Observable Consistency Checking across Requirements and Models

Tianhai Liu[1][a], Shmuel Tyszberowicz[2][b] and Bernhard Beckert[1][c]
[1]*Karlsruhe Institute of Technology (KIT), Karlsruhe, Germany*
[2]*Afeka Academic College of Engineering, Tel Aviv, Israel*

Keywords: Model-Driven Development, Consistency Checking, Ontology, Observables, LLM, RAG.

Abstract: Model-Driven Development (MDD) promises productivity gains in software development. However, its adoption remains limited due to persistent challenges in maintaining consistency, both between requirements and the models derived from them, and across the models themselves. Heterogeneous requirements and models from multiple disciplines, such as software, mechanical, or electrical engineering, often overlap in describing a system. They use different terminology and value ranges, which potentially leads to misalignments and contradictions. Existing syntactic methods capture only structural differences, while constraint-based approaches struggle to integrate heterogeneous artefacts. We address this challenge through *observables*: domain properties constrained by requirements and models, which serve as a unifying abstraction for consistency. We present an ontology-driven framework that leverages retrieval-augmented generation and large language models (LLMs) to automatically extract observables and constraints from heterogeneous artefacts. The extracted observables are harmonised via ontology construction, and the constraints are compiled into solver-ready formulas. This enables observable-centric semantic consistency checks across requirements and models, ensuring a consistent and accurate representation of data. It helps engineers identify inconsistencies earlier and reduces manual effort in consistency analysis. Evaluation on synthetic datasets and an industrial automotive case study demonstrates the feasibility and scalability from small sets to thousands of requirements.

1 INTRODUCTION

Model-Driven Development (MDD) elevates abstract models to primary artefacts, enabling engineers to describe system structure and behaviour at a higher level and transform these models into executable components. It improves communication and reduces errors through automation. Despite its benefits, MDD faces persistent challenges. A key problem is *inconsistency across models*: different system views (e.g., structural, behavioural, deployment) overlap in their descriptions of concerns, so changes in one view can easily introduce conflicts in others unless they are synchronised. In complex domains such as automotive or aerospace engineering, where software, electrical, and mechanical artefacts must interoperate, heterogeneity of languages, tools, and assumptions aggravates the problem. The same artefact may be described by different terms across disciplines (e.g., *rotor*, *disc*, *brake component*). Conversely, the same term may denote different artefacts depending on context (e.g., *controller* could mean an *electronic control unit*, a *control algorithm*, or a *Simulink block*). Syntactic checks cannot capture such semantic misalignments, limiting the feasibility and scalability of MDD.

Consistency should be defined not merely at the syntactic level, but instead grounded in semantics. The *Virtual Single Underlying Model* (V-SUM) methodology formalises consistency as a relation across heterogeneous models and supports reasoning via transformation rules, semantic consistency, or co-satisfiability (Klare et al., 2021). Crucially, each consistency relation is associated with a canonical natural semantics that captures precisely the information required to decide consistency. In (Färber et al., 2025), consistency is defined in terms of *observables* which are measurable domain properties constrained by models. Only model implementations whose observable behaviour satisfies these constraints are considered admissible.

What remains unresolved is how to systematically *identify relevant observables and their admissi-*

[a] https://orcid.org/0000-0001-5881-1920
[b] https://orcid.org/0000-0003-4937-8138
[c] https://orcid.org/0000-0002-9672-3291

Liu, T., Tyszberowicz, S. and Beckert, B.
Observable Consistency Checking across Requirements and Models.
DOI: 10.5220/0014719400004015

In *Proceedings of the 21st International Conference on Evaluation of Novel Approaches to Software Engineering (ENASE 2026) - Volume 1*, pages 50-63
ISBN: 978-989-758-828-0; ISSN: 2184-4895

ble values in practice, and how to integrate them into (meta)models to ensure consistency across heterogeneous views. Existing approaches only partially address this challenge. Logic-based approaches are expressive but struggle to scale to large, heterogeneous artefacts. While LLM-based pipelines are promising for extracting information from technical text, they face inherent limitations: *restricted context windows, a tendency to hallucinate, and the inability to access proprietary industry IP that lies outside their pretraining corpora*. None of these approaches alone provides a scalable way to ground consistency in semantics while addressing the scale, heterogeneity, and proprietary nature of industrial documentation.

The challenges of large-scale documentation, multidisciplinary integration, and consistency management are particularly acute in safety-critical, highly regulated domains such as automotive, aerospace, rail, and healthcare. In these contexts, requirements and standards often span hundreds or even thousands of pages, covering topics ranging from communication protocols and software behaviour to mechanical tolerances. Manually maintaining consistency across documentation of this scale is unrealistic and highly error-prone.

To address these limitations, we employ *Retrieval-Augmented Generation (RAG)* (Gao et al., 2023) to enhance LLM inference with externally retrieved knowledge. Instead of relying solely on an LLM's internal representations, relevant fragments from authoritative sources are retrieved and supplied as additional context during inference. RAG reduces the LLM's reliance on vague internal guesses and increases its reliance on trusted domain documents. In our setting, RAG primarily serves as a scalability mechanism for handling large volumes of requirements and standards, while improving traceability and contextual grounding. Based on the knowledge graph constructed from these artefacts, we subsequently perform a conservative construction of synonym groups over its entities and derive solver-ready logical formulas, enabling automated SMT-based verification (Barrett et al., 2010) of consistency across requirements and models.

Contributions. Maintaining consistency among requirements and across models remains a persistent challenge in multidisciplinary model-driven development. This paper introduces an *observable-centric* approach that enables the derivation of solver-ready SMT formulas for automated consistency analysis. Observables are treated as first-class entities to support both syntactic and semantic alignment across heterogeneous artefacts. The main contributions of this work are:

- **Observable-Centric Consistency Formulation.** We propose an observable-based representation that enables the systematic derivation of SMT constraints for automated consistency checking across requirements and models.
- **Ontology-Driven Extraction Framework.** We develop a pipeline that combines ontology-guided prompting with RAG to extract observables and constraints from large-scale technical documents.
- **Cross-View Consistency Analysis.** The approach enables consistency checking along both vertical (requirements–models) and horizontal (inter-model) development dimensions.
- **Tool Support.** We implement the approach in RAG4C, which automates the observable extraction, ontology consolidation, constraint generation, and SMT-based consistency checking. The tool can also derive structural UML models from requirements to support system understanding.
- **Empirical Evaluation.** Experiments on industrial datasets show that RAG4C detects inconsistencies with high accuracy (0.97) while reducing manual effort by more than an order of magnitude.

Our *observable consistency checking* approach provides a complementary form of requirements analysis that focuses on quantitative and categorical domain properties shared across disciplines, complementing conventional behavioural techniques and temporal consistency techniques in RE.

2 RELATED WORK

We organise the survey of related work according to the challenges our approach addresses.

LLMs in Requirements Engineering (RE). LLMs are increasingly used in RE. The potential of LLMs to enhance precision and reduce ambiguity in requirements specifications has been demonstrated (Hemmat et al., 2025), while limitations due to scarce domain-specific data and inconsistent annotations have also been reported (Norheim et al., 2024). To address these issues, integrating LLMs with formal RE techniques has been proposed to improve correctness and trust (Ferrari and Spoletini, 2025). Empirical results from the OntoURL benchmark (Zhang et al., 2025) further indicate that LLMs alone struggle with complex ontology reasoning, motivating hybrid solutions that integrate statistical language models with symbolic reasoning. Taken together, these studies illustrate both the promise and the risks (hallucination, scalability) of LLM-based RE, thereby motivating our

semantics-grounded approach, which ensures consistency among requirements.

