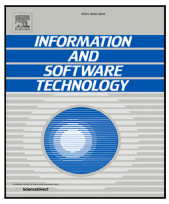
Information and Software Technology 197 (2026) 108154

Contents lists available at ScienceDirect

Information and Software Technology

journal homepage: www.elsevier.com/locate/infsof



Detecting semantic alignments between textual specifications and domain models

Shwetali Shimangaud [a], Lola Burgueño [b,*], Jörg Kienzle [b], Rijul Saini [c]

[a] Amazon AWS, 399 W Georgia St, Vancouver, BC V6B 1Z1, Canada
[b] ITIS Software, Universidad de Málaga, Avda. Cervantes 2, 29071 Málaga, Spain
[c] NAV CANADA, 151 Slater St., Suite 120, Ottawa, Ontario, Canada K1P 5M6

ARTICLE INFO

Keywords:
Large language models
Domain models
Textual requirements
(mis)alignment detection

ABSTRACT

Context: Having domain models derived from textual specifications has proven to be very useful in the early phases of software engineering. However, creating correct domain models and establishing clear links with the textual specification is a challenging task, especially for novice modelers.
Objective: We propose an approach for determining the alignment between a partial domain model and a textual specification.
Methods: To this aim, we use Natural Language Processing techniques to pre-process the text, generate an artificial natural language specification for each model element, and then use an LLM to compare the generated description with matched sentences from the original specification. Ultimately, our algorithm classifies each model element as either *aligned* (i.e., correct), *misaligned* (i.e., incorrect), or *unclassified* (i.e., insufficient evidence). Furthermore, it outputs the related sentences from the textual specification that provide the evidence for the determined class.
Results: We have evaluated our approach on a set of examples from the literature containing diverse domains, each consisting of a textual specification and a reference domain model, as well as on models containing modeling errors that were systematically derived from the correct models through mutation. Our results show that we are able to identify alignments and misalignments with a precision close to 1 and a recall of approximately 78%, with execution times ranging from 18 s to 1 min per model element.
Conclusion: Since our algorithm almost never classifies model elements incorrectly, and is able to classify over 3/4 of the model elements, it could be integrated into a modeling tool to provide positive feedback or generate warnings, or employed for offline validation and quality assessment.

1. Introduction

Textual specifications provide detailed descriptions of a system's desired behavior and constraints in natural language. On the other hand, domain models[1] are abstract, often visual representations of the key concepts and relationships relevant to the system under construction. Domain models have been shown to be very useful for purposes, e.g. communication and checking the completeness of requirements [1]. However, for novice modelers, building domain models and/or understanding how to establish clear, structured links between the two kinds of artifacts is challenging.

Despite numerous efforts in recent years to (semi-)automatically generate domain models from problem descriptions [2–9] as well as the associated traces (also known as links or relations) between the text and the model elements, the resulting models still require human validation before they can be reliably used in a model-driven development process.

As a result, it is essential for software developers to acquire strong modeling skills. The latest curricula from the Association for Computing Machinery (ACM) recognize modeling as a fundamental competency in software engineering education, emphasizing the importance of students becoming proficient in creating, analyzing, and refining software models [10,11]. Unfortunately, theoretical knowledge of modeling languages and semantics alone is not sufficient. Becoming a skilled modeler demands extensive practice—a skill that is challenging to develop.

One way to help modelers improve their modeling skills is by indicating to them the model elements that they modeled correctly as well

* Corresponding author.
E-mail address: lolaburgueno@uma.es (L. Burgueño).
[1] Sometimes also called concept models or conceptual schemas.

https://doi.org/10.1016/j.infsof.2026.108154
Received 20 October 2025; Received in revised form 22 March 2026; Accepted 14 April 2026
Available online 5 May 2026

Specification: [1]We have a **garage** that offers two **types** of **services** for **cars**: **repairs** and **maintenance**. [2]For each **car** that comes to the **garage**, the **first thing** to do is to register **its plate number**. [3]For each **service** provided, we record the **date** and the **type** of **service**. [4]When it comes to **repairs**, we also note which **car part** was fixed – whether it is the **engine**, **transmission**, **lights**, or braking **system**. [5]For **maintenance services**, we need to store the **date** until which the **service** is valid. [6]Note that each **service** happens in a **specific garage**, and every **garage** has its **own address**.

Fig. 1. Running Example: Car Service Specification.

as identifying the mistakes they made together with a rationale. This, however, is a challenging task, since modeling is a creative activity and there is not a single correct domain model for a given situation.

In this paper, we present an approach to identify semantic alignments and misalignments between textual specifications (i.e., requirements) and domain models which can be either complete or only partially complete (a.k.a., partial domain models) by means of LLMs. A piece of specification and an excerpt of a domain model are semantically aligned if they convey the same information. In contrast, they are misaligned if they convey contradictory information.

This paper is structured as follows. Section 2 presents the motivation of our work and a running example. Section 3 outlines the main components of our approach and explains details of the LLM-based classification algorithm. Section 4 presents our evaluation and showcases its results. Section 5 discusses our approach in the broader context of model-driven development. Section 6 presents the related work. Finally, Section 7 outlines the conclusions and future work.

2. Motivation and running example

Fig. 1 shows a short textual requirement specification that describes a car maintenance system, which we are using to illustrate our approach throughout the paper.[2] Based on this textual specification, a modeler might create a domain model such as the one shown in Fig. 2.

In this context, our approach could be used within a modeling tool for different purposes. For example, it could be used to guide the modeler during the modeling activity. To this aim, the modeling tool could highlight the model elements that are misaligned with the textual specification as incorrect. If the user selects the misaligned model element, the tool would display the corresponding misaligned sentence(s) from the textual specification. Similarly, when our approach determines a model element to be aligned with the textual specification, the modeling tool would highlight the element as correct. That way, especially novice modelers would feel more confident about these model elements and as a result can focus on the rest of the model.

While the obvious use of our approach is to assist the modeler during the creation of the domain model, it can also be used after the creation of the model to automatically establish traceability links between the textual specification and the model elements. Especially in a model-driven engineering context, where domain models constantly evolve and are typically refined into architectural and design models and then into code, our approach can significantly ease the burden of establishing and maintaining traceability of textual requirements to models and code.

3. Approach

This section presents a detailed description of the proposed approach. First, we outline the overall architecture, offering a high-level view of the main components and how they interact. Then, we describe each module's functionality.

