Machine Learning (2026) 115:116
https://doi.org/10.1007/s10994-026-07038-6

LLM-Driven Compliance Checking for Natural-Language Policies over Data Product Descriptors

Francesco Simbola[1] · **Diego Reforgiato Recupero**[1] · **Daniele Riboni**[1] · **Martina Salis**[1]

Received: 7 January 2026 / Revised: 13 March 2026 / Accepted: 25 March 2026 /
Published online: 11 May 2026
© The Author(s) 2026

Abstract
Modern data architectures increasingly rely on decentralized data products described through semi-structured artifacts, such as YAML descriptors, to support scalability, interoperability, and governance. Ensuring that these descriptors comply with organizational and regulatory policies remains a challenging and labor-intensive task, largely due to the natural-language nature of policies and the heterogeneity of descriptor schemas. In this paper, we investigate a model-based approach to evaluating natural-language policies over semi-structured data product descriptors expressed in YAML, leveraging Large Language Models (LLMs) as reasoning components. We formalize the policy-descriptor compliance task as a binary decision problem and propose an LLM-based microservice architecture that combines structured prompt engineering with deterministic output constraints to support automated policy verification. To evaluate the approach, we construct an augmentation-based benchmark comprising 4,000 policy-descriptor pairs derived from 40 manually annotated seed combinations. The benchmark is designed to assess model robustness to policy paraphrases and semantically equivalent descriptor rewrites. We evaluate four open-source instruction-tuned LLMs under identical conditions and report results in terms of accuracy, precision, recall, F1-score, and execution time. Experimental results show that LLMs can effectively support policy verification in semi-structured environments, achieving up to 72% accuracy. However, we observe a consistent conservative bias toward non-compliance, as well as systematic failure modes involving conjunctions, bidirectional constraints, and conditionally inapplicable rules. Our analysis highlights both the potential and the current limitations of LLM-driven governance, and suggests that expressing policies as explicit procedural checks substantially improves validation reliability.

Keywords Automated policy verification · Large Language Models (LLMs) · Semi-structured data · Prompt engineering

Francesco Simbola, Diego Reforgiato Recupero, Daniele Riboni and Martina Salis contributed equally to this work.

Editors: Lan Du, Benjamin C. M. Fung, Hady W. Lauw, Kalidas Yeturu, Longbing Cao.

Extended author information available on the last page of the article

1 Introduction

Organizations increasingly rely on data-driven processes to guide decision-making, ensure operational efficiency, and demonstrate compliance with internal and regulatory requirements. As data ecosystems grow in scale and heterogeneity, governance practices must keep pace with rapid changes in how data is produced, described, and consumed (Bernardo et al., 2024; Janssen et al., 2020). Modern data architectures, such as Data Mesh (Dehghani & Fowler, 2022), emphasize decentralized ownership of data products, each described through semi-structured artifacts, often in YAML, that capture metadata, access rules, quality constraints, and operational properties. Ensuring that these descriptors comply with organizational policies is critical for maintaining trust, interoperability, and regulatory alignment (Bena et al., 2025; Khatri & Brown, 2010).

However, verifying policy compliance remains a labor-intensive and error-prone task. Policies are typically written in natural language, which introduces ambiguity, inconsistent interpretation, and limited formal structure (Buthelezi et al., 2016). Translating such textual requirements into machine-verifiable constraints often requires domain expertise, manual mapping, and substantial engineering effort (Kiyavitskaya et al., 2007). This challenge is amplified by the diversity of descriptor schemas and the lack of standardized mechanisms to automatically align human-readable rules with machine-readable specifications. As a result, organizations struggle to maintain consistent governance across distributed data products, increasing operational risk and reducing transparency (Acev et al., 2025).

Recent advances in large language models (LLMs) offer new opportunities for addressing these challenges. LLMs have demonstrated strong capabilities in interpreting natural language instructions (OpenAI et al., 2024), manipulating structured and semi-structured data (Jaimovitch-López et al., 2023), and generating executable representations from textual descriptions (Khan, 2024). Prior research has shown promising results in policy interpretation (Ren et al., 2025), compliance checking (Sun et al., 2025), and natural-language querying over structured data (Bhathena et al., 2023). Nevertheless, existing approaches rarely address the specific setting of data product ecosystems, where descriptors vary widely in format and structure, and where policies must be evaluated across multiple heterogeneous YAML files. Moreover, to the best of our knowledge, no prior study has systematically investigated the situations and reasons why LLMs may fail in these settings. To address this gap, in this paper we introduce an LLM-based compliance-checking framework explicitly tailored to data product descriptors that aligns natural language policies with diverse semi-structured schemas through structured prompt design and deterministic output constraints. Our framework performs policy verification through a novel microservice pipeline, and is validated through a new benchmark designed to test robustness to both policy paraphrases and semantically equivalent descriptor variants. Our experimental evaluation enables an empirical analysis of recurring LLM failure modes for this task, including difficulties with conjunction reasoning and bidirectional constraints.

Specifically, our framework automatically evaluates natural-language policies against data product descriptors expressed in YAML. The system parses organizational policies, interprets the semantics of descriptor fields, and determines whether each descriptor satisfies the relevant requirements specified within the underlying policy. By combining natural-language understanding with semi-structured data processing, the approach reduces manual effort, enhances consistency, and enables scalable governance across decentralized data

environments. We evaluated our approach on an augmentation-based benchmark of 4,000 policy, descriptor pairs derived from 40 manually labeled seed combinations (10 policies $\times$ 4 descriptors). The benchmark is designed to test the robustness of LLM-based policy verification to paraphrases of policies and to semantically equivalent rewritings of YAML descriptors, rather than to measure generalization to entirely unseen policy families.

The main contributions of this paper are:

- a formalization of the problem of policy-descriptor compliance within data product ecosystems;
- a method leveraging LLMs to align unstructured policy text with heterogeneous YAML structures;
- a novel dataset of 4,000 policy, descriptor pairs that we publicly release;
- an empirical evaluation of multiple LLMs under a unified prompting protocol, including an ablation study on prompt design that quantifies the impact of structured reasoning guidance;
- an error taxonomy and qualitative analysis of recurrent LLM failure modes in this task, including conjunction handling, bidirectional constraints, scope ambiguity, and vacuous truth in conditional rules.

The remainder of this paper is organized as follows. Section 2 reviews related work. Section 3 introduces the proposed task and describes the dataset we constructed for evaluation, consisting of both policies and descriptors. Section 4 presents the architecture of the proposed system. Section 5 reports the experimental evaluation. Section 6 provides an error analysis of the four used LLMs in validating YAML descriptors against natural language policies. Finally, Sect. 7 concludes the paper and outlines directions for future research.

2 Related Work

The automatic verification of policy compliance has gained increasing attention due to the growing complexity of software systems and data management processes (Beach et al., 2024; Bernardo et al., 2024). Policies are statements, often expressed in natural language, that specify rules that must be enforced to ensure correct and compliant system behavior. However, natural language often introduces ambiguity, inconsistent interpretations, and difficulties in translating rules into actionable controls (Hosseini et al., 2021). As a result, policy assessment frequently requires manual review (Al-Turki et al., 2024). Furthermore, there is a lack of intelligent tools capable of automating policy verification, supporting users in data governance tasks, and guaranteeing compliance with regulatory requirements[1].

Policies are commonly used to validate semi-structured data formats such as JSON, XML, or CSV, including YAML descriptors (Jiang et al., 2017). In the context of data ecosystems, and particularly in Data Mesh architectures, a data product descriptor captures metadata, structural information, and access specifications of a data product (Dehghani & Fowler, 2022). These descriptors expose machine-readable interfaces for querying, governance, and access control, and may also be linked to services that automatically evaluate descriptors against specific organizational policies.

[1] https://www.relyance.ai/data-journeys/ai-governance-frameworks-existing-dspm-approaches-fall-short

Recent research has explored the use of intelligent systems, especially LLMs, to improve transparency, consistency, and scalability in the automated interpretation and enforcement of policies (Ren et al., 2025). A first line of work focuses on LLM-based policy compliance and enforcement. For example, Wang et al. (2025) propose LLM-AccessShield, a framework that interprets natural-language policies and translates them into executable enforcement rules for domain-specific applications, enabling dynamic and privacy-preserving access decisions. Other studies demonstrate that LLMs can reliably edit or manipulate structured and semi-structured documents with minimal prompting (Weber, 2024), emphasizing their strong pattern-matching abilities and the importance of well-structured task formulations to reduce hallucinations.

The combination of LLMs with structured representations has also been applied to compliance checking in domains such as Building Information Modeling (BIM). Chen et al. (2024) integrate LLMs, deep learning, and ontologies to interpret normative requirements and apply them to semi-structured BIM data, showing how language models can support rule formalization and semantic disambiguation. Similarly, Ying and Sacks (2024) propose an agent-based system for generic LLM-driven compliance verification in BIM contexts. In parallel, Liu et al. (2023) develop GPT-ACC, a method for automatic regulatory compliance checking in the Architecture, Engineering, and Construction (AECO) domain, demonstrating how structured prompts can guide LLMs to generate coherent and interpretable rule assessments.