Consistency in MDD. Research on consistency in model-based engineering spans several viewpoints. A classification of consistency notions along syntactic–semantic and binary–multiparty dimensions has been provided (Feichtinger et al., 2024), and a seven-dimensional schema for unifying related terminology has also been introduced (Kühn et al., 2023). Operationalisation has been realised through rule-based frameworks, such as Vitruvius (Klare et al., 2021), which coordinate delta-based rules across artefacts using Object Constraint Language (OCL), graph transformations, or triple graph grammars. More recent work lifts syntactic relations into semantic spaces to obtain a canonical natural semantics for deciding consistency (Pascual et al., 2024). However, these approaches remain mainly syntactic and rely on manual engineering. In contrast, RAG4C automates this process by using LLMs to extract observables and constraints, replacing brittle rules with ontology-grounded abstractions that scale across heterogeneous artefacts and domains.

Consistency in RE. Consistency issues in RE arise in different forms. Temporal conflicts stem from contradictory event orders (e.g., "the airbag shall deploy before impact detection" vs. "deployment occurs only after impact confirmation"). Such inconsistencies are typically addressed using model-checking and temporal-logic techniques (Clarke et al., 1999; Pnueli, 1977; Ma et al., 2025). Other inconsistencies arise from incompatible action conditions (e.g., "in case of an emergency, the safety shutdown mechanism shall activate automatically" vs "the safety shutdown mechanism shall activate only after explicit user confirmation"). Since both constraints share the same trigger but impose mutually exclusive activation policies, they cannot be satisfied jointly. Such conflicts are commonly analysed using behavioural or state-based models that represent transitions and control flow (Uchitel et al., 2003; Fuxman et al., 2004). In contrast, observable consistency concerns quantitative and categorical domain properties shared across heterogeneous artefacts and disciplines. These conflicts may remain latent until integration and often arise from inconsistent value ranges, incompatible measurement assumptions, or misaligned abstractions across disciplinary views. In our framework, behavioural conditions can be represented as predicates over shared categorial observables (e.g., emergency state, confirmation status, activation flag). In this state-based form, incompatibility reduces to logical unsatisfiability and can be detected through solver-based constraint reasoning. While our approach does not perform trace-based temporal verification, it complements conventional behavioural and temporal techniques by grounding consistency in normalised observables and addressing cross-disciplinary conflicts.

GraphRAG. GraphRAG (Edge et al., 2024) extends retrieval-augmented generation by constructing knowledge graphs from corpora and has been shown to outperform standard RAG in query-focused summarisation. Building on this idea, we extend GraphRAG by grounding extracted entities and relations in ontology-based observables and translating them into SMT-ready formulas, strengthening systematic consistency analysis. In addition, GraphRAG focuses purely on summarisation, while we focus on semantic consistency checking. Other domain-specific approaches also employ LLMs to generate structured artefacts. LLM-ACNC has been proposed as an LLM-based reasoning for constructing aerospace requirement graphs (Liu et al., 2025), but its scope is limited to direct information extraction. In contrast, we employ LLMs only indirectly through GraphRAG, which helps mitigate LLM hallucinations and biases in domain-specific documents.

Ontology-Driven Approaches. Ontologies have long been employed in RE to align heterogeneous artefacts and support inconsistency detection across models and specifications (Verlaine et al., 2012). More recent work extends this direction by verifying ontology-based process models with explicit parameter interdependencies (Jeleniewski et al., 2025). To reduce the manual effort involved in ontology modelling, ontology-learning approaches aim to discover domain concepts from text automatically. For example, Xu et al. (Xu et al., 2018) cluster noun phrases to construct modular ontologies. However, such approaches still presuppose a *stable and sufficiently complete vocabulary*. Their effectiveness degrades in large, evolving, and authorisation-restricted requirement corpora where terminology changes frequently and expert curation is costly. In contrast to these *knowledge-driven* approaches, our approach is *discovery-driven*, enabling scalable, bottom-up extraction of observables and synonym groups directly from natural-language (NL) requirements.

LLM-Based UML Generation. Recent work has explored using LLMs to generate UML models from NL requirements, including the approaches (NL-OOPS, LIDA, CM-Builder, and ABCD), as surveyed

for class-diagram extraction (Abdelnabi et al., 2021) and LLM-based approaches for generating UML behavioural models (Ferrari et al., 2024). These approaches focus primarily on software design. In contrast, we leverage observable extraction and constraint reasoning to support cross-disciplinary consistency analysis, while deriving UML structures as an auxiliary artefact.

3 APPROACH

Two artefacts (e.g., two requirement documents) are consistent if at least one system design satisfies both. To illustrate our approach to identifying inconsistencies, Figure 1 presents a two-layer diagram. The lower layer, labelled *Requirement Specification*, contains NL requirements. The upper layer, labelled *System Models*, shows the corresponding models (e.g., class diagrams), each aligned to one requirement document. The diagram highlights three key consistency relations, all defined with respect to shared observables: (i) each model must be consistent with its associated requirements; (ii) requirements must be mutually consistent; and (iii) models must be mutually consistent.

We adopt the notion of *observables*, system artefacts that can be identified across requirements and models and that are associated with values and constraints. Observables serve as a unifying abstraction for verifying both terminological alignment and quantitative consistency across domains. For example, the observable *car weight* may be specified in one requirement as lying within [1000,1200] kg, while another requirement constrains it to [1500,2500] kg. Our approach detects this conflict and reports the inconsistency. On the other hand, if the latter is [1100,2000] kg, the overlap indicates consistency, and our approach can construct a corresponding assignment of values for the observables. Likewise, for system-level observables that represent aggregates (e.g., total vehicle weight as the sum of component weights), our method checks that values specified in different requirements are compatible and that their composition satisfies the aggregate constraint. These illustrate the broader principle of our method: verifying consistency and, when it holds, producing a witness model that demonstrates joint satisfiability.

We propose an ontology-driven framework, *RAG4C*, that automates the extraction of observables and their associated constraints from requirements spanning heterogeneous domains, checks their consistency, and derives structural models that organise the entities and relationships in the requirements to facilitate their understanding. The workflow of RAG4C consists of four primary stages (Figure 2): (i) Knowledge Graph Construction, (ii) Ontology Construction, (iii) Consistency Checking, and (iv) Model Generation. The subsections below describe each stage in detail, using an automotive braking system as a running example.

3.1 Automotive Braking System Use Case

We illustrate our approach using a simplified vehicle braking system (Listing 1). The example abstracts from real systems for clarity, while preserving the essential cross-domain characteristics. Observables are of two kinds: *categorical* and *numerical*. Categorical observables have finite sets of values that denote the status of an observable, e.g., the validity of a torque command. Numerical observables denote measurable quantities or ranges, e.g., pedal effort, measured pedal force, and pedal force. These observables refer to the same physical quantity, albeit with different names. Despite using different terminology, all three aim to ensure that brake-pedal force remains within admissible bounds during operation. Early detection of inconsistencies across observables is essential for reliable and efficient MDD. Such terminological variation hinders automated reasoning about consistency and traceability across specifications and models. Mapping these terms to shared concepts restores semantic alignment across disciplines. This example highlights the need for ontology-driven unification of observables, where synonym resolution and a controlled vocabulary are key to maintaining consistency in automotive multidisciplinary requirements.

3.2 Repository Preparation

The document repository used for retrieval is curated for observable extraction. We select requirement documents that contain rich quantitative and categorical observables, including the industrial datasets used in the evaluation (Sec. 4) and synthesised requirement sets constructed for controlled experiments.

