mitigates model-specific blind spots. For instance, *phi4* frequently confused conditional requirements (SHOULD NOT) with unconditional permission (MAY), evidenced by multiple patterns where *phi4* alone predicted SHOULD or MUST, while others correctly predicted MAY. However, the weighted voting mechanism prevented these systematic errors from propagating to the final ensemble prediction in all 23 analyzed cases.

Finally, the table values reveal how multi-round consistency strengthens a model's influence. Models that consistently predict the same requirement across all rounds contribute their maximum weight (e.g., 1.04 for *qwen3*), while inconsistent predictions result in split contributions across multiple requirements. This mechanism naturally rewards both accuracy (via model weights) and reproducibility (via multi-round consistency).

5 Limitations and Future Work

While our framework demonstrates measurable performance on TLS cipher suite classification, several limitations con-

Table 4 Patterns where 3-model ensemble corrected individual model errors. Values show each model’s prediction with its accumulated weighted vote contribution in parentheses (model weight $\times$ number of rounds predicting that requirement).

Pattern	qwen3	gpt-oss	phi-4	Ensemble
TLS_PSK_WITH_AES_128_CBC_SHA256	MAY (1.04)	SHOULD NOT (1.00)	MAY (0.32)	MAY
TLS_PSK_DHE_WITH_AES_128_CCM_8	MAY (1.04)	MUST NOT (0.33)	MUST (0.32)	MAY
TLS_DHE_PSK_WITH_AES_128_CBC_SHA	MAY (1.04)	SHOULD (0.33)	MAY (0.64)	MAY
TLS_DHE_PSK_WITH_AES_128_GCM_SHA256	MAY (1.04)	MAY (0.67)	SHOULD (0.64)	MAY
TLS_ECDHE_PSK_WITH_AES_256_CBC_SHA	MAY (0.69)	SHOULD NOT (0.33)	MAY (0.64)	MAY
TLS_DH_DSS_WITH_AES_128_CBC_SHA	MAY (1.04)	SHOULD NOT (0.33)	MAY (0.64)	MAY
TLS_DHE_PSK_WITH_AES_128_CBC_SHA256	MAY (1.04)	MAY (0.67)	SHOULD (0.64)	MAY
TLS_DH_DSS_WITH_AES_128_CBC_SHA256	MAY (1.04)	MAY (1.00)	SHOULD NOT (0.32)	MAY
TLS_ECDHE_ECDSA_WITH_AES_256_GCM_SHA384	MAY (1.04)	MAY (1.00)	SHOULD (0.64)	MAY
TLS_ECDH_ECDSA_WITH_AES_128_GCM_SHA256	MAY (1.04)	MAY (1.00)	SHOULD (0.32)	MAY
TLS_PSK_WITH_AES_256_CCM	MAY (1.04)	MAY (0.33)	MUST (0.32)	MAY
TLS_DHE_PSK_WITH_AES_256_CBC_SHA	MAY (1.04)	SHOULD (0.33)	SHOULD NOT (0.32)	MAY
TLS_DHE_PSK_WITH_AES_128_CCM	MAY (1.04)	SHOULD NOT (1.00)	MAY (0.96)	MAY
TLS_ECDHE_PSK_WITH_AES_256_CBC_SHA384	SHOULD NOT (0.35)	MAY (0.67)	MAY (0.96)	MAY
TLS_DHE_PSK_WITH_AES_256_CCM	MAY (1.04)	SHOULD NOT (0.33)	SHOULD (0.32)	MAY
TLS_PSK_WITH_AES_256_GCM_SHA384	MAY (1.04)	SHOULD NOT (0.33)	MAY (0.32)	MAY
TLS_PSK_WITH_AES_128_CBC_SHA	MAY (1.04)	SHOULD (0.33)	MUST (0.32)	MAY
TLS_PSK_WITH_AES_128_GCM_SHA256	MAY (1.04)	SHOULD (0.33)	SHOULD (0.64)	MAY
TLS_PSK_WITH_AES_256_CBC_SHA384	MAY (1.04)	MAY (1.00)	SHOULD (0.64)	MAY
TLS_PSK_WITH_AES_256_CCM_8	MUST NOT (0.35)	MAY (0.67)	MUST (0.32)	MAY
TLS_RSA_WITH_AES_256_CBC_SHA	MAY (1.04)	SHOULD NOT (1.00)	SHOULD NOT (0.96)	SHOULD NOT
TLS_DHE_PSK_WITH_AES_256_CBC_SHA384	MAY (1.04)	SHOULD NOT (0.33)	MAY (0.32)	MAY
TLS_PSK_DHE_WITH_AES_256_CCM_8	MAY (1.04)	MUST NOT (0.33)	MUST NOT (0.32)	MAY

strain the generalizability and practical applicability of these results:

- (i) **Evaluation on a single standard.** Our validation is conducted exclusively on NIST SP 800-52 Revision 2, a single technical standard with a specific structure, vocabulary, and organizational style. Although the framework’s three-phase architecture is designed to be document-agnostic, its effectiveness on standards with different organizational conventions (e.g., ISO, IEC, ETSI, BSI documents) remains unverified. These organizations use different phrasing patterns, cross-referencing conventions, and normative language styles that may interact differently with LLM reasoning. Future work should evaluate the framework across multiple standards from different organizations to assess generalizability.
- (ii) **Class imbalance and small dataset.** The evaluation dataset of 144 cipher suite patterns exhibits class imbalance where MAY (58.3%) and MUST NOT (34%) dominate, while SHOULD NOT accounts for only 6.9%, MUST for 0.7%, and SHOULD has zero instances in the ground truth. This prevents reliable assessment of framework performance on the underrepresented classes.
- (iii) **Deterministic retrieval limitations.** Phase 1 relies on deterministic regex matching of structured identifiers (cipher suite names) to retrieve relevant document contexts. This approach works well for standards organized around searchable technical identifiers but may not generalize to documents where requirements are expressed in free-form prose without such anchors. Extending the retrieval mechanism to support hybrid strategies—combining pattern matching with semantic retrieval or section-aware parsing—would broaden the framework’s applicability to less structured standards.
- (iv) **English-only evaluation.** All prompts, reasoning chains, and evaluated documents are in English. Many security standards are published in multiple languages (e.g., ANSSI guidelines in French, BSI standards in German), and normative language conventions may differ across languages. The cross-lingual robustness of LLMs for normative classification is entirely unexplored in this work.
- (v) **Error concentration at the SHOULD NOT–MAY boundary.** Our error analysis reveals that misclassifications concentrate at the boundary between SHOULD NOT and MAY, where conditional discouragement language creates systematic ambiguity. The structured reasoning chain, while providing useful traceability, does not resolve this confusion: all four models exhibited the same bias despite having identical reasoning templates. This suggests the limitation is in the models’ fundamental semantic understanding rather than in prompt design. Future work could explore: (a) fine-tuning models on RFC 2119 classification with augmented data that specif-

ically targets the SHOULD NOT–MAY boundary, (b) post-hoc rule-based correction layers that detect and flag conditional discouragement patterns, or (c) providing feedback to standardization bodies on how normative writing style could be improved to reduce ambiguity.

6 Conclusion

In this paper, we presented an evaluation of large language models for automated extraction and classification of RFC 2119 normative requirements from technical security standards, combining deterministic pattern-based retrieval, structured chain-of-thought prompting, and accuracy-weighted multi-model ensemble voting.

Evaluated on 144 TLS cipher suite patterns from NIST SP 800-52 Revision 2, the best individual model (*gemma3*) achieved 93.06% accuracy, while the 4-model ensemble reached 93.75%—a 0.69 pp improvement over the best individual model. An ablation study confirms this improvement is robust: removing the top performer from the ensemble still matched its individual accuracy. However, with 8 out of 144 patterns still misclassified, current LLMs remain insufficient for compliance-critical deployment. Error analysis identified the SHOULD NOT–MAY boundary as the dominant and persistent source of misclassification, where conditional discouragement language is systematically confused with unconditional permission across all models.

Our findings lead to the following main conclusions:

- I) **Current LLMs are not sufficiently reliable for standalone compliance classification.** The error rate of approximately 6% is orders of magnitude too high for security compliance contexts where every classification carries accountability. An organization relying solely on LLM-based classification would likely deploy discouraged cipher suites and fail to meet compliance requirements.
- II) **Ensemble methods improve accuracy but cannot bridge the compliance gap.** The weighted multi-model ensemble achieves a 0.69 pp improvement over the best individual model, and the ablation study confirms this gain is robust. Nevertheless, the ensemble’s 93.75% accuracy remains far below what compliance auditing demands.
- III) **Structured reasoning chains expose errors but do not prevent them.** The six-step reasoning chain provides valuable traceability for human auditors to identify where misclassifications originate. However, all four LLMs exhibited the same systematic bias at the SHOULD NOT–MAY boundary despite identical reasoning templates, indicating a fundamental limitation in semantic understanding that prompt design alone cannot resolve.