A second research direction investigates systems that process natural-language queries over semi-structured data. Foundational work by Abiteboul (1997) introduced early query models for semi-structured data, inspiring subsequent approaches for natural-language-driven information extraction. Ibba et al. (2025) generate executable queries from natural-language questions to retrieve data from semi-structured sources. Shen et al. (2022) propose semiPQA, a question-answering framework over JSON product data for customer query resolution. Other works bridge natural language with structured retrieval, such as Sondhi and Zhai (2014), who map semi-structured information to relational schemas for SQL-based querying, and Waltinger et al. (2013), who enable natural-language access to industrial semi-structured datasets. More recently, Gupta et al. (2023) introduce TEMPTABQA, a benchmark designed to evaluate temporal reasoning over semi-structured tabular data.

Taken together, these research threads highlight the increasing importance of integrating natural language understanding, semi-structured data processing, and automated policy interpretation.

However, to the best of our knowledge, prior work does not explicitly target the automated evaluation of natural-language governance policies against heterogeneous YAML-based data product descriptors in Data Mesh-like ecosystems, nor does it provide an empirical analysis of LLM failure modes in this specific setting. Our work bridges this gap by leveraging LLMs to interpret organizational policies and automatically verify their compliance across different YAML descriptors, enabling a unified and scalable solution for policy checking in semi-structured environments.

3 Background

In this section, we describe the task addressed in this paper and the dataset we created to conduct the evaluation.

3.1 The Proposed Task

The proposed task addresses the automatic verification of data product descriptors against a predefined set of organizational policies. Descriptors, expressed in YAML format, capture the technical and structural characteristics of data products. The verification process leverages an LLM to interpret the descriptor and assess its compliance with the specified constraints in a flexible, context-aware manner, moving beyond rigid syntactic rule matching.

Compliance is formulated as a binary decision problem: given a policy and a descriptor, the policy is either satisfied or violated. In cases of non-compliance, the task additionally requires (i) identifying the reason for the violation and (ii) indicating the relevant part(s) of the descriptor involved in the violation (when identifiable).

Overall, this task investigates the feasibility of combining policy-driven validation with LLM-based reasoning to enhance transparency, governance, and quality assurance throughout the lifecycle of data products.

In this work, we evaluated the binary compliance decision, leaving the assessment of graded compliance levels or partial satisfaction to future work.

3.2 Dataset

The dataset used in the evaluation consists of a set of policies and a set of descriptors. It includes 100 policies and 40 descriptors[2], yielding 4,000 policy-descriptor pairs. The descriptors represent real-world data product specifications in the business domain, with a focus on financial transactions, customer profiles, marketing campaigns, and sales records.

In this work, a *descriptor* (also referred to as a *data product descriptor*) is a reusable and self-contained unit of data and metadata designed to deliver value through standardized and discoverable interfaces. In decentralized data architectures, data products are typically owned by a domain, addressable, reliable, and intended for consumption by other domains or systems.

The main components of a descriptor are:

- Metadata: fields such as identifier, name, type, environment, domain, and description.
- Components: modular elements (e.g., `outputPort`, `workload`) describing how the data product is produced and exposed.
- Data contract: specifications (often within `outputPort` components) defining the schema, including field names, data types, and semantic descriptions.
- Interfaces: mechanisms for exposing or consuming data (e.g., APIs or message brokers).
- Ownership and governance: domain affiliation and, when applicable, access control information.

[2] Publicly available at https://github.com/francescosimbola/Policies-and-Descriptors/tree/main

```yaml
dataProductOwnerDisplayName: Mario Rossi
projectOwner: user:mario.rossi.italia.it
projectOwnerDisplayName: Mario Rossi
environment: development
domain: finance
projectKind: system
kind: dataproduct
domainId: urn:dmb:dmn:finance
id: urn:dmb:dp:finance:cashflow:0
description: This represents all the operating cashflows
    generating incoming or outgoing liquidity
devGroup: datameshplatform
ownerGroup: mario.rossi_italia.it
dataProductOwner: user:mario.rossi_italia.it
email: mario.rossi@italia.it
version: 0.13.0
fullyQualifiedName: Liquidity Cash Flows
name: CashFlow
informationSLA: null
maturity: Strategic
useCaseTemplateId: urn:dmb:utm:dataproduct-template:0.0.0
infrastructureTemplateId: urn:dmb:itm:cdp-aws-dataproduct-no-name
    :1
billing: {}
tags:
  - GDPR
specific: {}
business_attributes:
  valueProposition: Cash flow records related to customer
      purchases.
  okr: Increase the CLV
  pricingType: Pay Per Use
  stakeholderRoles:
    - CRO
    - CFO
security_attributes:
  visibility: Global
  confidentiality: Confidential
  gdpr: Yes
consumable: false
shoppable: false
taxonomy:
  id: urn:dmb:taxonomy:data-mesh-taxonomy
  name: data-mesh-taxonomy
components:
  - kind: workload
    id: urn:dmb:cmp:finance:cashflow:0:deletion-workload
    description: "Procedure to delete data according to the right
        to be forgotten"
    name: Data Deletion Workload
    technology: DBT
    platform: Azure
    version: 1.0.0
    infrastructureTemplateId: urn:dmb:itm:gdpr-tech-adapter:1
    useCaseTemplateId: urn:dmb:utm:gdpr-workload-template:1
    consumable: false
    shoppable: false
    specific: {}
    dependsOn: []
    tags: []
  - kind: outputport
    id: urn:dmb:cmp:finance:cashflow:0:cashflow-outputpor
    description: A Kafka based Output Port with data contract
    name: CashFlow OutputPort
    fullyQualifiedName: null
    version: 0.0.0
```

Listing 1 Example of a descriptor

Listing 1 illustrates an example descriptor. At the top level, it includes core fields such as `id`, `name`, `description`, `domain`, and `environment`. Additional attributes (e.g., `tags`, `maturity`, `taxonomy`) provide further semantic characterization. The lower part defines the components: `workload` components specify planning and orchestration of operational tasks, whereas `outputPort` components govern data exposure to consumers through structured and annotated interfaces, including (when present) data contracts and schema definitions.

A policy is a short natural-language statement (on average $\sim$50 words) that expresses a precise and verifiable technical requirement. For the sake of this work, we assume that natural language policies are not ambiguous, thereby avoiding uncertainty in compliance assessment for each policy, descriptor pair.

The dataset was expanded by systematically increasing the variability of both components to evaluate model robustness to structural and syntactic permutations. Starting from a set of seed policies and descriptors, we applied controlled semantic and structural augmentation procedures, followed by a full combinatorial pairing of the resulting variants. The compliance labels of the original pairs were then deterministically propagated to all derived combinations.

Specifically, the variants were generated as follows:

- Policy Augmentation: Each of the 10 seed policies was rewritten into nine semantically equivalent variants, yielding 100 policies in total. Augmentation introduced syntactic diversity by modifying sentence structure (e.g., active-passive transformations), replacing terms with domain-appropriate synonyms, and varying verbosity, while strictly preserving the original logical constraints and compliance thresholds.
- Descriptor Augmentation: Each of the 4 seed descriptors was expanded into ten structural variants, producing 40 distinct YAML files. These variants were created through semantically neutral structural perturbations, including reordering independent keys across nesting levels, inserting optional metadata or irrelevant tags, and modifying string formatting. The core schema, declared components, and policy-targeted attributes remained logically invariant.

This controlled expansion produced a fully crossed matrix of $100 \times 40 = 4{,}000$ policy-descriptor combinations. Because the augmentation procedures introduced only surface-level syntactic and structural variations without altering the underlying properties, the original semantics of each policy and descriptor were preserved. Consequently, the compliance outcome of every generated pair exactly mirrors that of its originating seed pair. Each combination therefore deterministically inherits its *True* or *False* label, ensuring a rigorous and unambiguous ground truth across the entire benchmark. Sensitive information (e.g., ownership data) was anonymized throughout the expansion process.