As a preprocessing step, source PDFs are converted to structured text using the MinerU OCR framework (MinerU, 2024). Graphical elements (e.g., diagrams and figures) are excluded during extraction to focus on textual requirements and ensure comparability with conventional natural language processing (NLP) baselines. Document noise is reduced by removing project-management artefacts such as page numbers, headers, footers, and administrative metadata (e.g., partner lists, milestones, and funding refer-

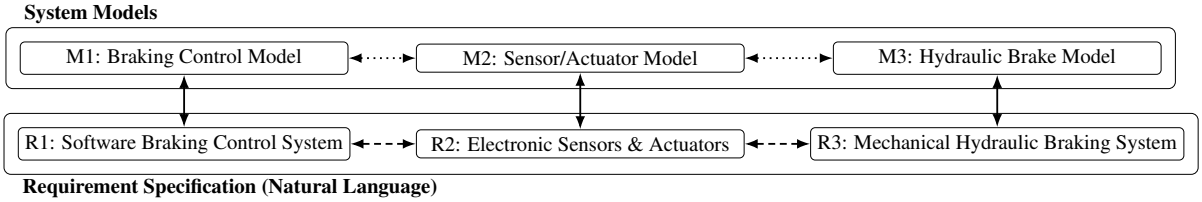

Figure 1: Two-layer view linking requirements (bottom) to model counterparts (top).

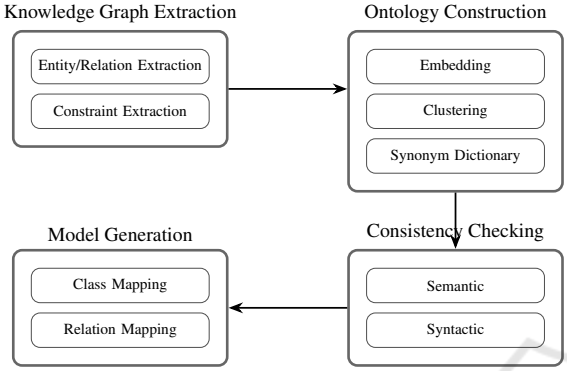

Figure 2: The workflow of RAG4C.

ences), retaining mainly system descriptions and requirement statements. Units and dimensions (e.g., Nm vs Nmm) are normalised during preprocessing using a lightweight Python script, allowing us to focus on observable extraction and synonym grouping while remaining compatible with extensible unit-handling mechanisms. Document identifiers are preserved throughout processing to maintain traceability between extracted observables and their source artefacts.

Listing 1: Illustrative braking system requirements.

```
The braking system consists of a mechanical pedal/booster
assembly, an electronic sensing unit, and a software
controller. The driver's pedal effort is converted
mechanically into brake actuation force. The electronic
unit measures the resulting pedal force and produces a
digitised signal. The software controller consumes the
sensor signal and computes a brake torque command, which
is then provided to the actuator.
=== Mechanical Requirements Specification ===
MRS-001: The pedal effort produced by the driver shall be
within the range [0, 300] N.
=== Electronic Requirements Specification ===
ERS-001: The measured pedal force reported by the sensor
shall lie within the range [400, 500] N.
=== Software Requirements Specification ===
SRS-001: The brake torque command shall be valid whenever
the reported pedal force is at least 50 N.
```

3.3 Knowledge Graph Construction

At this stage, RAG4C extracts observable candidates (i.e., entities mentioned in the requirements), together with their relationships and associated constraints. Our approach extends GraphRAG (Edge et al., 2024) to segment the preprocessed text corpus into fine-grained retrieval units using the default configuration (chunk size of 1200 tokens with a 100-token overlap). For each unit, the LLM is then prompted to extract entities and their relationships, which are subsequently assembled into a knowledge graph. Conventional NLP tools (e.g., spaCy (Honnibal et al., 2020)) offer reliable context-unaware text recognition, but ambiguity in NL hinders interpretation. LLMs leverage large training corpora to identify entities in requirements, yet they remain prone to hallucination in complex reasoning. Domain-specific prompts mitigate this risk and improve extraction precision (e.g., (Chen et al., 2023)), but standard prompts often fail to unify cross-disciplinary terms, leading to fragmented recognition and incomplete relations (e.g., (Ke and Ribeiro, 2025)). To address this, we design prompts to extract observables and constraints from requirements.

Prompt. Table 1 defines the entity and relation types that guide the LLM during knowledge graph construction, and provides a structure for organising entities of the specified system, which are used for model generation in Sec. 3.6. Entities formalise the building blocks of the requirements model: systems, observables, requirements, and enumerated types. In particular, observables of quantitative and categorical types are essential for constraint reasoning. Relations connect entities, including hierarchical decompositions (`CONTAINS`), where systems are composed of subsystems, and operational dependencies that capture data flow or resource exchange (`PRODUCES` and `CONSUMES`). By constraining relationships to a predefined set of controlled types, the schema prevents spurious links and ensures generated models are represented in a consistent form.

Figure 3 defines the grammar that guides the LLM to construct constraints from NL requirement

Table 1: Entity and relation types.

Type	Description
`System` (Entity)	Vehicle system, subsystem, or component.
`Observable` (Entity)	Measurable / estimated property of a system.
`Requirement` (Entity)	Requirement item with title and description.
`EnumType` (Entity)	Reusable enumeration for categorical observable values.
`CONTAINS` (Relation)	System includes another as part of its structure (e.g., Vehicle `CONTAINS` Battery).
`PRODUCES` (Relation)	A component exposes an observable. (e.g., Sensor `PRODUCES` Measurement).
`CONSUMES` (Relation)	A component depends on an observable (e.g., Controller `CONSUMES` Measurement).

specifications according to a predefined, normalised structure. This helps obtain a uniform, machine-interpretable representation of the extracted constraints, which in turn supports downstream automated consistency analysis.

We use chain-of-thought reasoning to improve extraction precision and interpretability. Instead of generating entities, relations, and constraints in a single step, LLM performs stepwise inference: first identifying entities, then reasoning about relations, and finally producing structured triples. We further provide a small few-shot prompt to demonstrate the expected output format. Listing 2 presents the prompt template used for knowledge-graph construction. Table 2 summarises the extracted knowledge graph for the automotive requirements in Listing 1, and Figure 4 illustrates the resulting class-diagram view.

3.4 Ontology Construction

GraphRAG can group the extracted entities into communities based on co-occurrence and structural relationships within the knowledge graph. This provides a topological organisation, but it does not reflect the community's semantic similarity. Consequently, terms referring to the same concept (e.g., `pedal effort` and `pedal force`) may appear in different disciplines and across widely separated parts of the documentation. Consequently, they are treated as distinct entities, reducing the precision of the consistency analysis.

$$
\begin{aligned}
\langle requirement \rangle &::= \langle bool\_formula \rangle \\
\langle bool\_formula \rangle &::= \langle clause \rangle \mid \texttt{NOT}\ \langle bool\_formula \rangle \\
&\quad \mid \langle bool\_formula \rangle\ \langle logic\_op \rangle\ \langle bool\_formula \rangle \\
\langle clause \rangle &::= \langle term \rangle\ \langle bool\_exp \rangle\ \langle term \rangle \\
\langle term \rangle &::= \langle observable \rangle \mid \langle number \rangle \mid \langle enum\_val \rangle \\
&\quad \mid \langle term \rangle\ \langle operator \rangle\ \langle term \rangle \\
\langle bool\_exp \rangle &::= (= \mid != \mid < \mid <= \mid > \mid >=) \\
\langle logic\_op \rangle &::= (\texttt{IMPLIES} \mid \texttt{IFF} \mid \texttt{AND} \mid \texttt{OR}) \\
\langle operator \rangle &::= (+ \mid - \mid * \mid /) \\
\langle observable \rangle &::= qualified\_name \\
\langle enum\_val \rangle &::= \text{UPPER\_SNAKE} \\
\langle number \rangle &::= integer \mid real
\end{aligned}
$$

Figure 3: Bnf grammar of normalised constraints.