[2] At this point, please, ignore the fact that part of the text is in bold and underlined.

3.1. Overview

Fig. 3 shows the overall structure of our approach. The system comprises five main components. Component A, the *NLP Specification Preprocessor*, takes as input a requirements specification or problem description written in natural language. This module processes the description using rule-based natural language processing (NLP) to extract noun and relation tokens, which serve as candidates for possible text concepts (tc) and text relationships (tr), along with their traceability mappings.

Component B, the *Model Slicer*, receives the domain model created by the modeler as input, which may still be under construction and thus represent a *partial* domain model. This component traverses all attributes, associations, compositions, inheritance relationships, and enumerations in the domain model and extracts a minimal *model slice* m for each element.

The system passes the outputs of Component A (textual concepts and relationships) and Component B (model slices) to Component C. The Semantic Matcher aligns the textual concepts and relationships produced by the *NLP Specification Preprocessor* with the model elements in the domain model. This process determines which sentences in the textual specification talk about which model elements, and it produces a set of matching specification sentences $\{sS\}$ for each model element.

Component D, the *Model Sentence Generator*, uses a rule-based algorithm to translate each model slice into a corresponding natural language sentence, providing an alternative description mS for each model element.

The outputs of Component C and Component D are finally passed to Component E, *Semantic Alignment Detection*, which uses LLMs to conduct three specific tests: the *Semantic Equivalence* test, the *Contradiction* test, and the *Inclusion* test. These tests classify each model element as *aligned*, *misaligned*, or *unclassified*. Note that instead of using more traditional approaches in components A, B, C and D we could have used LLMs for all components in the pipeline. We preferred the more traditional approaches because (1) we had already created and validated components A and C in the past, and they outperformed the LLMs available at the time of running the experiment, (2) the deterministic implementation of components B and D in Python was straightforward, and (3) using LLMs for all components would have incurred higher financial, energy and time costs.

The following subsections now present the details of each component.

3.2. NLP specification preprocessor

To preprocess the textual specification, we use a simplified version of the analysis introduced in our previous work [12]. In [12], we first use coreference resolution technique to identify all linguistic expressions that refer to the same entity or concept within a given specification, forming a *references map*. This map facilitates pronoun resolution by replacing pronouns with their corresponding concept terms. Subsequently, the approach uses the *spaCy* library [13] to process each statement sentence (sS) and construct a structure of linguistic features. This structure is further processed with rule-based NLP to

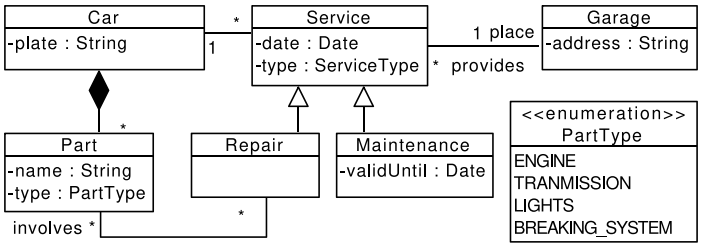


Fig. 2. Running Example: Car Service Domain Model.

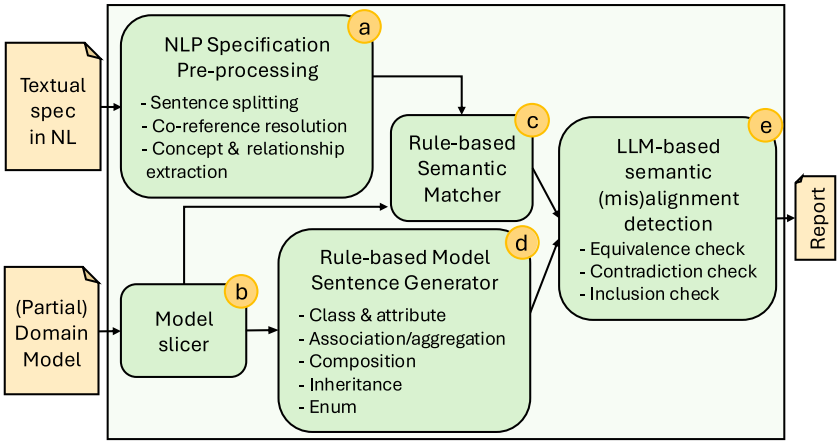


Fig. 3. Approach Overview.

extract *noun chunks* and *relation tokens* as candidates for possible *text concepts* (tC) and *text relationships* (tR), respectively. While the noun chunks are based on the linguistic features such as *noun singular* (NS) part-of-speech (POS) tags, relation tokens are derived from the POS tags like *verb* and *preposition*. In this paper, we reuse and adapt this component to process each word in a noun chunk to determine the final set of text concepts using rule-based NLP. As an example, see the words highlighted in **bold** in the textual specification in Fig. 1. Next, the component identifies the sets of potential source and target text concepts for each token of relation and then uses additional heuristics to identify the final textual relationships. As an example, see underlined words in the textual specification in Fig. 1. The outputs of the component are (1) a set of tuples that map each text concept to the sentences that talk about the concept, and (2) a set of tuples that maps each relation to the source and target concept as well as the set of sentences that talk about the relation. Note that we only use modified sentences from the *references map* for the extraction phase, but form the tuples using the original sentences {sS}. The output of the *NLP Specification Preprocessor* component is formalized in Eq. (1).

$$\begin{aligned}&\text{TextualConcepts} = \{(tC, \{sS\})\}\\&\text{TextualRelations} = \{(tR, tC_{source}, tC_{target}, \{sS\})\}\end{aligned} \tag{1}$$

For example, for our running example, the *NLP Specification Preprocessor* would output:

$$\begin{aligned}\text{TextualConcepts} = {} & (car, \{s1, s2, s4\}), (service, \{s1, s3, s5, s6\}),\\ & (plate, \{s2\}), (date, \{s3\}), \ldots\\ \text{TextualRelations} = {} & (happens, service, garage, \{s6\}),\\ & (offers, garage, service, \{s1\}), \ldots\end{aligned} \tag{2}$$

3.3. Model slicer

On the domain model side, the model slicer component traverses all attributes, associations, compositions, inheritance relationships and enumerations of the provided domain model and extracts a minimal *model slice* for each of them. A minimal model slice, apart from the model element in focus, only contains other model elements that are necessary in order to obtain a valid model. For example, in the case of the domain model for our running example shown in Fig. 2, the model slice extracted for the attribute `date` would include the attribute itself and also the class `Service`, because an attribute must be part of a class and the class sets the context of the attribute. Likewise, the model slice extracted for the association between the class `Service` and the class `Garage` would contain the association itself, the `Service` class and the `Garage` class, as well as the association ends containing the role name `place` with multiplicity `1` and the role name `provides` with multiplicity `*`.