The dataset is intentionally designed to evaluate robustness to surface-level variability (policy paraphrases and descriptor rewrites) under semantic invariance, rather than open-world generalization across unrelated policy families. The following block reports the 10 original policies used as the seed set:

Policies

1) To comply, the data product must implement at least three data quality checks using the Great Expectations framework.
2) Before performing any validation, extract and list all schema fields for each data contract in the descriptor. Each field MUST contain: - 'name' (optional) - 'dataType' (optional) - 'description' (required and must be non-empty) - 'constraint' (optional) Validation rules:
A. If a 'description' field is explicitly present, assume it is correctly formatted.
B. Every schema field must contain a description field. If the key is missing, report an error.
C. If a description is present but empty, return an error only if the JSON explicitly has '"description": ""'.
D. Validate only on the extracted schema fields-DO NOT infer missing values or assume defaults.
E. If description is explicitly present and non-empty, assume it is correctly formatted. If all required fields are present, return 'satisfiesPolicy = true'. Otherwise, return 'satisfiesPolicy = false' with missing fields listed.
3) Column tags must contain at least one 'businessTerms' coherent with the domain and the product's purpose; in suggestions, list all terms found and their count.
4) Any data product containing PII must be tagged GDPR at product level.
5) If the product has a GDPR tag, at least one column must be PII-tagged; conversely, if any column has a PII tag, the product must have GDPR at DP level.
6) GDPR-tagged data products must include at least one implemented data deletion procedure for the right to be forgotten.
7) At least one data contract must define a Service Level Agreement (SLA).
8) The data product must use Airflow for orchestration and, after each ETL that refreshes data contracts, invoke the data contract guardian routine.
9) At the end of the Airflow DAG, publish the Data Product refreshed event on the shared event bus.
10) Any schema field that represents PII must be tagged with FQN = PII.

4 Architecture and Methods

The architecture shown in Fig. 1 of the proposed microservice for policy verification is composed of three main components: the front-end, the back-end, and the LLM responsible for evaluation. The interaction begins on the user side, through the front-end interface, where the two required inputs are provided: the policy, written in plain text, and the descriptor of a data product, expressed in YAML format. These inputs are submitted to the back-end via a REST request, which serves as the system's entry point. The back-end first validates the input and parses the descriptor from YAML into JSON, in order to simplify processing and enable the LLM to more clearly interpret the structure and internal relationships of the document. The policy and the converted descriptor are then combined with a set of carefully designed instructions that guide the model by specifying how to evaluate the correspondence between the policy requirements and the descriptor fields, as well as the exact format expected for the response. The integration of these elements results in a single prompt,

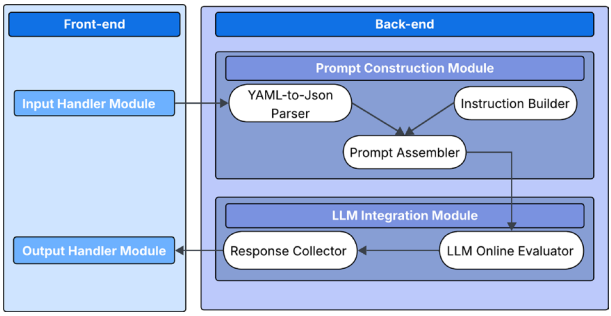


Fig. 1 Detailed architecture of the proposed policy evaluation microservice, highlighting the front-end modules for input and output handling, and the back-end modules responsible for prompt construction and LLM integration

which is sent via REST call to the LLM. Leveraging its contextual reasoning capabilities, the LLM performs the evaluation task and produces a structured output. This output is subsequently interpreted by the back-end, which normalizes it into a clear result, indicating whether the policy is satisfied or not and, in case of non-compliance, providing the detected violations along with an explanatory rationale. Finally, the result is returned to the front-end, where it is displayed to the user in a human-readable form, explicitly highlighting whether the policy is respected (true) or not (false), together with additional details on the identified issues. In the following, we will describe in detail each module. The architecture is designed to prioritize modularity, reproducibility, and deterministic LLM interaction rather than architectural novelty, ensuring practical integration within governance pipelines.

4.1 Front-end

The front-end acts as the user-facing layer of the system and is composed of two main modules, as shown in Fig. 1: the Input Handler Module and the Output Handler Module. The Input Handler Module is responsible for receiving user inputs (the plain-text policy and the YAML descriptor of the data product), performing preliminary validation to detect empty or malformed values, and forwarding the request to the back-end via a REST call. On the other hand, the Output Handler Module processes the results returned by the back-end and displays them in a user-friendly manner. It clearly indicates whether the policy is satisfied, and in cases of non-compliance, it highlights the associated explanation, helping the user to identify which elements of the descriptor failed to meet the policy requirements. This modular structure ensures that the front-end remains lightweight and focused on input reliability and output clarity, while delegating all evaluation logic to the back-end.

4.2 Back-end

The back-end implements the core logic of policy evaluation and is organized into two major components: the Prompt Construction Module and the LLM Integration Module, both represented in Fig. 1. The Prompt Construction Module is responsible for building the input that will be sent to the LLM. It consists of three subcomponents: the YAML-to-JSON Parser, which converts the descriptor into a normalized JSON format; the Instruction Builder, which generates task-specific directives that specify how the policy should be evaluated against the descriptor and how the answer should be structured; and the Prompt

Assembler, which merges the policy, the normalized descriptor, and the instructions into a single, unified prompt.

We created a prompt template that includes a general **small context**, the **policy**, the **descriptor** in JSON format, and **specific instructions** for the model to follow. The instructions guide the model to check for compliance and provide structured feedback on detected violations. The prompt constrains the model's output to a fixed JSON format, enabling reliable parsing by the back-end. Specifically, the model is instructed to return only a JSON object containing: (i) a boolean field indicating whether the descriptor satisfies the policy, (ii) a suggestion describing how to revise the descriptor to achieve compliance, and (iii) a list of error labels when violations are detected.

Beyond combining the policy, the normalized descriptor, and the output schema, the Prompt Construction Module embeds a structured evaluation protocol that guides the LLM through the compliance decision. Rather than relying on a single generic instruction, the prompt decomposes the task into a sequence of seven stages: policy reading, rule canonicalization, scope resolution, evidence localization, requirement evaluation, final decision, and constrained output generation. This decomposition was designed to reduce ambiguity when policies refer to heterogeneous descriptor elements distributed across multiple nesting levels. In particular, the intermediate stages explicitly guide the model in translating natural-language requirements into operational checks, identifying the relevant scope within the descriptor, and isolating the evidence to be used for validation. As discussed later in Sect. 5.2, these stages are not merely implementation details, but key contributors to the overall performance of the system. The prompt is shown in the following block.

```
small_context = """
    Goal: Decide if the descriptor satisfies the policy. Be
     ↪  deterministic, vendor-agnostic, and concise.

    Normalization & safety:
    - Compare keys case-insensitively; map '-', '_' and spaces to a
     ↪  single separator.
    - Trim strings. Consider empty/whitespace-only strings as empty.
    - NON-EMPTY object: at least one field is non-null and non-empty
     ↪  (recursively).
    - Numeric operations: only on numeric values; attempt safe
     ↪  conversion from strings, else type_mismatch.

    Counting, dedup, scope:

    - Unit of counting = smallest coherent JSON object matching the
     ↪  requirement subject.
    - Deduplicate by (unit path + salient subkeys) to avoid double
     ↪  counting.
    - If scope is unspecified, evaluate at product/root first; if
     ↪  inconclusive, extend to all sub-units/components. Record
     ↪  ambiguity if relevant.

    Ignore irrelevant noise:
    - Ignore labels/notes/long descriptions unless the requirement
     ↪  explicitly targets them.

    Errors (use these labels):
    - subject_not_found, ambiguous_scope, placeholder_only,
     ↪  type_mismatch, insufficient_evidence, parse_failure.
"""
descriptor_validation = """
    **Descriptor (JSON to evaluate)**
    content:
    {{content}}

    Descriptor processing:
    - Build a normalized index (path -> value) for all objects and
     ↪  arrays.
    - Identify evidence candidates by matching subject terms to
     ↪  normalized keys and object shapes.
    - Treat present-but-empty objects as placeholders (NOT NON-EMPTY).
    - For lists, each element can be a counting unit if it independently
     ↪  matches the subject.
"""
policy_validation = """
    **Policy**
    The policy states:
    {{policy}}

    Strict evaluation pipeline:

    Step 1 - ReadPolicy
    - Split into atomic requirements. Extract quantities/thresholds and
     ↪  logical connectives (ALL/ANY). Note any explicit scope.

    Step 2 - Canonize
    - Canonical record per requirement:
    {subject_terms[], operator, threshold?, quantifier?, logic(ALL|ANY),
     ↪  scope_hint?, exceptions?}
    - Operators: EXISTS, NOT_EXISTS, NON_EMPTY, ==, !=, >=, <=,
     ↪  CONTAINS, MATCHES.
```

```
    - Quantifiers: ANY, AT_LEAST_N, EXACTLY_N, ALL.
    - Heuristics:
    - "must define/include/provide" -> EXISTS (+ optionally NON_EMPTY).
    - "must not" -> NOT_EXISTS.
    - "at least N/>=N/one or more/any" -> AT_LEAST_N with N (default 1
     ↪  if implied).
    - "exactly N" -> EXACTLY_N.

    Step 3 - SetScope
    - If scope_hint is absent, start at product/root; else use
     ↪  scope_hint.
    - If still ambiguous, extend to all plausible sub-units/components
     ↪  that could host the subject. Record ambiguous_scope if multiple
     ↪  interpretations exist.

    Step 4 - LocateEvidence
    - Use the descriptor index to collect candidate units for each
     ↪  requirement.
    - Ignore noisy/descriptive-only sections unless directly targeted by
     ↪  the subject_terms[].

    Step 5 - Evaluate
    - EXISTS: satisfied if >=1 matching unit exists.
    - NOT_EXISTS: satisfied if no matching unit exists.
    - NON_EMPTY: satisfied if >=1 unit is NON-EMPTY (not
     ↪  placeholder_only).
    - ==, !=, >=, <=, CONTAINS, MATCHES: evaluate per unit; for numeric
     ↪  ops, require numeric values or mark type_mismatch.
    - Apply quantifiers and logic (ALL/ANY) over deduplicated units.

    Step 6 - Decide
    - If any requirement is violated -> satisfiesPolicy=false.
    - If all are satisfied -> satisfiesPolicy=true.
    - If unresolved indeterminacy prevents a safe positive decision ->
     ↪  satisfiesPolicy=false and list concise reasons.

    Step 7 - Output
    - Return ONLY a JSON object in the exact format specified below. No
     ↪  extra text.
"""

json_response = """
    Create an answer that is a JSON structured like this.
    YOU MUST RETURN ONLY THIS JSON, no other information.
    The structure is fixed, and all fields are mandatory:

    {
    "satisfiesPolicy": true,
    "details": {
     "suggestion": ""
     },
     "errors": []
     }
 """
```