Listing 2: Prompt for knowledge graph construction.

```
--Goal--
Extract a knowledge graph from the text using only allowed
entity/relationship types.
--Entities--
Types: System, Observable, Requirement, EnumType. Format:
("entity", <name>, <type>, <description>)
Rules: 1) <description> = normalised expr, 2) <name> =
requirement ID (e.g., MRS-001).
--Relationships--
Types: CONTAINS, PRODUCES, CONSUMES. Format:
("relationship", <src>, <tgt>, <desc>, <strength>)
--Steps--
1) Extract entities, 2) Extract relationships, 3) Output:
entities and relationships separated
[Definitions for entities and relationships omitted]
[Quality rules omitted]
--Example--
Input: ERS-001: The pedal force sensor readings must be
valid if the booster sensor reports a valid status.
Output: ("entity", ers001, Requirement,
(BOOSTER_SENSOR=VALID) IMPLIES (pedalForceSensor=VALID))
--Real Data--
Input: {input_text}
Output:
```

Table 2: Extracted entities and relationships from use case.

Type	Items
`System`	braking system, mechanical subsystem, electronic subsystem, software subsystem
`Observable`	pedal effort, pedal force, measured pedal force, brake torque command
`Requirement`	mrs-001, ers-001, srs-001
`CONTAINS`	(braking system, mechanical subsystem), (braking system, electronic subsystem), (braking system, software subsystem)
`PRODUCES`	(mechanical subsystem, pedal force), (electronic subsystem, measured pedal force), (software subsystem, brake torque command)
`CONSUMES`	(electronic subsystem, pedal force), (software subsystem, measured pedal force)

We consolidate extracted entities into a cross-disciplinary ontology. An ontology is a formal, structured representation of domain knowledge. It defines concepts (classes), individuals (instances), relationships (properties), and constraints, providing a logically consistent model that enables semantic interoperability (Zhang et al., 2025). This ontology provides a controlled vocabulary spanning domains and serves as the semantic backbone for automated reasoning.

Embedding. We construct a compact textual representation for each entity that preserves its original surface form while enriching it with contextual semantics. The representation consists of three components: (i) the raw entity name, (ii) a tokenised variant of the name obtained by splitting on common delimiters and case transitions, and (iii) the NL description associated with the entity. Appending the description further captures the entity's semantic intent beyond purely lexical similarity.

The resulting textual representations are embedded into dense vector spaces using a semantic embedding model compatible with the OpenAI embeddings API (OpenAI, 2025b). All embeddings are L2-normalised to unit length, enabling reliable use of cosine similarity (and equivalently, cosine distance) for downstream clustering.

Clustering—Synonym Grouping. We employ *Agglomerative Clustering*, a hierarchical algorithm that begins by treating each entity as an individual cluster and iteratively merges the most similar clusters until a full hierarchy (dendrogram) is built (Ieva et al., 2018). Unlike *K-Means*, which enforces a fixed number of clusters and may force unrelated entities together, it allows flexible merging thresholds, making it suited for synonym identification. The quality of clustering, particularly its precision and recall, is influenced by the choice of distance metric. We use the cosine distance, defined as: cosine distance = 1 - cosine similarity. For example, a cutoff such as "merge if cosine distance < 0.35" groups only semantically close entities. Adjusting the threshold balances precision (avoiding false merges) and recall (capturing true equivalences). From each cluster, one synonym is chosen to represent the ontology term. If a cluster contains only a single synonym, that synonym itself is used as the corresponding ontology term.

To reduce spurious alignments, we adopt a conservative clustering strategy and deliberately limit over-merging by restricting clustering to observables with compatible types and contexts in the knowledge graph. An optimal *distance threshold* (Sec. 4.2) is used to reduce over-merging risk. Average linkage limits bridging effects that might otherwise chain unrelated terms into the same cluster. It measures the distance between two clusters as the mean of all pairwise distances, reducing the chaining problem of single linkage while remaining less conservative than complete linkage, and thus offers a practical balance for semantic clustering.

Synonym Dictionary. We maintain a synonym dictionary that maps raw observables in the requirements to canonical ontology terms. Each observable is assigned a unique qualified name (e.g., `REQ001_observable`), and linked to its corresponding ontology concept. Formally, the dictionary provides a mapping

$$dict : Q \to S$$

from qualified raw observables Q to ontology terms S, ensuring that all reasoning is performed on a unified ontology. For backward traceability, we use the induced inverse mapping

$$dict^{-1}(s) = \{q \in Q \mid dict(q) = s\},$$

which allows recovery of all raw observables associated with the same ontology term. Thus, consistency violations detected during reasoning can be reported using the original requirement items and terms.

3.5 Consistency Checking

Semantic consistency prevents contradictions in admissible values among artefacts, vertically and horizontally; syntactic consistency avoids naming and structural mismatches across layers (Figure 1).

3.5.1 Semantic Consistency Checking

Categorical observables are encoded either as finite-domain integers with explicit value constraints or as uninterpreted SMT sorts with enumerated axioms. Numerical observables are represented as SMT `Real` sorts to support arithmetic reasoning.

Although these requirements originate from different stakeholders, they share a common structure: each encodes constraints over observables. Semantic checking, therefore, follows a uniform workflow: (i) extract and normalise observables using the ontology and synonym dictionary; (ii) translate all constraints into a unified constraint grammar; and (iii) check the satisfiability of the conjunction of all constraints. For clarity, we present the theory using requirements as the running example, although the same principles also extend to design models. For example, when provided with UML class diagrams and accompanying OCL constraints in textual form, RAG4C extracts observables, constructs the corresponding ontology concepts, and translates the constraints into SMT formulas for consistency checking. Consider a case where a UML class `BrakeController` declares attribute `pedalForce: Real` with value range constraints, while a requirement refers to `measuredPedalForce`. Via the synonym dictionary, both are mapped to the same observable, allowing us to verify that the constraints are aligned and to flag inconsistencies when

declared ranges diverge. In the current implementation, we focus on UML class diagrams.

Nevertheless, our approach is primarily oriented toward requirement specifications, and its effectiveness on models depends on the amount of contextual information present. If models are underspecified or lack semantic annotations, RAG4C may not reliably recover all relevant observables or constraints.

Alias Equality. Let $\mathcal{R}$ be the set of requirements, Q be the set of all qualified raw names of observables extracted during knowledge graph construction (Sec. 3.3), and S be the set of canonical ontology terms produced during the ontology construction (Sec. 3.4). We encode the dictionary function $dict$ as alias equalities between variables:

$$\mathcal{A} = \bigwedge_{dict(q)=s} (v_q = v_s).$$

where v_q and v_s denote the SMT variables associated with q and s.

Final Formula for Consistency Checking. To evaluate the consistency of the requirement set $\mathcal{R}$ with respect to observables shared across disciplines, we define the following satisfiability formula

$$\Psi \triangleq \mathcal{A} \wedge \bigwedge_{r\in\mathcal{R}} \widehat{C}_r, \quad \widehat{C}_r \triangleq \bigwedge_{c\in C_r} dict(c).$$

Ψ expresses whether all requirements are semantically consistent, $\widehat{C}_r$ denote the conjunction of all normalised constraints derived from requirement r. Normalisation rewrites each constraint $c \in C_r$ by replacing all occurrences of a qualified raw name $q \in Q$ by the corresponding variable determined by $dict(q)$.