The output of the *Model Slicer* component is a set of tuples that map each model element of interest to a model slice as formalized in Eq. (3).

$$\text{Slices} = \{(m, slice)\}, \text{ where}$$

$$slice = \begin{cases} m \text{ is attribute: } class \cup attribute\\ m \text{ is association: } association \cup ends \cup both\ classes\\ m \text{ is inheritance: } inheritance \cup both\ classes\\ m \text{ is enum literal: } enumeration \cup enum\ literal\\ \text{otherwise: } m \end{cases} \tag{3}$$

For example, for our running example, the *Model Slicer* would output:

$$\begin{aligned}\text{Slices} = {} & (plate, \{Car, plate\}), (date, \{Service, date\}), \ldots\\ & (service \leftrightarrow garage, \{service \leftrightarrow garage,\\ & place, provides, Service, Garage\}), \ldots\end{aligned} \tag{4}$$

3.4. Semantic matcher

The purpose of the *Semantic Matcher* component is to relate the textual concepts and relationships produced by the *NLP Specification*

Preprocessor with the model elements in the domain model, with the ultimate goal of determining which sentences in the textual specification refer to which model elements.

To this aim, the matcher compares each model slice with the textual concepts and relationships from the *NLP Specification Preprocessor* using a set of heuristics, i.e., syntactical word closeness and word similarity of the model element names and the textual concept names. While each slice focuses on one particular model element, all model elements of the slice are considered by the matcher. For example, for associations and inheritance, the connected classes are also used as additional information for the comparison. At this point, we also treat the association ends of associations and compositions separately, i.e., each association end (which includes role name and multiplicity) is considered a standalone model element. Association ends that do not have a multiplicity specified are not considered.

Whenever a match is made between a model element m and a textual concept tC or textual relation tR, the corresponding sentences from the textual specification are associated with the model element. In the end, a set of tuples associating each model element with a set of matched sentences of the textual specification is produced.

$$\text{MatchedSentenceSets} = \{(m, \{sS\})\} \tag{5}$$

For our running example, the *Semantic Matcher* would output:

$$\text{MatchedSentenceSets} = (plate, \{s2\}), (place, \{s1, s6\}), \ldots \tag{6}$$

3.5. Sentence generator

The *Sentence Generator* component constructs sentences in natural language for each model element of interest in the domain model from the corresponding *model slice* using rule-based NLP. We based our rules on the rules proposed by Arora et al. [14] and extended them further to support additional model elements such as enumerations and association role names. The final output of this component is a set of tuples that maps each model element to a generated sentence:

$$\text{GeneratedSentences} = \{(m, mS)\} \tag{7}$$

In the remainder of this subsection, we further explain the rules the *Sentence Generator* uses to build sentences for attributes, associations, compositions, inheritance, and enumerations.

3.5.1. Attributes

We process the word(s) in an attribute name using the *spaCy* library to obtain linguistic features such as nouns and verbs. The text generation algorithm employs different rules to generate a sentence based on these features. For example, the generated sentence for the model slice focusing on the `plate` attribute in the class `Car` in Fig. 2 results in the sentence 'A car has a plate.'.

3.5.2. Association (and aggregation) relationships

Since aggregation is a special type of association relationship whose semantics are not fully defined in the UML [15], our approach treats aggregation in the same way as plain associations. In our approach, one sentence is generated for each association end for which a multiplicity is provided. First, we tokenize the role name using *spaCy* to extract linguistic features such as nouns and verbs. Second, we derive a sentence using the role names. If the role name is absent or the role name does not contain a verb, we use the auxiliary verb 'has' ('have' in case of a plural) to complete the sentence. For example, the *model slice* consisting of an association relationship from the `Service` class to the `Garage` class results in the two sentences 'A service has a place which is a garage.' and 'A garage provides services.'.

3.5.3. Composition relationships

For compositions, we use the wording 'is made up of' to construct the sentence. For example, the *model slice* consisting of the composition relationship between the `Car` and `Part` classes leads to the sentence 'A car is made up of parts.'.

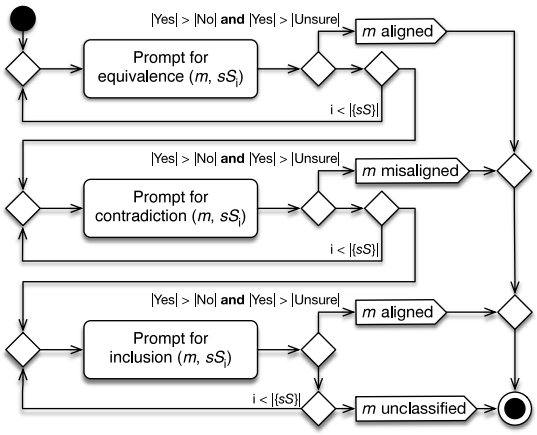


Fig. 4. Classification Workflow.

3.5.4. Inheritance relationships

For inheritance, we generate one sentence for the subclass that is present in the *model slice*. We use the wording 'is a type of'. For example, the *model slice* with the `Service` and `Repair` classes in Fig. 2 results in the sentence 'Repair is a type of service.'.

3.5.5. Enumerations

For enumerations, we generate one sentence for each enumeration literal. We use the wording 'is a/an' to generate the sentence. For example, the model slice that has an enumeration `PartType` with `ENGINE` as an enumeration literal, leads to the sentence 'Engine is a part type.'.

Based on the above explanations, for our running example, the generated sentences would look as follows:

$$\begin{aligned}\text{GeneratedSentences} = {} & (plate, \text{“A car has a plate.”}),\\ & (place, \text{“A service has a place.”}),\\ & (provides, \text{“A garage provides services.”}), \ldots\end{aligned} \tag{8}$$

3.6. LLM-based semantic (mis)alignment detection

The most advanced component of our approach is the *LLM-based Semantic (Mis)Alignment Detection*[3] component (component E in Fig. 3). It queries an LLM using the combined information produced by the *Semantic Matcher* and the *Sentence Generator* to decide, for each model element, whether it is *aligned* with the textual specification (i.e., *correct*), *misaligned* with the textual specification (i.e., *incorrect*), or whether it leaves it *unclassified* (i.e., there is not enough evidence to make a decision). It does so by asking the LLM three sets of prompts as shown in the overview Fig. 4. The detailed algorithm is described in Section 3.6.2.