Once the prompt is assembled, it is processed by the LLM Integration Module. This module contains the LLM Online Evaluator, which communicates with the external model through a REST API call and performs the actual compliance assessment, and the Response Collector, which gathers and normalizes the model's output into a structured result. This result includes a boolean value indicating whether the policy is satisfied, along with potential errors and explanations in cases of non-compliance. The normalized output is then returned to the front-end for visualization. By clearly separating prompt construction from

model integration, the back-end ensures modularity, robustness, and scalability while relying on the LLM for contextual reasoning.

4.3 Software Architecture Description

Following established guidelines for software architecture documentation (Górski, 2026), we describe the system using both structural and behavioral architectural views. The structural perspective is presented through a component-based decomposition of the system, while the behavioral perspective illustrates the runtime interaction between the main software components.

Figure 2 presents the component-based architectural view of the proposed policy verification microservice. Each module encapsulates specific responsibilities within the policy verification workflow. The **InputHandler** class manages incoming requests from the user interface and performs initial validation of the policy and descriptor inputs. The **YamlToJsonParser** converts YAML descriptors into a normalized JSON structure that can be more easily processed by the downstream components. The **InstructionBuilder** generates the evaluation directives used to guide the LLM during policy verification, while the **PromptAssembler** combines the policy text, the normalized descriptor, and the generated instructions into a structured prompt that is submitted to the model. Within the LLM Integration Module, the **LLMEvaluator** is responsible for communicating with the external LLM service through REST API calls and executing the compliance assessment. The **ResponseCollector** processes and normalizes the model's output before forwarding the result to the front-end layer. Finally, the **OutputHandler** formats the result and presents the compliance decision and related explanations to the user. This component-based decomposition promotes modularity, separation of concerns, and extensibility of the policy verification pipeline.

To complement the structural view, Fig. 3 illustrates the runtime interaction between the main system components during the policy verification process. The sequence diagram shows how a user request propagates through the system: the descriptor is first parsed into JSON format, the evaluation instructions are generated, and the prompt is assembled and submitted to the LLM. The response returned by the model is then collected, normalized, and formatted before the final result is delivered to the user. This behavioral view clarifies the execution flow of the system and highlights the coordination between the front-end, back-end components, and the external LLM service.

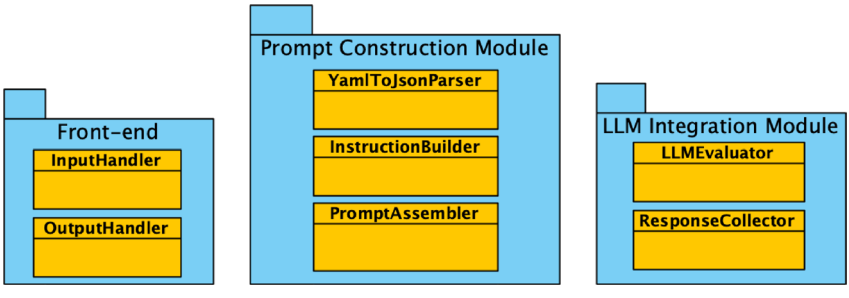


Fig. 2 Component diagram of the proposed policy verification microservice, showing the main architectural modules and their key classes

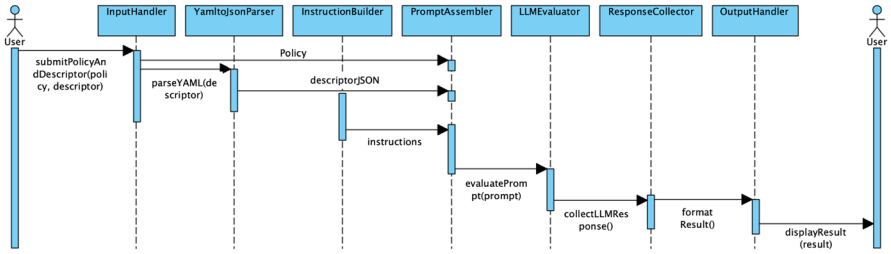


Fig. 3 Sequence diagram illustrating the runtime workflow of the policy verification process. The user submits a policy and a YAML descriptor, which are processed through descriptor parsing, instruction generation, prompt assembly, LLM-based evaluation, and response formatting before the final result is returned to the user

5 Experimental Evaluation

In this section, we present the experimental evaluation of the proposed system for policy verification. The evaluation aims to assess the system's effectiveness in accurately determining compliance between policies and data product descriptors.

5.1 Experimental Setup

The tests were conducted using a dataset of 100 policies and 40 descriptors, resulting in a total of 4,000 policy-descriptor combinations, as already mentioned in Sect. 3.2. We evaluated different LLMs to process the prompt. We considered 4 open-source models that can be efficiently executed on mid-range workstations.

We aimed to evaluate a variegated set of widely used LLM alternatives, in order to consider the heterogeneity of models that practitioners may realistically consider in data-governance deployments. Hence, we did not restrict the comparison to models with identical design objectives, but included both general-purpose instruction-tuned models (Mistral Instruct 7B, Granite 3.2 8B) and reasoning-oriented models (Qwen DeepSeek 8B, Mistral Instruct 12B).

For each model, we evaluate the performance using the standard metrics of accuracy, precision, recall, and F1-score, where the latter is the harmonic mean of precision and recall. We consider the task as a binary classification problem where, for each <policy, descriptor> pair, the label is 'True' when the descriptor satisfies the policy, and it is 'False' in the other case. The dataset is slightly unbalanced: indeed, 2, 100 instances belong to the 'True' class, while the remaining 1, 900 instances belong to the 'False' class. All models were executed locally using the same inference back-end (LM Studio/llama.cpp) with deterministic decoding (temperature = 0, top-p = 1). All experiments were conducted using a proprietary prototype developed within the DAMPAI project.[3]

[3] Due to intellectual property restrictions associated with the industrial collaboration of the DAMPAI project, the full software implementation cannot currently be released publicly. The prototype can however be made available for research purposes upon reasonable request to the authors.

5.2 Ablation Study on Prompt Design

To assess the contribution of the main reasoning-oriented components of our prompt, we conducted an ablation study on the structured evaluation protocol introduced in Sect. 4. The goal was to determine whether the intermediate guidance stages embedded in the prompt effectively improve compliance classification, beyond the basic task description and output-format constraints.

For this analysis, we used ***DeepSeek-R1-0528-Qwen3-8B***, i.e., the best-performing model in the main evaluation, and re-ran the full benchmark of 4,000 policy–descriptor pairs under different prompt configurations. We started from the **Full Structured Prompt**, which includes the entire evaluation pipeline, and then produced three ablated variants by removing one reasoning component at a time while leaving the rest of the prompt unchanged.

The considered variants are the following:

- Full Structured Prompt: the complete prompt, including all stages of the evaluation pipeline.
- Without Rule Canonicalization: removes the stage that instructs the model to translate policy statements into an internal representation based on operators, thresholds, quantifiers, and logical constraints.
- Without Scope Resolution: removes the stage that explicitly guides the model in identifying the appropriate hierarchical scope within the descriptor (e.g., root level, component level, or nested substructure).
- Without Evidence Localization: removes the stage that directs the model to locate the relevant evidence in the descriptor and to ignore semantically irrelevant descriptive content.