The formula Ψ is constructed as the conjunction of two parts: (i) the background axioms $\mathcal{A}$ encoding the aliasing equalities and (ii) the normalised constraints of all requirements.

If Ψ is **satisfiable**, there exists an SMT model (or solution) $\mathcal{M}$ assigning concrete values to observables that satisfy all constraints in $\mathcal{R}$, therefore producing a concrete solution evidencing that the set of requirements is consistent. If Ψ is **unsatisfiable**, no SMT model satisfies all constraints, and the requirement set is therefore *inconsistent*. In this case, RAG4C requests an unsat core from the solver and projects it back into human-readable requirements to identify the minimal set that guarantees inconsistency.

Final Formula for Redundancy Checking. To assess whether a distinguished requirement $r^* \in \mathcal{R}$ is logically implied by the remaining requirements with respect to the observables shared across $\mathcal{R}$, we define the following satisfiability formula:

$$\Phi(r^*) \triangleq \mathcal{A} \wedge \bigwedge_{r\in\mathcal{R}\setminus\{r^*\}} \widehat{C}_r \wedge \neg\widehat{C}_{r^*}.$$

The formula $\Phi(r^*)$ evaluates whether the remaining requirements semantically entail the constraints of r^*. For each requirement r, all extracted constraints are first normalised and combined into $\widehat{C}_r$. The formula $\Phi(r^*)$ is then constructed as the conjunction of: (i) the background axioms $\mathcal{A}$ encoding aliasing equalities between observables, (ii) the normalised constraints of all requirements in $\mathcal{R} \setminus \{r^*\}$, and (iii) the negation of the normalised constraints of r^*.

If $\Phi(r^*)$ is **satisfiable**, there exists a SMT model $\mathcal{M}$ that assigns concrete values to observables satisfying all constraints in $\mathcal{R} \setminus \{r^*\}$ while violating r^*. This constitutes a counterexample, showing that r^* is not implied by the remaining requirements. To explain this result, we analyse $\mathcal{M}$ by identifying the observables whose assigned values violate $\widehat{C}_{r^*}$, mapping them back to their original qualified raw names via $dict^{-1}$, and reporting the corresponding requirements and constraints in a human-readable form. If $\Phi(r^*)$ is **unsatisfiable**, no such counterexample exists, and r^* is implied by the remaining requirements. In this case, r^* is redundant under the given abstraction. As in consistency checking, a minimal unsatisfiable core may optionally be requested to support traceability and explanation.

Edge Cases. When observables are not shared across requirement sets, satisfiability results can be misleading. Negating a requirement whose observables do not appear elsewhere yields a trivially satisfiable formula, without indicating any semantic conflict. Conversely, negating a requirement set that does not reference observables used by others may lead to either satisfiability or unsatisfiability, neither of which reliably reflects cross-disciplinary consistency. To filter out such cases, we post-process each potential counterexample: using the inverse mapping $dict^{-1}$, we check whether the involved observables occur in all requirement sets. Only genuinely shared observables can support a meaningful inconsistency claim; otherwise, the result is marked inconclusive.

Use-Case Encoding. Listing 3 shows the SMT-LIB 2 (Barrett et al., 2010) encoding of the constraints extracted from the use case in Sec. 3.1. The solver reports `unsat` with the unsat core `{MRS_001, ERS_001}` witnessing the inconsistency.

Listing 3: SMT-LIB 2 formula for consistency checking.

```
(set-logic QF_NRA)
(set-option :produce-models true)
(set-option :produce-unsat-cores true)
(declare-fun measured_pedal_force () Real)
(declare-fun pedal_effort () Real)
(declare-fun pedal_force () Real)
; Alias equalities
(assert (= pedal_effort measured_pedal_force))
(assert (= pedal_force measured_pedal_force))
; MRS
(assert (! (and (>= pedal_effort 0.0) (<= pedal_effort
300.0)) :named MRS_001))
; ERS
(assert (! (and (>= measured_pedal_force 400.0) (<=
measured_pedal_force 500.0)) :named ERS_001))
; SRS
(assert (! (=> (>= measured_pedal_force 50.0) (>
brake_torque_command 0.0)) :named SRS_001))
(check-sat)
(get-model)
(get-unsat-core)
```

3.5.2 Syntactic Consistency Checking

Syntactic consistency assesses whether observable names and declarations in the requirements align with those in the corresponding models, supporting traceability between requirement-level observables and their model counterparts.

Design models may present views that differ from those expressed in the requirements specifications, and mismatches in observables between requirements and models may therefore be intentional. Accordingly, syntactic checking mainly helps engineers understand structural differences between artefacts and detect mismatches when the model is expected to be complete with respect to the requirements.

We employ the synonym dictionary (Sec. 3.4) to resolve aliases to canonical names before comparison. Violations occur when observables are missing, inconsistently named, or ambiguously represented. If observables are intentionally not shared across artefacts, the result is inconclusive, but the analysis still exposes how requirements and models relate. For example, if a requirement refers to `motor` while the model uses `engine` without a synonym entry, this may indicate either (i) unimplemented behaviour (the model captures only part of the requirement) or (ii) undocumented model semantics (the model contains information not reflected in the requirement).

3.6 Model Generation

RAG4C derives class diagrams from the extracted knowledge graph and renders them using PlantUML (PlantUML, 2025), enabling automated generation and easy integration into development workflows. We use class diagrams since they are sufficient to express the structural aspects relevant to our setting while remaining lightweight and widely understood. Each `System` becomes a UML class. `Observables` associated with a system are rendered as class attributes with their declared types, and each `EnumType` is translated into a UML enumeration. Relationships are mapped to associations: `CONTAINS` links systems, and `CONSUMES` connects systems to the observables they use. Figure 4 illustrates the use case in Sec. 3.1, making explicit the entities and their relationships that are otherwise implicit in the text, thereby helping designers in understanding the requirements.

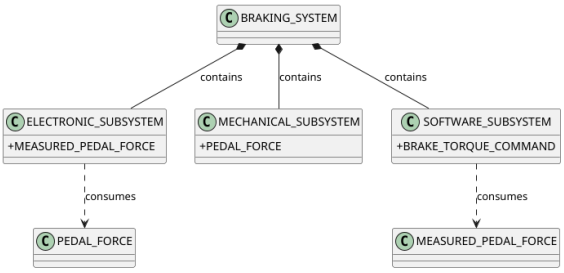

Figure 4: Class diagram view of braking-system use case.

The current approach is limited to structural model generation. We provide a static structure, but we don't claim the full power of UML (associations, classes, qualifiers, etc.). Likewise, we focus on a minimal set of relationships sufficient to support our observable-centric analysis. These choices reflect the goal of isolating the core feasibility of automated model generation, rather than providing full model synthesis. New relationships can be introduced by adjusting the prompt used for knowledge-graph construction and are then automatically propagated to the generated UML views.

3.7 Normalised Observable-Level Specification

Beyond consistency checking, RAG4C derives a *single normalised observable-level specification* from heterogeneous requirements artefacts. Through ontology construction and synonym consolidation, constraints are rewritten using canonical observable terms, projecting all extracted observables into a unified semantic space and yielding a unified constraint set in a common language.

This normalised specification provides a semantic backbone that abstracts from disciplinary terminology while preserving domain semantics. It establishes a unique and unambiguous interpretation of the

requirements and forms the basis for further analysis, including inconsistency diagnosis and potential repair of the original NL requirements. Thus, RAG4C not only checks consistency but also constructs a consolidated, solver-grounded observable view that supports cross-disciplinary traceability and systematic refinement of the requirements.

4 EXPERIMENTS

We evaluate RAG4C on requirement extraction and analysis, assessing observable extraction, synonym construction, scalability, and semantic consistency using precision, recall, and F1-score. To enable verifiability, all datasets, prompts, scripts, and solver encodings are publicly available at https://doi.org/10.5281/zenodo.18351040.