3.6.1. Prompt engineering

We performed some initial experiments with different LLMs (GPT 3.5, GPT 4 and Gemini) using both zero shot and few shot prompting and empirically determined that: (1) GPT 4 performed the best; (2) few shot prompting did not improve the performance; (3) the responses of the LLM vary considerably depending on the complexity of the statements, the domain, and the exact prompt wording. Due to the first two points, we decided to use GPT 4 with zero shot prompting.

To illustrate the last point, consider the following example, where we need to determine whether the original statement "A football player

[3] We refer to alignments and misalignments collectively as (mis)alignments.

can be sold to another football team." includes the generated statement "A football player plays for a football team.". The LLM's answer to the question "Can the generated statement be *implied* from the original statement?" was "Yes, the generated statement can be implied from the original statement. The fact that a football player can be sold to another football team indicates that they play for a football team in the first place.", whereas the LLM's answer to the question "Can the generated statement be *derived* from the original statement?" was "No, the generated statement cannot be derived from the original statement. While both involve the concept of a football player being part of a football team, the original statement does not inherently confirm the fact stated in the generated statement.".

To deal with these cases with variance in responses, we decided to ask the LLM *multiple* diverse, but semantically equivalent questions and use relative majority voting to determine the final result. This technique is inspired by voting techniques used in fault tolerance, e.g., N-version programming [16].

To distill a set of diverse, yet semantically equivalent prompts for testing *Equivalence*, *Contradiction*, and *Inclusion* we ran a meticulous experimentation and analysis on five representative domain models consisting of a total of 26 model elements. For each type, we started with a larger number of prompts, asking the LLM to answer *Yes*, *No* or *Not Sure*. We then selected the final prompts by eliminating those that provided false positives (i.e., incorrect answers), as well as those that did not introduce any diversity in the responses, i.e., provided exactly the same answers. Finally, since the *Semantic Matcher* component at times determines matches with the sentences from the problem description that are actually unrelated, we also gave preference to the prompts that for unrelated sentences correctly output *Unsure*, and made sure that in the end we have an odd number of prompts.

For example, for the *Equivalence* test, we ended up with the following five prompts:

Equivalence Prompt 1:
Are the following two statements semantically equivalent?
Statement 1: A car has a plate.
Statement 2: For each car that comes to the garage, the first thing to do is to register its plate number.
Equivalence Prompt 2:
Do the following two statements convey the same information? [...]
Equivalence Prompt 3:
Are the following two statements conveying the same meaning? [...]
Equivalence Prompt 4:
Are these statements synonymous? [...]
Equivalence Prompt 5:
Do statement 1 and statement 2 have identical implications? [...]

Checking for contradiction is done in a similar way as checking for equivalence, but using the following questions in the prompt: 'Do these statements contradict each other?', 'Are these statements mutually exclusive?', 'Do these statements clash or conflict with each other?', 'Do these statements negate each other?', 'Are these statements inconsistent?', 'Are these statements in disagreement?', and 'Are these statements incompatible?'.

Finally, for checking inclusion, the questions in the prompts sent to the LLM are: 'Can Statement 1 be inferred from Statement 2?', 'Can Statement 1 be implied from Statement 2?', 'Can Statement 1 be determined from Statement 2?', 'Can Statement 1 be derived from Statement 2?', 'Does Statement 1 logically follow from Statement 2?', 'Can Statement 1 be concluded based on Statement 2?' and 'Does Statement 2 support Statement 1?'.

3.6.2. Algorithm

The pseudocode of the algorithm that performs the sentence comparisons to classify the model elements is described in Algorithm 1.

After initializing the output data structures (lines 10–12), the main `for` loop iterates through all model elements in the domain model. For each model element m, a nested `for` loop goes through all matched sentences from the textual specification (previously identified by the *Semantic Matcher*). For each matched sentence sS, the algorithm asks the LLM whether sS is equivalent to mS, the generated sentence for m (produced by the *Sentence Generator*) by calling the function `checkForEquivalence` on line 19.

How this is done is explained in detail in the pseudocode of the function `checkForEquivalence` on lines 51–63. First, an *odd number* of semantically equivalent prompts are created using the five diverse questions listed in the previous subsection. Then, to determine whether the statements are considered equivalent or not, we use a relative majority vote (see line 62). If the two sentences are not judged to be

```
1  Input:
2    model: ..... {m}
3    generatedSentences (one tuple for each m): ..... {(m, mS)}
4    matchedSentenceSets (one tuple for each m): ..... {(m, {sS})}
5  Output:
6    aligned m and sentences: ..... {(m_aligned, {sS_aligned})},
7    misaligned m and sentences: ..... {(m_misaligned, {sS_misaligned})},
8    unclassified m: ..... {m_unclassified}

9  def classify(model, generatedSentences, matchedSentenceSets) begin
10   alignments ← ∅
11   misalignments ← ∅
12   unclassified ← ∅
13   for m in model do
14     aligned ← (m, ∅)
15     misaligned ← (m, ∅)
16     included ← (m, ∅)
17     for sS in matchedSentenceSets(m) do
18       mType ← getType(m)
19       if checkForEquivalence
20            (sS, generatedSentences(m), mType) then
21         aligned.addSentence(sS)
22       else
23         if aligned.containsNoSentences and checkForContradiction
24              (sS, generatedSentences(m), mType) then
25           misaligned.addSentence(sS)
26         else
27           if checkForInclusion
28                (sS, generatedSentence(m), mType) then
29             included.addSentence(sS)
30           end
31         end
32       end
33     end
34     if aligned.containsSentences() then
35       alignments.addTuple(aligned)
36     else
37       if misaligned.containsSentences() then
38         misalignments.addTuple(misaligned)
39       else
40         if included.containsSentences() then
41           alignments.addTuple(included)
42         else
43           unclassified.add(m)
44         end
45       end
46     end
47   end
49   return alignments, misalignments, unclassified
50 end

51 def checkForEquivalence(sS, mS, mType) begin
52   prompts ← createEquivalencePrompts(mType, mS, sS)
53   yes ← 0, no ← 0, unsure ← 0
54   for pt in prompts do
55     response ← prompt(llm, pt)
56     switch parse(response) do
57       case "Yes": yes = yes + 1
58       case "No": no = no + 1
59       case "Unsure": unsure = unsure + 1
60     end
61   end
62   return yes > no and yes > unsure
63 end
64 // Similar code for the contradiction and inclusion checks
```

Algorithm 1: LLM-based (Mis)Alignment Detection

equivalent, the algorithm proceeds to ask the LLM whether the two sentences contradict each other by calling `checkForContradiction` on line 23. If the two sentences are also not judged to be contradicting, then the algorithm finally asks the LLM whether the meaning of the generated sentence is included in the meaning of the sentence from the textual specification by calling `checkForInclusion` on line 27.