In all ablated configurations, the initial policy-reading stage, the final requirement-evaluation stage, and the constrained JSON output instructions were preserved. This choice ensures that all variants remain executable within the same microservice pipeline and that performance differences can be attributed to the removed reasoning component rather than to output-format failures.

Table 1 reports the results. The **Full Structured Prompt** achieves the best overall performance, with 72% accuracy and 72% weighted F1-score. Removing **Rule Canonicalization** leads to a moderate degradation, reducing the weighted F1-score to 68%. This suggests that explicitly reformulating policy statements into a more operational internal structure helps the model apply the intended constraints more consistently.

Table 1 Ablation study results for the prompt design using DeepSeek-R1-0528-Qwen3-8B

Prompt Configuration	Accuracy (%)	Weighted Precision (%)	Weighted Recall (%)	Weighted F1 (%)
Full Structured Prompt	**72**	**73**	**72**	**72**
Without Rule Canonicalization	70	76	70	68
Without Scope Resolution	68	71	68	67
Without Evidence Localization	68	72	68	67

Best results are highlighted in bold

The largest drop is observed when removing either **Scope Resolution** or **Evidence Localization**. In both cases, accuracy decreases to 68% and weighted F1-score to 67%. These results indicate that a substantial part of the task difficulty lies not only in understanding the policy itself, but also in identifying where the relevant evidence is located within the hierarchical YAML structure. Without explicit guidance on scope and evidence selection, the model is more likely to associate policy conditions with irrelevant or incomplete portions of the descriptor.

Overall, the ablation study shows that the effectiveness of the proposed approach depends not only on the LLM itself, but also on the procedural structure of the prompt. In particular, the results support the view that prompt stages devoted to rule formalization, scope disambiguation, and evidence localization play a significant role in making LLM-based compliance checking more reliable and deterministic.

5.3 LLM Results

The first LLM that we evaluated is the ***Mistral-7b-instruct-v0.3*** (AI, 2024a)[4] model with 7 billion parameters. *Mistral-7b-instruct-v0.3* is a fine-tuned version of the Mistral 7B model, optimized for instruction-following tasks. The evaluation results for the Mistral-7B-instruct-v0.3 model are presented in Table 2. Overall, the model achieves 59% accuracy. It obtains better results in the recognition of the majority class 'True' (77% recall, 58% precision), and lower results for the minority class (39% recall, 61% precision). Most errors of this model are false positives; i.e., 'False' class instances incorrectly classified as 'True'. Its weighted average F1-Score is 58%.

The second LLM that we evaluated is ***DeepSeek-R1-0528-Qwen3-8B*** (Team, 2024)[5], another model with 8 billion parameters. *DeepSeek-R1-0528-Qwen3-8B* is a fine-tuned version of the Qwen 8B model, optimized for instruction-following tasks. The results obtained with *DeepSeek-R1-0528-Qwen3-8B* are reported in Table 3. This model achieves significantly higher accuracy than the previous one. Indeed, DeepSeek-R1-0528-Qwen3-8B obtains 72% accuracy. Its performance on the two classes is well balanced, since it achieves 71% F1-Score for the majority class, and 73% F1-Score for the minority class. Most errors produced by this model are false negatives; i.e., 'True' class instances incorrectly classified as 'False'. Its weighted average F1-Score is 72%.

Table 2 Results of Mistral Instruct 7B model

		Mistral Instruct 7B		
		Pred True		Pred False
Exp True		1,614 (TP)		486 (FN)
Exp False		1,151 (FP)		749 (TN)
	Precision (%)	Recall (%)	F1-Score (%)	Support
Class 'True'	58	77	66	2,100
Class 'False'	61	39	48	1,900
Accuracy	Weighted Average Precision (%)	Weighted Average Recall (%)	Weighted Average F1 (%)	
59%	59	59	58	

[4] https://huggingface.co/lmstudio-community/Mistral-7B-Instruct-v0.3-GGUF

[5] https://lmstudio.ai/models/deepseek/deepseek-r1-0528-qwen3-8b

Table 3 Results of Qwen DeepSeek model

		DeepSeek-R1-0528-Qwen3-8B		
		Pred True		Pred False
Exp True		1,384 (TP)		716 (FN)
Exp False		403 (FP)		1,497 (TN)
	Precision (%)	Recall (%)	F1-Score (%)	Support
Class 'True'	77	66	71	2,100
Class 'False'	68	79	73	1,900
Accuracy	Weighted Average Precision (%)	Weighted Average Recall (%)	Weighted Average F1 (%)	
72%	73	72	72	

Table 4 Results of Granite 3.2 model

		Granite 3.2 8B		
		Pred True		Pred False
Exp True		1,305 (TP)		795 (FN)
Exp False		848 (FP)		1,052 (TN)
	Precision (%)	Recall (%)	F1-Score (%)	Support
Class 'True'	61	62	61	2,100
Class 'False'	57	55	56	1,900
Accuracy	Weighted Average Precision (%)	Weighted Average Recall (%)	Weighted Average F1 (%)	
59%	59	59	59	

The third LLM we evaluated is ***Granite 3.2 8B*** (Research, 2024)[6], an 8B-parameter instruction-tuned model designed for reliable adherence to prompts and structured outputs. Using this model, we obtained the confusion matrix and results reported in Table 4. This model achieves 59% accuracy, which is only slightly better than the one obtained by Mistral Instruct 7B, and significantly lower than the one of Qwen. The Granite performance on the two classes is rather balanced, achieving 61% F1-Score for the majority class, and 56% F1-Score for the minority class. The error types produced by this model are well distributed between false positives (848 instances) and false negative (795 ones). Its weighted average F1-Score is 59%.

Finally, we evaluated the ***Mistral-NeMo-12B-Instruct*** (AI, 2024b)[7] LLM having 12 billion parameters, jointly trained by NVIDIA and Mistral AI. The results achieved with this model are presented in Table 5. This model obtained 62% accuracy, which is slightly superior to the one of Mistral Instruct 7B and Granite 8B, but significantly lower than the one achieved by DeepSeek-R1-0528-Qwen3-8B. By inspecting the results, we observe that this model is strongly biased toward predicting the 'False' class. Consequently, it achieves high recall (80%) for that class, and low recall (45%) for the other class. Most errors produced by this model are false negatives (1162 instances), while false positives are significantly less frequent (376 instances). Overall, Mistral Instruct 12B achieves 60% weighted average F1-Score.

[6] https://huggingface.co/ibm-research/granite-3.2-8b-instruct-GGUFF

[7] https://lmstudio.ai/models/mistralai/mistral-nemo-instruct-2407

Table 5 Results of Mistral Instruct 12B model

		Mistral Instruct 12B		
		Pred True		Pred False
Exp True		938 (TP)		1,162 (FN)
Exp False		376 (FP)		1,524 (TN)
	Precision (%)	Recall (%)	F1-Score (%)	Support
Class 'True'	71	45	55	2,100
Class 'False'	57	80	66	1,900
Accuracy	Weighted Average Precision (%)	Weighted Average Recall (%)	Weighted Average F1 (%)	
62%	64	62	60	

Table 6 Summary of the results of the four models

	Accuracy (%)	Weighted Average Precision (%)	Weighted Average Recall (%)	Weighted Average F1 (%)	Total execution time (hh:mm:ss)
Mistral Instruct 7B	59	59	59	58	00:57:15
Qwen Deep-Seek 8B	**72**	**73**	**72**	**72**	03:52:45
Granite 3.2 8B	59	59	59	59	01:14:35
Mistral In-struct 12B	62	64	62	60	07:46:48

Best results are highlighted in bold

5.4 LLMs Comparative Analysis

Table 6 provides a comprehensive summary of all performance metrics obtained from the experimental evaluation, expressed as percentages to facilitate direct comparison across all tested models. The table also reports the average execution time of each model in seconds.

According to our experimental evaluation, the ***QDeepSeek-R1-0528-Qwen3-8B*** model emerged as the top performer across all metrics, showing a strong ability to accurately identify both satisfied and unmet policies. The ***Mistral Instruct 12B*** model ranked second, offering moderate effectiveness despite its larger parameter count. By contrast, ***Mistral-7B-Instruct-v0.3*** and ***Granite 3.2 8B*** delivered comparable yet overall mediocre results.

The modest performance of some smaller models reflects the intrinsic difficulty of enforcing strict logical constraints and highlights the discriminative power of the evaluation framework.

It is worth noting that the superior performance of ***DeepSeek-R1-0528-Qwen3-8B*** comes at a high computational cost: its evaluation required nearly four hours on the whole 4,000 instances; i.e., substantially longer than Mistral 7B and Granite 3.2 8B, which, however, produced the weakest outcomes. The Mistral Instruct 12B model required the longest processing time overall, despite achieving only intermediate performance. Its intermediate performance can be partly explained by its design: it is optimized to handle very large input contexts and to produce more discursive and free-form responses. Consequently, while the model excels at conversational tasks, it demonstrates lower efficacy in adhering to the rigid rule-based constraints required by this validation framework.