The structured reasoning chains provide the interpretability that security compliance auditing demands—each classification is accompanied by explicit justification grounded in RFC 2119 definitions, enabling human auditors to verify and trace automated decisions. However, transparency does not compensate for insufficient accuracy. We recommend that practitioners treat current LLM-based requirement classification as a *screening tool* that flags potential requirements for human review, rather than as a replacement for expert analysis. The SHOULD NOT–MAY confusion identified in this work should serve as a specific caution for any deployment targeting standards with conditional discouragement language.

Declarations

Funding The authors declare that no funds, grants, or other support were received during the preparation of this manuscript.

References

1. Arora, C., Grundy, J., Abdelrazek, M.: Advancing requirements engineering through generative ai: Assessing the role of llms pp. 129–148 (2024)
2. Boonstra, L.: Prompt engineering (2025)
3. Bradner, S.: Key words for use in rfcs to indicate requirement levels. Tech. Rep. RFC 2119, IETF (1997). URL <https://www.rfc-editor.org/rfc/rfc2119.html>
4. Brown, T., Mann, B., Ryder, N., Subbiah, M., Kaplan, J.D., Dhariwal, P., Neelakantan, A., Shyam, P., Sastry, G., Askell, A., et al.: Language models are few-shot learners. *Advances in neural information processing systems* **33**, 1877–1901 (2020)
5. Chen, J., Saha, S., Bansal, M.: Reconcile: Round-table conference improves reasoning via consensus among diverse llms. In: *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics*, pp. 7066–7085 (2024)
6. Dalpiaz, F., Niu, N.: Requirements engineering in the days of artificial intelligence. *IEEE Software* **37**(4), 7–10 (2020)
7. Dietterich, T.G.: Ensemble methods in machine learning. *International Workshop on Multiple Classifier Systems* pp. 1–15 (2000)
8. Du, Y., Li, S., Torralba, A., Tenenbaum, J.B., Mordatch, I.: Improving factuality and reasoning in language models through multiagent debate. In: *Forty-first international conference on machine learning* (2024)