4.1 Datasets (DSs)

We evaluated two types of datasets (DSs): synthetic and industrial requirements.

Synthetic Datasets. We generated synthetic requirement specifications using ChatGPT-5.2 (OpenAI, 2025a), leveraging its NL generation capabilities to produce diverse, realistic text. Each DS consists of three sections (mechanical, electronic, and software) describing an automotive braking system from three viewpoints. Every requirement includes a single numerical constraint, enabling manual identification of observables and synonym relations. Additionally, each document contains at least one intentionally introduced inconsistency.

We generated DSs at varying scales. Small-scale DSs (1 to 7 requirements per section) are used to evaluate precision and correctness, while large-scale DSs (10 to 1000 requirements per section) are intended to test scalability trends. Small DSs are comprehensively validated by experts, establishing a precise ground truth for observable identification, synonym mapping, and consistency. Consequently, these DSs serve as a baseline for analysing trends in evaluation metrics as the data size increases. While large-scale DSs do not undergo the same level of exhaustive manual checking, experts still perform static syntactic checks on the terms and units to provide an approximate range of potential observables and synonyms. In total, 10 requirement documents were generated, including 3,414 requirement statements evenly distributed across three domains.

Industry Datasets. Direct access to proprietary OEM or Tier-1 requirements is typically restricted, so we instead use public deliverables from large collaborative R&D projects, which define realistic, multidisciplinary requirements. Specifically, we focus on the EU-funded AI$_4$CSM project (AI$_4$CSM, 2025), which develops electronic components and systems for next-generation electric, connected, autonomous, and shared vehicles. We select three publicly available AI$_4$CSM deliverables that cover complementary abstraction levels of subsystem requirements, spanning over 150 pages: D1.1 (system-level use cases and KPIs), D1.2 (system design for an AI-based EV demonstrator), and D1.4 (component-level propulsion and energy storage requirements). These requirements are also used to evaluate scalability, and domain experts manually review the resulting reports.

4.2 Evaluation

Ground truth is established from manual annotations applied to synthetic DSs and from chosen subsets of the AI$_4$CSM deliverables. RAG4C processes raw requirements and generates extracted observables, constructed synonyms, and consistency results (either satisfiable models or unsatisfiable cores). We consider two pure LLM approaches, ChatGPT 5.2 and Gemini Flash 2.5 (DeepMind, 2025), and a traditional NLP approach, spaCy, as baselines.

We include LLM-based approaches as baselines because RAG4C also relies on the same underlying ChatGPT for observable extraction. Rather than prompting an LLM directly, RAG4C augments the LLM with graph-based retrieval and a structured prompt. Thus, the comparison reveals the benefit of these additions over using the LLM alone. All LLM experiments used new accounts and fresh sessions, with memory disabled and temperature fixed to minimise context leakage. All models were queried using a standardised prompt: “Given an input specification, output: (i) the set of observables, (ii) the set of synonyms (if any), and (iii) a consistency result indicating whether all requirements are jointly satisfiable together with a justification (e.g., a satisfying assignment or an explanation of conflicts).”

SpaCy is applied using its named-entity recognition, noun-phrase extraction, and candidate terms grouping pipeline. As a general-purpose NLP tool without domain-specific grounding or reasoning, spaCy serves as a conventional NLP baseline, allowing us to contrast our approach with a purely linguistic method that does not rely on embeddings, retrieval, or LLM-based inference.

Metrics (precision, recall, and F1-score) are com-

puted based on *semantic equivalence* with the ground truth rather than strict lexical identity.

Observable Extraction. Table 3 presents the results for both observable extraction and synonym construction in small-scale synthetic DSs. Focusing first on observable extraction, RAG4C achieves very high performance on most DSs, with a macro-average precision of 0.99, recall of 0.98, and F1-score of 0.99. It improves observable extraction quality by a large relative margin (approximately 40%) compared to standalone ChatGPT. This gain primarily stems from our type-specific observable extraction: entities are first identified via LLM-based semantic parsing, then organised into a graph based on their contextual relations. Although the datasets were generated with ChatGPT, it (F1-score of 0.70) performs worse than Gemini (F1-score of 0.94) on extraction. The reason could be that generating requirements and extracting structured observables are different tasks: ChatGPT might over-identify entities (in the worst case, reducing precision to 0.00), whereas Gemini adheres more closely to the extraction rules.

SpaCy exhibits consistently high recall (0.94) but substantially lower precision (0.33). It tends to extract partial noun phrases, specification artefacts, and template-related tokens as observables in the absence of domain-aware grounding or reasoning. This behaviour leads to a high false-positive rate, suggesting limitations of general-purpose NLP tools for unstructured requirements.

We do not compare the default GraphRAG configuration with RAG4C for observable extraction because GraphRAG's built-in prompts support only a limited set of entity types (e.g., Person, Organisation, and Event) and do not extract observables without manual customisation as we did. Consequently, evaluating observable extraction would not yield a meaningful baseline for GraphRAG. Instead, the next section focuses on the downstream task synonym construction, where we compare GraphRAG's clustering mechanism with our agglomerative algorithm using the observables produced by RAG4C.

Synonym Construction. GraphRAG is directly evaluated in this study and serves as the baseline for comparison. In particular, we compare GraphRAG's Leiden clustering mechanism with RAG4C's agglomerative algorithm, using the observables generated by RAG4C. Figure 5 shows metrics for each dataset. In this figure, the results for ChatGPT, Gemini, and spaCy are included as visualisations of the values reported in Table 3, whereas GraphRAG is presented as an explicit baseline. As shown in the figure, RAG4C

Table 3: Precision (P), recall (R), and F1-score (F1) for observable extraction and synonym-group construction.

		RAG4C			ChatGPT			Gemini			spaCy		
DS	**Task**	**P**	**R**	**F1**	**P**	**R**	**F1**	**P**	**R**	**F1**	**P**	**R**	**F1**
1	Obs.	1.00	1.00	1.00	0.00	0.00	0.00	1.00	1.00	1.00	0.23	1.00	0.37
1	Syn.	1.00	1.00	1.00	0.00	0.00	0.00	1.00	1.00	1.00	0.20	1.00	0.33
2	Obs.	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	0.31	0.83	0.45
2	Syn.	1.00	1.00	1.00	0.20	0.20	0.20	0.20	0.20	0.20	0.26	0.80	0.40
3	Obs.	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	0.39	1.00	0.56
3	Syn.	1.00	1.00	1.00	0.71	0.71	0.71	1.00	0.29	0.44	0.35	0.85	0.50
4	Obs.	0.92	0.92	0.92	0.00	0.00	0.00	0.83	0.83	0.83	0.29	1.00	0.45
4	Syn.	0.90	0.90	0.90	0.00	0.00	0.00	0.50	0.10	0.17	0.38	1.00	0.56
5	Obs.	1.00	1.00	1.00	1.00	1.00	1.00	0.87	0.87	0.87	0.33	0.93	0.49
5	Syn.	0.90	0.90	0.90	0.90	0.90	0.90	0.40	0.40	0.40	0.36	0.80	0.50
6	Obs.	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	0.36	0.94	0.52
6	Syn.	1.00	1.00	1.00	0.29	0.18	0.22	0.06	0.06	0.06	0.35	0.82	0.49
7	Obs.	1.00	1.00	1.00	0.91	0.86	0.88	0.91	0.86	0.88	0.39	0.91	0.55
7	Syn.	1.00	1.00	1.00	0.10	0.10	0.10	0.15	0.10	0.12	0.33	0.81	0.47
Avg	Obs.	**0.99**	**0.98**	**0.99**	0.70	0.69	0.70	0.94	0.94	0.94	0.33	0.94	0.49
Avg	Syn.	**0.97**	**0.97**	**0.97**	0.31	0.30	0.30	0.47	0.31	0.34	0.31	0.87	0.47