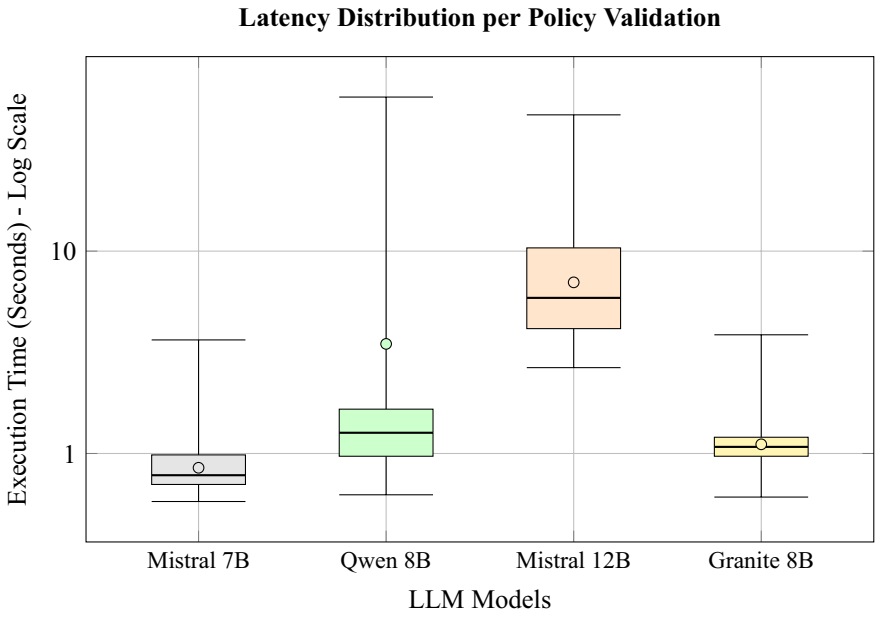


Fig. 4 Latency distribution per model ($N = 4,000$). The black line inside each box indicates the median execution time, while the circle mark represents the mean value. The colored box represents the Interquartile Range (IQR), spanning from the 25th ($Q1$) to the 75th ($Q3$) percentile. Note that the Y-axis employs a logarithmic scale to visualize the wide variance between models

5.5 Execution Time Analysis

While accuracy is paramount, the operational efficiency of the microservice is a critical factor for production deployment. Figure 4 illustrates the distribution of execution times for the 4,000 validation tasks across the four models. Note that a logarithmic scale is used to visualize the disparate magnitude of the results.

Mistral Instruct 7B demonstrated the best performance in per-task latency, achieving a remarkably low median of **0.78 seconds** and a mean of **0.85 seconds**. This sub-second latency makes it the most efficient model for high-frequency validation tasks.

Granite 3.2 8B followed closely with exceptional performance, achieving a median of approximately **1.08 seconds**. Its stability is highlighted by the close proximity of its mean (**1.11 seconds**) to the median, indicating a very narrow distribution with minimal computational overhead.

In contrast, **Mistral Instruct 12B** showed the highest latency among the tested models. While the median sits at **5.88 seconds**, the mean latency rises to **7.01 seconds**, reflecting the computational weight of the larger parameter count. This consistently higher processing time per task accumulates significantly over large batches.

Interestingly, while **DeepSeek-R1-0528-Qwen3-8B** maintained a competitive median latency (≈ 1.27 seconds), it exhibited extreme instability. The discrepancy between its low median and the significantly higher mean of **3.48 seconds**, along with an upper whisker reaching **57.72 seconds**, confirms a heavy-tailed distribution. These sporadic high-latency outliers drastically penalize the average throughput.

Cumulative Throughput Analysis: The impact of latency stability on the total processing time ($N = 4,000$) is evident in the final throughput results:

- **Mistral Instruct 7B** emerged as the most efficient overall, completing the entire benchmark in just **57 minutes and 15 seconds**.
- **Granite 3.2 8B** followed closely, with a total duration of **1 hour, 14 minutes, and 35 seconds**.
- **DeepSeek-R1-0528-Qwen3-8B** highlights the cost of instability: despite a low median similar to Granite, the presence of severe outliers pushed the average processing time up, resulting in a total duration of **3 hours, 52 minutes, and 45 seconds** (nearly $3\times$ longer than Granite).
- **Mistral Instruct 12B** confirmed its higher computational cost. Driven by its 7-second average latency, it recorded the longest total execution time of **7 hours, 46 minutes, and 48 seconds**.

6 Discussion and Error Analysis

In this study, we evaluated the capabilities of the four LLMs in validating YAML descriptors against natural language policies. This section analyzes the quantitative performance of the best-performing model (DeepSeek-R1-0528-Qwen3-8B) and provides an error analysis.

6.1 Quantitative Analysis: The Pessimistic Bias

The benchmark results identified **DeepSeek-R1-0528-Qwen3-8B** as the most accurate model, achieving an overall accuracy of 72% (F1-Score: 72%).

A deeper inspection reveals a pronounced asymmetry between the two classes. In particular, the model achieves higher recall on the ‘False’ class than on the ‘True’ class:

- Recall (Class ‘False’): 79%
- Recall (Class ‘True’): 66%

This pattern suggests a **pessimistic (conservative) bias**: the model tends to favor predicting non-compliance. The high recall for the ‘False’ class indicates that the model is effective at detecting actual violations (True Negatives). However, this behavior also increases the number of *False Negatives* (716 in Table 3), i.e., cases where compliant descriptors are incorrectly flagged as non-compliant. In practice, the model behaves as an "over-zealous" validator: when faced with ambiguity or complex structural relationships, it defaults to rejecting a descriptor rather than risking a false approval.

6.2 Analysis of Strengths: Procedural Instruction Following

Our findings indicate that policy phrasing has a substantial impact on the reliability of LLM-based validation. Policies expressed as explicit procedural checklists, clearly specifying steps, scopes, and evaluation criteria, are validated more consistently than abstract declarative rules. Based on our observations, we derive the following practical guidelines for policy authoring:

- decomposing complex policies into atomic requirements;
- stating the scope explicitly (e.g., product-level, component-level, or schema-field-level);
- avoiding biconditional statements by expressing them as two separate implications when possible.

Adopting these guidelines reduces ambiguity and improves determinism in LLM-driven policy verification workflows.

Practical Example (Success Case):

- Descriptor: `employees_2.yaml`, characterized by a complex nesting of output ports and schemas.
- Policy: `policy_2_1.txt`, a verbose, multi-step instruction requiring the extraction of all schema fields and the verification that the `description` key exists and is non-empty.
- Expected Result: `True`.
- Actual Result: `True`.*Analysis:* In this case, the policy effectively functioned as a step-by-step program. The prompt's *Strict Evaluation Pipeline* aligned well with the procedural structure of the rule, enabling the model to systematically traverse the YAML hierarchy, inspect nested lists, and validate the presence and content of required keys. This confirms that LLMs are particularly effective as validators when governance rules are articulated as explicit procedural checks rather than abstract logical constraints.

6.3 Error Analysis

While the quantitative analysis highlighted the overall accuracy and the pessimistic bias of the model, this section provides a granular qualitative analysis of specific failure modes. By examining individual descriptor-policy pairs, we identify the root causes of the observed errors, focusing on the tension between the semantic nature of LLMs and the strict logical requirements of the validation task.

One of the main causes of the errors (specifically False Positives and False Negatives) can be traced to a misalignment between the 'Strict Evaluation Pipeline' enforced by the prompt and the stochastic, semantic nature of LLMs.

6.3.1 Weakness: Fuzzy Matching vs. Strict Boolean Logic

The most critical weakness observed is the model's tendency to prioritize semantic association ('fuzzy matching') over strict logical operators, particularly the `AND` constraint. When a policy requires the co-existence of multiple components, the model often returns a positive result if it finds strong semantic evidence for *one* component, ignoring the absence of the other.

A clear illustration of this failure is the evaluation of the descriptor `cashflow_4.yaml` against `policy_8_9.txt`. The policy explicitly mandated two simultaneous conditions: the pipeline must be orchestrated by *Airflow* AND it must invoke a specific *Data Contract Guardian*.

In the actual test case, the descriptor contained a valid 'Guardian' component but relied on DBT and Kafka technologies, with no mention of Airflow. Despite the clear absence

of the mandatory orchestrator, the model incorrectly flagged the descriptor as compliant (`True`). It appears that the semantic weight of finding the technical term 'Guardian', which strongly matched the policy's subject, overwhelmed the logical check for the second condition. The model effectively allowed a partial semantic match to override the strict boolean `AND` instruction provided in the prompt.

6.3.2 Logical Gaps I: The Trap of Bidirectionality

A distinct failure mode emerged when dealing with bidirectional dependencies, often phrased as 'if and only if' ($A \iff B$). The models frequently failed to verify the implication in both directions, validating only one side of the relationship.