9. Ferrari, A., Dell'Orletta, F., Esuli, A., Gervasi, V., Gnesi, S.: Natural language requirements processing: a 4d vision. *IEEE SOFTWARE* **34**(6), 28–35 (2017)
10. Gao, Y., Xiong, Y., Gao, X., Jia, K., Pan, J., Bi, Y., Dai, Y., Sun, J., Wang, H., Wang, H., et al.: Retrieval-augmented generation for large language models: A survey. *arXiv preprint arXiv:2312.10997* **2**(1), 32 (2023)
11. Geremia, R., Manfredi, S.: stfbk/tls-compliance-dataset (2025). DOI 10.5281/zenodo.15011610. <https://doi.org/10.5281/zenodo.15011610>
12. Geremia, R., Manfredi, S., Rizzi, M., Sciarretta, G., Tomasi, A., Ranise, S.: Automating compliance for improving tls security postures: An assessment of public administration endpoints. In: *Proceedings of the 21st International Conference on Security and Cryptography - SECRIPT*, pp. 450–458. INSTICC, SciTePress (2024). DOI 10.5220/0012764700003767
13. Hassani, S., Sabetzadeh, M., Amyot, D., Liao, J.: Rethinking legal compliance automation: Opportunities with large language models. In: *2024 IEEE 32nd International Requirements Engineering Conference (RE)*, pp. 432–440. IEEE (2024)
14. Jiang, D., Ren, X., Lin, B.Y.: Llm-blender: Ensembling large language models with pairwise ranking and generative fusion pp. 14165–14178 (2023)
15. Joint Task Force Transformation Initiative: Nist special publication 800-53b: Control baselines for information systems and organizations. Tech. Rep. NIST SP 800-53B, National Institute of Standards and Technology (2020). URL <https://pages.nist.gov/OSCAL/>. OSCAL (Open Security Controls Assessment Language) is the machine-readable framework for representing these controls
16. Khot, T., Trivedi, H., Finlayson, M., Fu, Y., Richardson, K., Clark, P., Sabharwal, A.: Decomposed prompting: A modular approach for solving complex tasks. *arXiv preprint arXiv:2210.02406* (2022)
17. Kiyavitskaya, N., Zeni, N., Mich, L., Berry, D.M.: Requirements for tools for ambiguity identification and measurement in natural language requirements specifications. *Requirements engineering* **13**(3), 207–239 (2008)
18. Kojima, T., Gu, S.S., Reid, M., Matsuo, Y., Iwasawa, Y.: Large language models are zero-shot reasoners. *Advances in neural information processing systems* **35**, 22199–22213 (2022)
19. Lewis, P., Perez, E., Piktus, A., Petroni, F., Karpukhin, V., Goyal, N., Küttler, H., Lewis, M., Yih, W.t., Rocktäschel, T., et al.: Retrieval-augmented generation for knowledge-intensive nlp tasks. *Advances in neural information processing systems* **33**, 9459–9474 (2020)
20. Li, J., Zhang, Q., Yu, Y., Fu, Q., Ye, D.: More agents is all you need. *arXiv preprint arXiv:2402.05120* (2024)
21. Manfredi, S., Ranise, S., Sciarretta, G.: Lost in tls? no more! assisted deployment of secure tls configurations. In: *IFIP Annual Conference on Data and Applications Security and Privacy*, pp. 201–220. Springer (2019)
22. Maxwell, J.C., Antón, A.I.: Developing production rule models to aid in acquiring requirements from legal texts. In: *2009 17th IEEE International Requirements Engineering Conference (RE)*, pp. 101–110. IEEE (2009). DOI 10.1109/RE.2009.21
23. Morante, R., Sporleder, C.: Modality and negation: An introduction to the special issue. *Computational linguistics* **38**(2), 223–260 (2012)
24. Narouei, M., Takabi, H., Nielsen, R.: Automatic extraction of access control policies from natural language documents. *IEEE Transactions on Dependable and Secure Computing* **17**(3), 506–517 (2018)
25. National Institute of Standards and Technology: Guidelines for the selection, configuration, and use of transport layer security (tls) implementations. Tech. Rep. Special Publication 800-52 Revision 2, NIST (2019). DOI 10.6028/NIST.SP.800-52r2
26. Reynolds, L., McDonnell, K.: Prompt programming for large language models: Beyond the few-shot paradigm. In: *Extended abstracts of the 2021 CHI conference on human factors in computing systems*, pp. 1–7 (2021)
27. Ronanki, K., Cabrero-Daniel, B., Horkoff, J., Berger, C.: Requirements engineering using generative ai: Prompts and prompting patterns. In: *Generative AI for effective software development*, pp. 109–127. Springer (2024)
28. Sancheti, A., Garimella, A., Srinivasan, B.V., Rudinger, R.: Agent-specific deontic modality detection in legal language. In: *Proceedings of the 2022 Conference on Empirical Methods in Natural Language Processing*, pp. 11563–11579 (2022)
29. Sleimi, A., Sannier, N., Sabetzadeh, M., Briand, L., Dann, J.: Automated extraction of semantic legal metadata using natural language processing. In: *2018 IEEE 26th International Requirements Engineering Conference (RE)*, pp. 124–135. IEEE (2018)
30. Touvron, H., Martin, L., Stone, K., Albert, P., Almahairi, A., Babaei, Y., Bashlykov, N., Batra, S., Bhargava, P., Bhosale, S., et al.: Llama 2: Open foundation and fine-tuned chat models. *arXiv preprint arXiv:2307.09288* (2023)
31. Wang, J., Wang, J., Athiwaratkun, B., Zhang, C., Zou, J.: Mixture-of-agents enhances large language model capabilities. *arXiv preprint arXiv:2406.04692* (2024)
32. Wang, X., Wei, J., Schuurmans, D., Le, Q., Chi, E., Narang, S., Chowdhery, A., Zhou, D.: Self-consistency improves chain of thought reasoning in language models. *arXiv preprint arXiv:2203.11171* (2022)
33. Wei, J., Wang, X., Schuurmans, D., Bosma, M., Xia, F., Chi, E., Le, Q.V., Zhou, D., et al.: Chain-of-thought

-
- prompting elicits reasoning in large language models. *Advances in neural information processing systems* **35**, 24824–24837 (2022)
34. Willard, B.T., Louf, R.: Efficient guided generation for large language models. *arXiv preprint arXiv:2307.09702* (2023)
35. Zhao, L., Alhoshan, W., Ferrari, A., Letsholo, K.J., Ajagbe, M.A., Chioasca, E.V., Batista-Navarro, R.T.: Natural language processing for requirements engineering: A systematic mapping study. *ACM Computing Surveys (CSUR)* **54**(3), 1–41 (2021)
36. Zhou, D., Schärli, N., Hou, L., Wei, J., Scales, N., Wang, X., Schuurmans, D., Cui, C., Bousquet, O., Le, Q., et al.: Least-to-most prompting enables complex reasoning in large language models. *arXiv preprint arXiv:2205.10625* (2022)