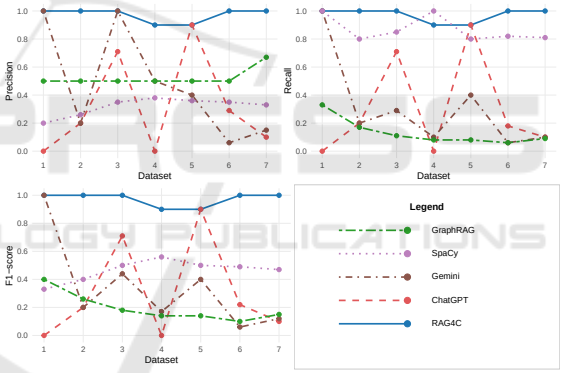

Figure 5: Synonym-grouping performance across datasets.

consistently outperforms GraphRAG across datasets, with an increase in F1-score of up to nearly 380%. This follows from GraphRAG's design: instead of clustering by semantic similarity, it performs topological grouping, where each cluster contains a requirement and its associated observable. As each requirement in our dataset corresponds to a single observable, GraphRAG does not produce synonym groups, leading to poor performance on this task.

Compared with the other tools (ChatGPT, Gemini, and spaCy), RAG4C more than doubles performance, achieving over a 100% improvement in F1 relative to the best baseline on the synonym-grouping task. The performance gap mainly stems from fundamental differences in the grouping mechanisms. ChatGPT and Gemini rely largely on surface-level lexical cues and intuitive guessing, which can lead to inconsis-

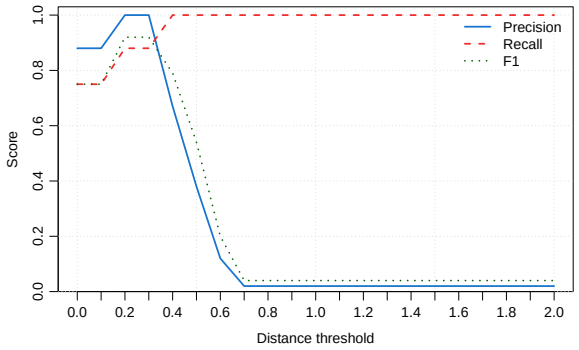

Figure 6: Macro-averaged score across datasets.

tent or incorrect synonym groupings. For example, in the requirement "The vehicle speed limiter shall cap the velocity at 210 km/h," ChatGPT and Gemini often treat vehicle speed and velocity as distinct entities, despite their clear reference to the same physical observable. In contrast, the performance gain of RAG4C stems from its embedding-based agglomerative clustering with a calibrated cosine similarity threshold, which enforces systematic, geometry-driven grouping of observables rather than heuristic language-model guesses. While spaCy's noun-phrase extraction achieves relatively high recall, the absence of domain grounding and semantic clustering leads to fragmented synonym groups and frequent singleton clusters. Consequently, spaCy struggles to capture conceptual equivalence beyond shallow lexical overlap, limiting its effectiveness for synonym-group construction in technical requirement specifications.

Distance Threshold. In addition, we evaluate the impact of the *distance threshold* τ on synonym grouping quality using the small-scale synthetic DSs. For each dataset, we vary τ over the range $[0, 2.0]$ in steps of 0.1, and report macro-averaged metrics at each threshold (Figure 6). At low thresholds ($\tau < 0.2$), precision is high, but recall is low, as only clear synonym relations are captured. As τ increases, recall saturates while precision stays stable, then drops once over-merging occurs. This trade-off peaks in $\tau \in [0.2, 0.4]$, where the F1-score is maximal. Even in the most conservative setting ($\tau = 0$), RAG4C achieves an F1-score of 0.75, corresponding to a $\sim 60\%$ improvement over the best baseline. Table 3 shows detailed synonym group construction results at $\tau = 0.35$.

Consistency Checking. RAG4C aligns with the ground truth on all seven datasets, whereas ChatGPT and Gemini do so on only five, representing an improvement of $\sim 40\%$ over pure LLM approaches.

For instance, in DS 6, they classified the limits "steering wheel torque ≤ 5 Nm" and "motor torque sensor range up to 4.8 Nm" as inconsistent, interpreting the 0.2 Nm measurement gap as unsafe. However, *steering wheel torque* denotes the mechanical input applied by the driver, whereas the *motor torque sensor* measures the actuator-side quantity. The slight discrepancy may reflect calibration margins or sensor tolerances rather than a logical contradiction, and a realisable system instance can still satisfy both requirements. This example highlights a recurring challenge in observable reasoning: distinguishing safety concerns or engineering margins from genuine logical unsatisfiability of constraints.

Table 4: Synthetic and AI$_4$CSM DSs scalability.

Data	Words	#Req	Time [s]	Consistency
A	394	30	25.3	No
B	2,718	300	82.7	No
C	31,347	3,000	414.9	No
AI$_4$CSM	29,024	151	232.3	Yes

In DS 7, the constraints "braking distance ≤ 42 m at 100 km/h" and "regenerative braking implying ≥ 50 m" were also deemed inconsistent. The confusion arises from overlooking operational modes: regenerative braking decelerates the vehicle by converting kinetic energy into electrical energy, typically without full mechanical brake engagement, and is therefore distinct from conventional friction braking initiated via the brake pedal. Since the two constraints govern different braking modes, they are not contradictory and can coexist within a realisable system specification.

These examples show that observable consistency checking requires careful interpretation of domain context, operational modes, and measurement semantics. Even for experts, NL specifications can blur distinctions between measurement, control, and physical behaviour. By grounding constraints in normalised observables and checking formal satisfiability, RAG4C avoids such context-dependent misclassification while correctly identifying genuine cross-disciplinary contradictions.

For large-scale synthetic DSs, all tools reported inconsistencies, which were manually confirmed due to multiple pairs of contradictory requirements. For AI$_4$CSM, all tools reported the specification as consistent. The LLMs justified this primarily by stating that no explicit contradiction could be found, whereas RAG4C established consistency constructively by building valid observable assignments that satisfy all constraints.

Scalability. Table 4 reports runtimes on synthetic DSs (30, 300, and 3,000 requirements) and the AI$_4$CSM DS. Experiments were run on a Linux sys-

tem with an AMD Ryzen 7 PRO 7840U (up to 5.1 GHz) and 60 GB RAM. Runtime scales linearly with the number of entities and their pairwise relations. According to the AI$_4$CSM DS, runtime is strongly influenced by text length and observable density, not just the requirement count. We observed that knowledge-graph construction dominates the overall runtime. By contrast, reasoning with Z3 (de Moura and Bjørner, 2011) (an SMT-LIB compliant tool) contributes only marginally, since the SMT formulas are quantifier-free and lie within the linear theory of reals.

Human Effort. One domain expert required more than two days to manually check the consistency of seven small-scale datasets, whereas RAG4C completed the same checks in approximately two minutes; a second expert then spent about one hour validating the results produced by RAG4C. Given the achieved precision (0.97) and the reduction in manual effort, our experience suggests that automating large parts of requirements interpretation and cross-checking enables engineers to focus on targeted reviews while remaining in the loop. In addition, efficient early inconsistency detection helps prevent the downstream propagation of problematic models and enhances the reliability of MDD.