Once all the matched sentences for the model element *m* have been compared with the generated sentence and classified as *aligned*, *misaligned* or *included*, the final decision for *m* is obtained as shown in the pseudocode in lines 34–46. If at least one of the matched sentences was judged to be equivalent, *m* is classified as *aligned*. Only if no sentence was judged equivalent, then, in the case where a contradiction was detected, the model element is judged as *misaligned*. Finally, if no contradiction was detected, but at least one of the matched sentences was judged to include the generated sentence, then *m* is judged to be *aligned*.

The rationale for doing it this way is the following: All the LLMs that we tested our approach with were quite critical when being asked about equivalence. If the sentence from the specification provided only a little bit more detail, or talked about the model element in the context of an action,[4] then the LLM would not consider the sentence equivalent to the generated one. Because of that, our algorithm gives equivalence the highest priority.

Contradiction has second highest priority, which can again be explained by the fact that for the LLM to consider two sentences to contradict they really have to portray the same properties in the same context in an inconsistent way.

Finally, when the LLM judges the generated sentence to be included in one or several of the matched sentences from the textual specification, then we also consider the model element to be aligned. This is because sentences in textual specifications often describe multiple properties at once, while these properties are modeled as separate model elements in the domain model. For example, when the textual specification sentence is a composite sentence containing a conjunction, e.g., 'and', then very likely the textual specification contains more information than the generated sentence, and hence would be judged as not equivalent, but as *including* the generated sentence. This is the case in the sentence 'For each service provided, we record the date and the type of service.' of our running example. This *one* sentence is related to *two* attributes, namely the attributes `date` and `type` of the class `Service` in Fig. 2. Hence, the sentence is not equivalent to, but includes the generated sentences 'A service has a date.' and 'A service has a type.'.

3.7. Tool support

We have developed a proof of concept implementation for our approach and made it available in a Git repository [17]. The UML class diagrams are provided in `.cdm` format, an XML-based format used by the TouchCORE modeling tool [18], which itself is implemented using the Eclipse Modeling Framework. The implementation of the components of our approach, including the extraction of the model elements from the `.cdm` files, as well as the automation of the data processing shown in Fig. 3 is done in Python. The Python script uses NLP libraries such as *spaCy* for NLP preprocessing, *Stanza* for coreference resolution, *Inflect* for operations related to string, *NumPy* for numerical operations, and pandas to store results and CSV file operations. As LLM we have chosen GPT-4o due to its advanced reasoning capabilities. We used the following hyperparameters: temperature = 1, top_p = 1, max_completion_tokens = 2048, frequency_penalty = 0, presence_penalty = 0.

Let us note that the proposed approach remains agnostic to the programming language, the selected libraries and LLM, as these could be replaced by similar or improved technologies.

[4] This can happen often in textual specifications that are based on user stories or use cases.

4. Validation

In the next subsections, we describe our research questions, our empirical setup including the textual specifications and models that we have used for validation and the metrics we have used to answer the research questions. We present the results of our evaluation and discuss the answer to each research question. Finally, we discuss the threats to validity.

For transparency and reproducibility purposes, the dataset, scripts to reproduce our results and excel sheets with all the logs and results of our experiments are available in our repository [17].

4.1. Research questions

To evaluate our approach, we have defined three research questions regarding the correctness (exactness) of the obtained (mis)alignments, the completeness (coverage) of the identified (mis)alignments, and the scalability of our approach.

RQ1. Correctness. Of all the alignments and misalignments identified by our approach, how many are correct?

RQ2. Completeness. Of all the existing alignments and misalignments, how many are correctly identified by our approach?

RQ3. Scalability. How well does our approach scale as the size of the textual specification and corresponding domain model increase?

4.2. Experimental setup

4.2.1. Dataset

To evaluate our approach, we chose a publicly available dataset of 120 textual software requirements written in English [19]. According to the authors, these requirements have been acquired from industry (48%) as well as academia (52%), and carefully selected to cover a wide range of domains (business, finance, health, entertainment, education, technology, energy) and complexity. In the same paper, the authors use the first 30 requirements to compare two domain model extraction tools, and they published the domain models created from those 30 requirements. These textual requirements as well as the models are available on the IEEEDataPort.[5]

For our validation, we have chosen to use the same 30 textual requirements as the authors of [19] used for their experiment, as well as the domain models they provided (when possible). We observed that the domain models they published were either manually created or ChatGPT-generated. Furthermore, some domain models were expressed by means of a class diagram, while others were expressed in structured text.

In order to have a homogeneous dataset for our evaluation consisting of pairs of textual requirements and the corresponding domain model created by a human expert and expressed as a class diagram, for those descriptions whose domain model was generated by ChatGPT, we discarded the models from by [19] and asked a professor (who teaches a modeling course) to create the domain model. We also asked this professor to convert the structured text into a domain model expressed using a class diagram. All the textual requirements and corresponding domain models are available in our repository [17].

We used these models as *ground truth*, i.e., all elements of the model are considered correct/aligned with the textual specification. The resulting pairs of 30 textual requirements and domain models are listed in Table 1, together with information about the number of words and sentences in the text, as well as the number of classes, attributes,

[5] https://ieee-dataport.org/documents/dataset-text-requirements-models.