This behavior was evident in the evaluation of `employees_2.yaml` against `policy_5_2.txt`. The rule required a symmetric consistency: "GDPR status must be set at the product level *if and only if* at least one column is tagged as PII".

The descriptor in question did contain PII-tagged columns (satisfying the first condition) but failed to declare the GDPR tag at the root level (violating the consequent). Logically, this inconsistency should have resulted in a violation. However, the model marked the descriptor as valid. This suggests the LLM verified that the presence of PII justified the need for GDPR but failed to perform the reverse check-confirming that the GDPR tag was actually present. The model effectively reduced a bi-conditional rule to a simple, one-way association, failing to catch the missing requirement.

6.3.3 Logical Gaps II: Misunderstanding Vacuous Truth

Finally, the prompt structure failed to explicitly guide the model in handling **vacuous truths** in conditional logic.

In formal logic, a conditional statement 'If A, then B' is automatically valid whenever the antecedent A is false. However, without explicit instruction, the LLMs interpreted the absence of a condition as a validation failure rather than a pass.

Take, for instance, the case of `ecommerce_4.yaml` evaluated against `policy_4_3.txt`, which states: 'Presence of PII requires labeling the data product with GDPR'. The logic dictates that if no PII is present (A is False), the rule is inapplicable and the descriptor is compliant by default.

In our scenario, the descriptor contained absolutely no PII fields. Instead of recognizing this as a vacuous truth, the model produced a False Negative, marking the valid descriptor as non-compliant. The model, unable to find the 'PII' evidence mentioned in the premise, treated the inapplicability of the rule as a breach of policy. This confirms the model's pessimistic bias: when the triggering conditions of a policy are missing, the model prefers to reject the descriptor rather than inferring compliance from the absence of evidence.

Taken together, these failure modes highlight that, despite strong instruction-following capabilities, current LLMs struggle to consistently enforce strict logical constraints, particularly in the presence of conjunctions, bidirectional dependencies, and conditionally inapplicable rules.

7 Conclusions and Future Works

This paper investigated the use of LLMs for automatically validating data product descriptors against natural-language organizational policies in decentralized data ecosystems. We formalized the policy-descriptor compliance task, proposed a modular LLM-based microservice architecture, and conducted a large-scale empirical evaluation using an augmentation-based benchmark of 4,000 policy–descriptor pairs expressed in YAML.

Our experimental results demonstrate that LLMs can effectively support automated policy verification, achieving reasonable accuracy even in the presence of heterogeneous descriptor structures and paraphrased policy formulations, with overall accuracy ranging from 59% to 72% throughout the evaluated models. Among them, Qwen DeepSeek 8B achieved the best overall performance, obtaining 72% accuracy and 72% weighted F1, with per-class F1 of 71% for True and 73% for False. Considering the evaluated models, the average runtime per policy evaluation spans from 0.86 s using Mistral Instruct 7B to 7.00 s using Mistral Instruct 12B. The best-performing model, Qwen DeepSeek 8B, takes 3.49 s on average per check. We believe that these execution times are feasible for most applications in the considered domain. Importantly, the results also reveal consistent behavioral patterns across models, including a conservative bias toward predicting non-compliance: indeed, for the best-performing model, recall on the False class is 79% versus 66% on the True class.

Through qualitative error analysis, we identified several recurrent failure modes. These include difficulties in enforcing strict logical conjunctions, incorrect handling of bidirectional (if-and-only-if) constraints, and systematic misunderstandings of conditionally inapplicable rules (vacuous truths). Such errors highlight a fundamental tension between the probabilistic, semantic reasoning of LLMs and the deterministic logic required by governance and compliance tasks.

A key insight of this study is that policy formulation plays a critical role in validation reliability. Policies expressed as explicit procedural instructions, clearly defining scope, steps, and evaluation criteria, are handled significantly more consistently than abstract declarative rules. This finding suggests that LLM-based governance systems benefit not only from improved models, but also from careful policy authoring practices.

The ablation study further confirms the importance of structured prompt design. Removing intermediate reasoning stages that guide rule canonicalization, scope resolution, and evidence localization consistently degraded classification performance. This result indicates that LLM effectiveness in compliance-checking tasks does not depend solely on model capability, but also on the procedural structure imposed by the prompt.

We remark that our pipeline does not provide formal semantic guarantees, because the compliance decision depends on the LLM's interpretation of the policy. Indeed, in this work, we assume that natural-language policies are well-specified, non-ambiguous, and can be easily mapped by an LLM to explicit logical constraints. Techniques to provide correctness guarantees and handling uncertainty, including support for ambiguous or underspecified policies, will be investigated in future work, for instance via hybrid LLM and symbolic verification approaches.

Future work will explore several research directions. First, we plan to investigate hybrid approaches that combine LLM reasoning with symbolic or rule-based verification to enforce strict logical constraints. Second, we aim to extend the dataset with

independently annotated policies and descriptors to better assess generalization beyond paraphrase robustness. Finally, we will study interactive and explainable validation workflows, where LLMs assist human stakeholders in refining both policies and data product specifications.

In conclusion, while LLMs are not yet a drop-in replacement for formal policy engines, our results indicate that they can serve as a valuable component in scalable, semi-structured data governance pipelines when used with appropriate safeguards and design principles.

Acknowledgements We acknowledge the financial support of the project "Data Mesh Platform Builder with AI (DAMPAI)", funded under the "Fondo per la crescita sostenibile" by the "Ministero delle Imprese e del Made in Italy".

Author Contributions Conceptualization: Diego Reforgiato Recupero, Daniele Riboni, Francesco Simbola. Methodology: Diego Reforgiato Recupero, Daniele Riboni, Francesco Simbola. Dataset construction and curation: Martina Salis, Francesco Simbola. Software and system development: Francesco Simbola. Experimental evaluation: Francesco Simbola. Analysis and interpretation of results: Daniele Riboni, Martina Salis, Diego Reforgiato Recupero Supervision: Diego Reforgiato Recupero, Daniele Riboni. Writing - original draft: Francesco Simbola, Martina Salis. Writing - review and editing: Diego Reforgiato Recupero, Daniele Riboni.

Funding Open access funding provided by Università degli Studi di Cagliari within the CRUI-CARE Agreement. This research was conducted within the framework of the DAMPAI project.

Data Availability The dataset used for the paper is publicly available at: https://github.com/francescosimbola/Policies-and-Descriptors/tree/main.

Code Availability The software prototype used in this study was developed within the DAMPAI project in collaboration with an industrial partner. Due to intellectual property restrictions associated with the project, the full source code cannot currently be released in a public repository. However, the implementation can be made available for research purposes upon reasonable request to the corresponding author.

Declarations

Conflict of interest The authors declare that they have no competing interests that could have influenced the work reported in this paper.

Ethical Approval Not applicable.

Consent to Participate Not applicable.

Consent for Publication All participants provided explicit consent for the anonymous use and publication of their data for scientific research purposes.

Open Access This article is licensed under a Creative Commons Attribution 4.0 International License, which permits use, sharing, adaptation, distribution and reproduction in any medium or format, as long as you give appropriate credit to the original author(s) and the source, provide a link to the Creative Commons licence, and indicate if changes were made. The images or other third party material in this article are included in the article's Creative Commons licence, unless indicated otherwise in a credit line to the material. If material is not included in the article's Creative Commons licence and your intended use is not permitted by statutory regulation or exceeds the permitted use, you will need to obtain permission directly from the copyright holder. To view a copy of this licence, visit http://creativecommons.org/licenses/by/4.0/.

References

Abiteboul, S. (1997). Querying semi-structured data. In *Proceedings of the 6th international conference on database theory*. ICDT '97, pp. 1–18. Springer.