4.3 Threats to Validity

Use of LLM-Generated Datasets. ChatGPT-5.2 was used to generate the synthetic DSs, and the OpenAI reasoning model 4o was employed for RAG-based knowledge extraction. This setup may introduce evaluation bias, as successive model iterations can exhibit training data overlap and stylistic alignment with self-generated content, potentially simplifying downstream knowledge graph construction. Notably, however, ChatGPT alone does not reliably extract observables or construct accurate synonym groups even on its generated DSs. In contrast, RAG4C consistently achieves superior performance, demonstrating that the observed improvements stem from the proposed pipeline, particularly its explicit observable extraction and synonym grouping mechanisms, rather than from model self-alignment effects.

Framework Maturity. More broadly, RAG4C should be viewed as a general framework for semantic grouping and consistency analysis rather than a finalised solution. As such, its performance is expected to benefit from continued advances in language models, domain adaptation, and richer document representations. Future work will explore these directions, while the current results demonstrate the feasibility and practical value of the underlying approach.

Synthetic Dataset Limitations. The synthetic datasets were used to avoid intellectual property issues and enable controlled expert validation, but they do not fully capture the ambiguity and implicit domain knowledge of industrial documentation. To reduce this gap, we also evaluated RAG4C on requirement documents from the EU-funded AI$_4$CSM project, restricting the analysis to textual content for reproducibility. Extending the framework to multimodal artefacts (e.g., diagrams or tables via OCR or LLM-based extraction) is future work rather than a limitation of the approach.

Human Supervision. Although RAG4C employs optimised clustering strategies and is evaluated under fully automatic execution, expert supervision remains necessary in practice to validate synonym groupings and resolve borderline semantic cases. Rather than replacing human judgment, RAG4C facilitates design engineers by systematically reducing manual analysis effort and highlighting semantically plausible groupings, thereby supporting informed decision-making in safety-critical development processes.

5 CONCLUSION AND FUTURE WORKS

We mitigate key MDD challenges in multidisciplinary systems by focusing on semantic consistency among requirements across heterogeneous artefacts. We introduced observables (domain properties shared across artefacts) as a unifying abstraction to capture and preserve common semantics within diverse requirement specifications. To explore the practical applicability of this concept, we developed RAG4C, an ontology-driven framework that combines RAG and LLMs to extract observables and synonyms from requirements. The framework supports the systematic alignment of NL requirements by identifying shared observables and making their relationships explicit. It supports early inconsistency detection and helps establish a consistent requirements basis for subsequent development activities, improving the reliability and efficiency of MDD.

We plan to explore LLM-based techniques for unit normalisation and dimensional analysis to harmonise semantically equivalent quantities. Collaborations with industry stakeholders or the use of anonymised datasets will support a more representative evaluation.

ACKNOWLEDGEMENTS

This work was funded by the German Research Foundation (DFG) – SFB 1608 – 501798263.

REFERENCES

Abdelnabi, E. A., Maatuk, A. M., and Hagal, M. (2021). Generating uml class diagram from natural language requirements: A survey of approaches and techniques. In MI-STA, pages 288–293.

AI$_4$CSM (2025). AI$_4$CSM – automotive intelligence for connected shared mobility. https://ai4csm.eu/. Accessed: 2026-3-3.

Barrett, C., Stump, A., and Tinelli, C. (2010). The smt-lib standard: Version 2.0. In Workshop on SAT.

Chen, B. et al. (2023). Unleashing the potential of prompt engineering in large language models: a comprehensive review. CoRR, abs/2310.14735.

Clarke, E. M., Grumberg, O., and Peled, D. A. (1999). Model Checking. MIT Press.

de Moura, L. and Bjørner, N. (2011). Z3: An efficient smt solver. TACAS.

DeepMind, G. (2025). Gemini 2.5 flash [generative AI model]. https://docs.cloud.google.com/vertex-ai/generative-ai/docs/models/gemini/2-5-flash. Accessed 2025-12-28.

Edge, D. et al. (2024). From local to global: A graph RAG approach to query-focused summarization. CoRR, abs/2404.16130.

Färber, H. et al. (2025). Observable semantics for characterising consistency between heterogeneous models. In SEFM, volume 16192 of LNCS, pages 110–128.

Feichtinger, K. et al. (2024). Towards formalizing and relating different notions of consistency in cyber-physical systems engineering. In MODELS, pages 915–919.

Ferrari, A., Abualhaijal, S., and Arora, C. (2024). Model generation with llms: From requirements to uml sequence diagrams. In REW, pages 291–300.

Ferrari, A. and Spoletini, P. (2025). Formal requirements engineering and large language models: A two-way roadmap. Inf. Softw. Technol., 181:107697.

Fuxman, A., Pistore, M., Mylopoulos, J., and Traverso, P. (2004). Specification and analysis of early requirements in tropos. Requirements Engineering, 9(2):132–150.

Gao, Y. et al. (2023). Retrieval-augmented generation for large language models: A survey. arXiv preprint arXiv:2312.10997, 2.

Hemmat, A. et al. (2025). Research directions for using LLM in software requirement engineering: a systematic review. Frontiers Comput. Sci., 7.

Honnibal, M. et al. (2020). spaCy: Industrial-strength natural language processing in Python. Zenodo.

Ieva, C. et al. (2018). Discovering program topoi via hierarchical agglomerative clustering. IEEE Trans. Reliab., 67(3):758–770.

Jeleniewski, T. et al. (2025). Consistency verification in ontology-based process models with parameter interdependencies. CoRR, abs/2506.16087.

Ke, R. and Ribeiro, R. M. (2025). Roadmap for using large language models (LLMs) to accelerate cross-disciplinary research with an example from computational biology. CoRR, abs/2507.03722.

Klare, H. et al. (2021). Enabling consistency in view-based system development - The Vitruvius approach. J. Syst. Softw., 171:110815.

Kühn, T. et al. (2023). A formalized classification schema for model consistency. Technical Report 1, KIT.

Liu, Y. et al. (2025). LLM-ACNC: Aerospace requirement texts knowledge graph construction utilizing large language model. Aerospace, 12(6).

Ma, Z. et al. (2025). Bridging natural language and formal specification-automated translation of software requirements to LTL via hierarchical semantics decomposition using llms. In ASE, pages 1208–1220. IEEE.

MinerU (2024). Mineru: Document parsing and ocr framework. https://mineru.net. Accessed: 2025-3-3.

Norheim, J. J. et al. (2024). Challenges in applying large language models to requirements engineering tasks. Design Science, 10:e16.

OpenAI (2025a). Chatgpt 5.2 large language model. https://chat.openai.com. Accessed 2025-12-28.

OpenAI (2025b). OpenAI text embedding model 3 small. https://platform.openai.com/docs/models/text-embedding-3-small. Accessed: 2026-3-3.

Pascual, R. et al. (2024). Formal foundations of consistency in model-driven development. In ISoLA, volume 15221 of LNCS, pages 178–200.

PlantUML (2025). PlantUML: Draw diagrams as text. https://plantuml.com. Accessed 2025-12-28.

Pnueli, A. (1977). The temporal logic of programs. FOCS, pages 46–57.

Uchitel, S., Kramer, J., and Magee, J. (2003). Synthesis of behavioral models from scenarios. IEEE Transactions on Software Engineering, 29(2):99–115.

Verlaine, B. et al. (2012). Towards conceptual foundations for service-oriented requirements engineering: bridging requirements and services ontologies. IET Softw., 6(2):85–102.

Xu, Z. et al. (2018). Towards a term clustering framework for modular ontology learning. In IC3K, pages 178–201.

Zhang, X. et al. (2025). Ontourl: A benchmark for evaluating large language models on symbolic ontological understanding, reasoning and learning. CoRR, abs/2505.11031.