Table 1
Dataset characterization.

	domain	words	sentences	classes	attr.	assoc./ aggr.	comp.	enums	inh.
R1	Restaurant Management	255	20	10	8	12	1	2	1
R2	Employee Management	227	19	8	13	3	0	1	5
R3	Library	209	22	7	13	4	0	2	2
R4	Galaxy Sleuth Game	710	39	10	12	14	2	0	0
R5	Spy-Robot Game	412	17	8	5	3	0	3	2
R6	Academic Program	129	9	8	13	4	0	0	0
R7	Supermarket	543	21	9	4	6	3	2	1
R8	Hotel Reservation	312	20	7	5	2	0	1	3
R9	BeWell app	433	26	12	5	7	4	0	1
R10	File Manager	83	7	6	6	1	1	0	3
R11	Football team	135	12	3	4	1	0	1	0
R12	Rented Car Gallery	120	12	3	9	3	0	0	0
R13	Course Enrollment	234	15	7	6	6	2	0	0
R14	ATM	164	10	9	0	5	3	0	0
R15	Video Rental	221	17	6	5	4	1	0	0
R16	Cinema	172	9	4	4	3	0	0	0
R17	Timbered House	95	6	6	4	6	0	0	0
R18	Musical store	147	17	4	5	5	0	1	1
R19	Airport	303	20	11	7	15	1	0	0
R20	Monitoring Pressure	92	6	2	2	1	0	0	0
R21	Savings Account	381	24	2	7	1	0	2	0
R22	IPO application	411	29	3	13	2	0	1	0
R23	PIN	422	26	2	4	1	0	1	0
R24	Communication Prefs	311	17	1	3	0	0	1	0
R25	Apple Pay	301	17	3	1	2	0	2	1
R26	Block Card	290	19	3	3	2	0	3	1
R27	Biometric Login	493	30	5	7	2	0	1	2
R28	Donation	361	25	6	3	5	0	0	1
R29	Prepaid Card	339	20	4	4	2	0	2	1
R30	Wallet app	251	17	4	3	2	1	1	0
Avg. ± Std.	All	285.2 ± 148.0	18.3 ± 7.6	5.8 ± 3.0	5.9 ± 3.7	4.1 ± 3.7	0.6 ± 1.1	0.9 ± 1.0	0.8 ± 1.2

associations, compositions, enumerations and inheritance relationships in the domain models.

Since we also want to be able to evaluate the misalignment detection performance of our algorithm, we also needed domain models that contain errors. According to the IEEE Standard Classification for Software Anomalies [20], defects in artifacts such as models or code are due to elements that are either *missing*, *wrong* (i.e., inconsistent, incorrect or ambiguous), or unnecessary (redundant or extraneous). Since our approach aims to detect misalignments of *existing model elements* and should also work on partial models, we do not focus on detecting defects resulting from missing elements. Likewise, since during the process of domain modeling, experienced modelers often discover additional model elements that are not explicitly found in the problem description, we are also not concerned with detecting unnecessary model elements. In other words, our approach for misalignment detection focuses on finding *wrong* model elements, in particular *inconsistent* and *incorrect* ones.

In [21], the authors present a systematic mapping study that identifies 226 different kind of defects that have been detected in 28 primary studies involving model-driven development and classified them. In [22], the same authors focused on defects found in domain models, and defined a set of 50 mutation operators that are applicable to domain models to produce the kind of defects encountered in [21]. From this list of 50, many are related to adding or removing model elements, i.e., they introduce errors due to missing or unnecessary information. After inspection, only the following three mutation operators from [22] are applicable to our specific context:

- 29-WAS2*: Change the type of an association type, i.e., from normal to composite;
- 30-WAS4*: Change the multiplicity of an association member end, i.e., from 0..1 to 0..* or from 1..1 to 1..*, or vice-versa;
- 31-WGE: Change the member ends of a generalization, i.e., change the superclass or the subclass.

To introduce errors into the models in an unbiased way, while at the same time ensuring that we do not neglect a mutation operator, we applied the following strategy. For each model in our dataset and for each mutation operator, we determined the number of model elements in the model that the operator can be applied to. We then randomly applied the mutation operator to 20% of those model elements (rounding up). Due to the fact that the use of a mutation operator could still potentially produce a correct model with respect to the specification (especially when the specification is vague), we have manually checked that the mutated element actually introduced an error, i.e., it deviates from the information provided in the specification.

4.2.2. Evaluation metrics

To answer our research questions, we use several metrics depending on the nature of the question.

To evaluate RQ1 (correctness), we use the precision when calculating alignments (Precision_a), when calculating misalignments (Precision_m) and considering both alignments and misalignments at the same time (Precision). These precisions are defined as follows:

$$\text{Precision}_a = \frac{|\text{CPA}|}{|\text{PA}|} \tag{9}$$

$$\text{Precision}_m = \frac{|\text{CPM}|}{|\text{PM}|} \tag{10}$$

$$\text{Precision} = \frac{|\text{CPA}| + |\text{CPM}|}{|\text{PA}| + |\text{PM}|} \tag{11}$$

where CPA = correctly predicted alignments, PA = predicted alignments, CPM = correctly predicted misalignments, and PM = predicted misalignments.

RQ2 (completeness) can be answered by calculating the recall. Similarly, as we have done for RQ1, we calculate the recall separately for alignments and misalignments, as well as combined. For this, we use the following formulas:

$$\text{Recall}_a = \frac{|\text{CPA}|}{|\text{A}|} \tag{12}$$

Table 2
Results for correct models.