Acev, D., Biyani, S., Rieder, F., Aldenhoff, T. T., Blazevic, M., Riehle, D. M., & Wimmer, M. A. (2025). Systematic analysis of data governance frameworks and their relevance to data trusts. *Management Review Quarterly*. https://doi.org/10.1007/s11301-025-00545-1

AI, M. (2024a). Mistral-7B-Instruct-v0.3-GGUF. https://huggingface.co/lmstudio-community/Mistral-7B-Instruct-v0.3-GGUF

AI, M. (2024b). Mistral-nemo-instruct-2407. https://lmstudio.ai/models/mistralai/mistral-nemo-instruct-2407

Al-Turki, D., Hettiarachchi, H., Medhat Gaber, M., Abdelsamea, M. M., Basurra, S., Iranmanesh, S., Saadany, H., & Vakaj, E. (2024). Human-in-the-loop learning with llms for efficient rase tagging in building compliance regulations. *IEEE Access, 12*, 185291–185306. https://doi.org/10.1109/ACCESS.2024.3512434

Beach, T., Yeung, J., Nisbet, N., & Rezgui, Y. (2024). Digital approaches to construction compliance checking: Validating the suitability of an ecosystem approach to compliance checking. *Advanced Engineering Informatics, 59*, Article 102288. https://doi.org/10.1016/j.aei.2023.102288

Bena, Y. A., Ibrahim, R., Mahmood, J., Al-Dhaqm, A., Alshammari, A., Nasser, M., Nura Yusuf, M., & Ayemowa, M. O. (2025). Big data governance challenges arising from data generated by intelligent systems technologies: A systematic literature review. *IEEE Access, 13*, 12859–12888. https://doi.org/10.1109/ACCESS.2025.3528941

Bernardo, B. M. V., Mamede, H. S., Barroso, J. M. P., & dos Santos, V. M. P. D. (2024). Data governance & quality management–innovation and breakthroughs across different fields. *Journal of Innovation & Knowledge, 9*(4), Article 100598. https://doi.org/10.1016/j.jik.2024.100598

Bhathena, H., Joshi, A., & Singh, P. (2023). An efficient method for natural language querying on structured data. In S. Sitaram, B. Beigman Klebanov, & J. D. Williams (Eds.) *Proceedings of the 61st annual meeting of the association for computational linguistics* (Volume 5: Industry Track), pp. 322–331. Association for Computational Linguistics. https://doi.org/10.18653/v1/2023.acl-industry.31 . https://aclanthology.org/2023.acl-industry.31/

Buthelezi, M. P., Van Der Poll, J. A., & Ochola, E. O. (2016). Ambiguity as a barrier to information security policy compliance: A content analysis. In *2016 International conference on computational science and computational intelligence (CSCI)*, pp. 1360–1367. https://doi.org/10.1109/CSCI.2016.0254

Chen, N., Lin, X., Jiang, H., & An, Y. (2024). Automated building information modeling compliance check through a large language model combined with deep learning and ontology. *Buildings, 14*(7), Article 1983. https://doi.org/10.3390/buildings14071983

Dehghani, Z., & Fowler, M. (2022). *Data mesh: Delivering data-driven value at scale*. O'Reilly Media.

Górski, T. (2026). Software architecture description in original software publications. *Software Impacts, 27*, Article 100802. https://doi.org/10.1016/j.simpa.2025.100802

Gupta, V., Kandoi, P., Vora, M., Zhang, S., He, Y., Reinanda, R., & Srikumar, V. (2023). TempTabQA: Temporal question answering for semi-structured tables. In H. Bouamor, J. Pino, & K. Bali (Eds.) *Proceedings of the 2023 conference on empirical methods in natural language processing*, pp. 2431–2453. Association for Computational Linguistics. https://doi.org/10.18653/v1/2023.emnlp-main.149 . https://aclanthology.org/2023.emnlp-main.149/

Hosseini, M. B., Heaps, J., Slavin, R., Niu, J., Breaux, T. (2021). Ambiguity and generality in natural language privacy policies. In *2021 IEEE 29th international requirements engineering conference (RE)*, pp. 70–81. https://doi.org/10.1109/RE51729.2021.00014

Ibba, A., Reforgiato Recupero, D., & Alonso, R. (2025). Enabling natural language access to bim models with ai and knowledge graphs.

Jaimovitch-López, G., Ferri, C., Hernández-Orallo, J., Martínez-Plumed, F., & Ramírez-Quintana, M. J. (2023). Can language models automate data wrangling? *Machine Learning, 112*(6), 2053–2082. https://doi.org/10.1007/s10994-022-06259-9

Janssen, M., Brous, P., Estevez, E., Barbosa, L. S., & Janowski, T. (2020). Data governance: Organizing data for trustworthy artificial intelligence. *Government Information Quarterly, 37*(3), Article 101493. https://doi.org/10.1016/j.giq.2020.101493

Jiang, H., & Bouabdallah, A. (2017). Towards a json-based fast policy evaluation framework. In H. Panetto, C. Debruyne, W. Gaaloul, M. Papazoglou, A. Paschke, C. A. Ardagna, & R. Meersman (Eds.), *On the move to meaningful internet systems. OTM 2017 Conferences* (pp. 22–30). Springer.

Khan, I. (2024). Creating code with codex, pp. 287–301.

Khatri, V., & Brown, C. V. (2010). Designing data governance. *Communications of the ACM, 53*(1), 148–152. https://doi.org/10.1145/1629175.1629210

Kiyavitskaya, N., Zeni, N., Breaux, T. D., Antón, A. I., Cordy, J. R., Mich, L., & Mylopoulos, J. (2007). Extracting rights and obligations from regulations: toward a tool-supported process. In *Proceedings of the 22nd IEEE/ACM international conference on automated software engineering*. ASE '07, pp. 429–432. Association for Computing Machinery, New York, NY, USA. https://doi.org/10.1145/1321631.1321701 .

Liu, X., Li, H., & Zhu, X. (2023). A GPT-based Method of Automated Compliance Checking through Prompt Engineering. Technical report, Cardiff University ORCA repository. Accessed: 2025-12-07. https://orca.cardiff.ac.uk/id/eprint/159694/1/A%20GPT-based%20method%20of%20Automatic%20Compliance%20Checking%20through%20prompt%20engineering_20230417001851.pdf

OpenAI, Achiam, J., Adler, S., Agarwal, S., Ahmad, L., Akkaya, I., Aleman, F. L., Almeida, D., Altenschmidt, J., Altman, S., Anadkat, S., Avila, R., Babuschkin, I., Balaji, S., Balcom, V., Baltescu, P., Bao, H., & Zoph, B. (2024). GPT-4 technical report. https://arxiv.org/abs/2303.08774

Ren, R., Ma, J., & Zheng, Z. (2025). Large language model for interpreting research policy using adaptive two-stage retrieval augmented fine-tuning method. *Expert Systems with Applications, 278*, Article 127330. https://doi.org/10.1016/j.eswa.2025.127330

Research, I. (2024). granite-3.2-8b-instruct-GGUF. https://huggingface.co/ibm-research/granite-3.2-8b-instruct-GGUF

Shen, X., Barlacchi, G., Del Tredici, M., Cheng, W., & Gispert, A. (2022). semiPQA: A study on product question answering over semi-structured data. In S. Malmasi, O. Rokhlenko, N. Ueffing, I. Guy, E. Agichtein, & S. Kallumadi (Eds.) *Proceedings of the Fifth Workshop on e-Commerce and NLP (ECNLP 5)*, pp. 111–120. Association for Computational Linguistics, Dublin, Ireland. https://doi.org/10.18653/v1/2022.ecnlp-1.14 . https://aclanthology.org/2022.ecnlp-1.14/

Sondhi, P., & Zhai, C. (2014). Mining semi-structured online knowledge bases to answer natural language questions on community qa websites. In *Proceedings of the 23rd ACM international conference on conference on information and knowledge management*

Sun, J., Luo, Z., & Li, Y. (2025). A compliance checking framework based on retrieval augmented generation. In O. Rambow, L. Wanner, M. Apidianaki, H. Al-Khalifa, B. D. Eugenio, & S. Schockaert (Eds.) *Proceedings of the 31st international conference on computational linguistics*, pp. 2603–2615. Association for Computational Linguistics, Abu Dhabi, UAE. https://aclanthology.org/2025.coling-main.178/

Team, D. (2024). deepseek-r1-0528-qwen3-8b. https://lmstudio.ai/models/deepseek/deepseek-r1-0528-qwen3-8b

Waltinger, U., Tecuci, D., Olteanu, M., Mocanu, V., & Sullivan, S. (2013). Usi answers: Natural language question answering over (semi-) structured industry data. In *Proceedings of the AAAI conference on artificial intelligence* 27, pp. 1471–1478. https://doi.org/10.1609/aaai.v27i2.18985

Wang, Y., Cai, C., Xiao, Z., & Lam, P. E. (2025). LLM access shield: Domain-specific LLM framework for privacy policy compliance. https://arxiv.org/abs/2505.17145

Weber, I. (2024). Large language models are pattern matchers: Editing semi-structured and structured documents with chatgpt. https://doi.org/10.18420/AKWI2024-001

Ying, H., & Sacks, R. (2024). From automatic to autonomous: A large language model-driven approach for generic building compliance checking. EasyChair Preprint 15364, ??? (EasyChair, 2024)

Publisher's Note Springer Nature remains neutral with regard to jurisdictional claims in published maps and institutional affiliations.

Authors and Affiliations

Francesco Simbola[1] · Diego Reforgiato Recupero[1] · Daniele Riboni[1] · Martina Salis[1]

✉ Diego Reforgiato Recupero
diego.reforgiato@unica.it

Francesco Simbola
f.simbola1@studenti.unica.it

Daniele Riboni
riboni@unica.it

Martina Salis
marti.salis20@gmail.com

[1] Department of Mathematics and Computer Science, University of Cagliari, Via Ospedale 72, 09124 Cagliari, Italy