	A	PA	CPA	M	PM	CPM	$Prec_a$	$Prec_m$	Prec	Rec_a	Rec	$F1_a$	$F1_m$	F1
R1	32	24	24	0	0	0	1.00	–	1.00	0.75	0.75	0.86	–	0.86
R2	24	23	23	0	0	0	1.00	–	1.00	0.96	0.96	0.98	–	0.98
R3	28	21	21	0	0	0	1.00	–	1.00	0.75	0.75	0.86	–	0.86
R4	24	14	14	0	2	0	1.00	0.00	0.88	0.58	0.58	0.73	0.00	0.70
R5	14	11	11	0	0	0	1.00	–	1.00	0.79	0.79	0.88	–	0.88
R6	18	17	17	0	0	0	1.00	–	1.00	0.94	0.94	0.97	–	0.97
R7	23	18	18	0	0	0	1.00	–	1.00	0.78	0.78	0.88	–	0.88
R8	17	16	16	0	0	0	1.00	–	1.00	0.94	0.94	0.97	–	0.97
R9	16	11	11	0	0	0	1.00	–	1.00	0.69	0.69	0.81	–	0.81
R10	11	11	11	0	0	0	1.00	–	1.00	1.00	1.00	1.00	–	1.00
R11	7	7	7	0	0	0	1.00	–	1.00	1.00	1.00	1.00	–	1.00
R12	15	11	11	0	0	0	1.00	–	1.00	0.73	0.73	0.85	–	0.85
R13	17	12	12	0	0	0	1.00	–	1.00	0.71	0.71	0.83	–	0.83
R14	10	7	7	0	0	0	1.00	–	1.00	0.70	0.70	0.82	–	0.82
R15	12	8	8	0	0	0	1.00	–	1.00	0.67	0.67	0.80	–	0.80
R16	8	4	4	0	0	0	1.00	–	1.00	0.50	0.50	0.67	–	0.67
R17	14	11	11	0	0	0	1.00	–	1.00	0.79	0.79	0.88	–	0.88
R18	12	10	10	0	0	0	1.00	–	1.00	0.83	0.83	0.91	–	0.91
R19	33	21	21	0	0	0	1.00	–	1.00	0.64	0.64	0.78	–	0.78
R20	3	3	3	0	0	0	1.00	–	1.00	1.00	1.00	1.00	–	1.00
R21	14	12	12	0	0	0	1.00	–	1.00	0.86	0.86	0.92	–	0.92
R22	18	7	7	0	0	0	1.00	–	1.00	0.39	0.39	0.56	–	0.56
R23	8	7	7	0	0	0	1.00	–	1.00	0.88	0.88	0.93	–	0.93
R24	5	2	2	0	0	0	1.00	–	1.00	0.40	0.40	0.57	–	0.57
R25	10	8	8	0	0	0	1.00	–	1.00	0.80	0.80	0.89	–	0.89
R26	13	12	12	0	0	0	1.00	–	1.00	0.92	0.92	0.96	–	0.96
R27	7	6	6	0	0	0	1.00	–	1.00	0.86	0.86	0.92	–	0.92
R28	11	11	11	0	0	0	1.00	–	1.00	1.00	1.00	1.00	–	1.00
R29	11	8	8	0	0	0	1.00	–	1.00	0.73	0.73	0.84	–	0.84
R30	8	4	4	0	0	0	1.00	–	1.00	0.50	0.50	0.67	–	0.67
Sum	443	337	337	0	2	0								
Avg.							1.00	0.00	**0.996**	0.76	**0.77**	0.86	–	**0.86**
Std.							0.00	–	**0.023**	0.15	**0.17**	0.12	–	**0.12**

$$\text{Recall}_m = \frac{|\text{CPM}|}{|\text{M}|} \tag{13}$$

$$\text{Recall} = \frac{|\text{CPA}| + |\text{CPM}|}{|\text{A}| + |\text{M}|} \tag{14}$$

where A = alignments (according to the ground truth), and M = misalignments (according to the ground truth).

To complement the quantitative values obtained for RQ1 and RQ2, we also report on the F_1-score. The F_1-score is the harmonic mean of precision and recall:

$$F_1\text{-score} = 2 \times \frac{\text{Precision} \times \text{Recall}}{\text{Precision} + \text{Recall}} \tag{15}$$

Finally, to study the scalability of our approach (RQ3), we need to take into account two main factors: the size of the textual requirements in number of words (w) and the number of model elements in the domain model (m). We will elaborate the complexity of each component in our approach with respect to those two variables. Furthermore, to give an indication of the performance of our approach, we provide the time per component and the total time to run our approach on a subset of the models.

4.3. Results

4.3.1. Correctness and completeness

After applying our approach to the dataset described above, Tables 2 and 3 present, for each domain, the number of A, PA, CPA, M, PM and CPM as well as their sum, average and standard deviation at the bottom. They also present precision, recall and F_1-measure for each domain, as well as the weighted precision, recall and F_1-measure at the bottom of the table. Furthermore, Table 4 also shows the execution time it took to run our approach both broken down by component as well as the total time.

As reported in Table 2, our approach achieves a precision for classifying alignments ($Prec_a$) of 1 for all the domains. This means that when our approach predicts an alignment, it is always correct. The recall ($Recall_a$) is 0.76 ± 0.15, meaning that out of all the alignments, we identify around 76%. This results in a F_1-measure$_a$ of 0.86 ± 0.12.

Regarding the misalignments, the fact that we are running our approach on a set of correct models means that the number of misalignments (M) is 0 (therefore, the $Recall_m$ is undefined). For all the domains, except for R4, the number of predicted misalignments was 0, meaning that our approach did not make any mistake and incorrectly predicted a misalignment that does not exist (i.e., no true negatives). R4 is the only domain model for which our approach incorrectly predicted two misalignments. As a result, $Prec_m$ is undefined for all the domains except R4 where it is 0. Accordingly, the F_1-measure$_m$ can only be calculated for R4 and it is 0.

Overall, considering both alignments and misalignments, for the set of correct models, our approach has obtained a precision of $0,996 \pm 0,023$, a recall of 0.77 ± 0.17 and a F_1-measure of 0.86 ± 0.12.

We investigated the two cases in which our algorithm misclassified a correct element as incorrect. Both cases occurred in *R4* and were related to the multiplicities of association ends. We discuss those two cases in Section 4.3.3.

Table 3 presents the results for the dataset with mutated domain models, containing approximately 20% of wrong model elements.[6]

For all the models the Precision$_a$ is 1, which means that all the predicted alignments are correct. The $Recall_a$ is 0.78 ± 0.16, which means that our approach made correct predictions for around 78% of the alignments. The F_1-measure$_a$ is 0.88 ± 0.11. It is worth noting that the difficulties with R4 in the first experiment that only contained correct

[6] Note that we could not introduce mutations in the models of R5 and R24 because they would not produce incorrect models w.r.t. the specification.

Table 3
Results for mutated models (approx. 20% of model elements are incorrect).

	A	PA	CPA	M	PM	CPM	$Prec_a$	$Prec_m$	Prec	Rec_a	Rec_m	Rec	$F1_a$	$F1_m$	F1
R1	31	25	25	5	4	4	1.00	1.00	1.00	0.81	0.80	0.81	0.89	0.89	0.89
R2	20	20	20	4	4	4	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00
R3	24	17	17	3	2	2	1.00	1.00	1.00	0.71	0.67	0.70	0.83	0.80	0.83
R4	15	10	10	6	4	4	1.00	1.00	1.00	0.67	0.67	0.67	0.80	0.80	0.80
R6	16	14	14	2	2	2	1.00	1.00	1.00	0.88	1.00	0.89	0.93	1.00	0.94
R7	15	11	11	5	0	0	1.00	–	1.00	0.73	0.00	0.55	0.85	–	0.71
R8	11	10	10	5	2	2	1.00	1.00	1.00	0.91	0.40	0.75	0.95	0.57	0.86
R9	7	6	6	5	3	3	1.00	1.00	1.00	0.86	0.60	0.75	0.92	0.75	0.86
R10	8	8	8	2	2	2	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00
R11	7	7	7	1	1	1	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00
R12	13	9	9	1	1	1	1.00	1.00	1.00	0.69	1.00	0.71	0.82	1.00	0.83
R13	10	8	8	4	2	2	1.00	1.00	1.00	0.80	0.50	0.71	0.89	0.67	0.83
R14	4	3	3	4	3	3	1.00	1.00	1.00	0.75	0.75	0.75	0.86	0.86	0.86
R15	10	7	7	2	1	1	1.00	1.00	1.00	0.70	0.50	0.67	0.82	0.67	0.80
R16	6	3	3	2	1	1	1.00	1.00	1.00	0.50	0.50	0.50	0.67	0.67	0.67
R17	9	8	8	4	4	4	1.00	1.00	1.00	0.89	1.00	0.92	0.94	1.00	0.96
R18	10	8	8	2	1	1	1.00	1.00	1.00	0.80	0.50	0.75	0.89	0.67	0.86
R19	27	16	16	6	5	5	1.00	1.00	1.00	0.59	0.83	0.64	0.74	0.91	0.78
R20	2	2	2	1	1	1	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00
R21	13	12	12	1	0	0	1.00	–	1.00	0.92	0.00	0.86	0.96	–	0.92
R22	16	6	6	2	1	1	1.00	1.00	1.00	0.38	0.50	0.39	0.55	0.67	0.56
R23	8	7	7	1	0	0	1.00	–	1.00	0.88	0.00	0.78	0.93	–	0.88
R25	8	7	7	2	2	2	1.00	1.00	1.00	0.88	1.00	0.90	0.93	1.00	0.95
R26	11	10	10	2	2	2	1.00	1.00	1.00	0.91	1.00	0.92	0.95	1.00	0.96
R27	7	6	6	2	2	2	1.00	1.00	1.00	0.86	1.00	0.89	0.92	1.00	0.94
R28	8	8	8	3	2	2	1.00	1.00	1.00	1.00	0.67	0.91	1.00	0.80	0.95
R29	10	8	8	1	1	1	1.00	1.00	1.00	0.80	1.00	0.82	0.89	1.00	0.90
R30	6	3	3	1	1	1	1.00	1.00	1.00	0.50	1.00	0.57	0.67	1.00	0.73
Sum	301	234	234	74	50	50									
Avg.							1.00	1.00	**1.00**	0.78	0.68	**0.77**	0.88	0.87	**0.87**
Std.							0.00	0.00	**0.00**	0.16	0.29	**0.16**	0.11	0.15	**0.11**

model elements disappeared. This is due to the fact that, coincidentally, the two associations causing the algorithm to make an incorrect prediction were mutated.

$Precision_m$ is also 1 for all the cases of mutated domain models, except for R7, R21 and R23 in which it is undefined because our approach did not predict any misalignment correctly (the denominator is 0). $Recall_m$ is 0.68 ± 0.29, which means that our approach detected correctly around 68% of the misalignments and the F_1-$measure_m$ is 0.87 ± 0.15.

Finally, overall, the Precision is always 1, the Recall is 0.77 ± 0.16 and the F_1-measure is 0.87 ± 0.11.

4.3.2. Root cause analysis

We encountered 2 misclassifications and 104 cases in which our approach was not able to classify the model element. Root cause analysis determined that *Component A* did not make any mistakes; *Component C* mismatched sentences on 8 occasions (7.55%); *Component D* created awkward sentences on 11 occasions (10.38%); on 22 occasions (20.75%) the problem was due to an under-specification in the model (e.g., missing role names), and finally *Component E* misclassified 2 elements and was not able to classify the model element on 63 occasions (61.32%). A table with the details for each model following the metrics defined by Glaser et al. in [23] is provided in an appendix.

4.3.3. Detailed analysis for each model element type

In this subsection, we study whether there is a significant difference in performance of our algorithm for different model element types. We do so by revisiting the results for each category of model element. For space reasons we only present the summary of the analysis, but the detailed tables listing our results per model element category can be found in our repository [17].

Attributes. Our algorithm achieves a precision of 1 for attributes. The recall is 1.0 for 15 out of 30 models, the average recall is 0.83, and the lowest recall, 0.33, appears in two models: `Cinema` and `Communication Pref`. The algorithm displays several consistent patterns of failing to verify the alignment of attributes. In some cases, the attribute is not explicitly mentioned in the textual description, or the noun in the textual description differs from the attribute name in the class diagram (*synonym* pattern). For instance, in the `R21-Savings Account domain` example, the attribute `balance` is not explicitly stated, but it is implicitly understood that a `savings account` class must have a `balance` attribute. In seven cases, the problem was that the LLM applied temporal reasoning (*temporal reasoning pattern*). However, time is typically ignored in domain models, especially when specifying attributes. For example, if the `User` class contains an `email` attribute of type `String`, the sentence generated by the *Rule-based Model Sentence Generator* (component D) is "A user has an email.". In R21, the textual description states: "If the user has not any e-mail address registered with the banking system, the e-mail option should display as 'disabled,' and by default, the SMS option is selected." Here, the user may not possess an email, or might have initially registered with an email and later removed it. In such scenarios, the LLM indicated that more context is necessary to draw a definite conclusion.

Association relationships. For identifying alignments and misalignments of association relationships and multiplicities, the algorithm achieved an average precision of 0.995, an average recall of 0.76, and an average F1-score of 0.865.

As reported already, in the `R4-Galaxy Sleuth Game` domain, two *aligned* associations were wrongly predicted as *misaligned* by the algorithm. In the first case, the textual requirements talked about *two associations between the same two classes* (*multiple associations* pattern). When classifying one of the associations, our approach confused it with the other one: the LLM detected an incorrect multiplicity because it compared the generated sentence with a sentence from the original text that was describing the other association. In the second case,

the model contained an association between the class `Player` and the class `Hypothesis` with multiplicity `*`. The requirements stated that "If an announced hypothesis is incorrect, the player [...] cannot pose hypotheses any longer [...]". The LLM applied again the *temporal reasoning* pattern and concluded that, while the multiplicity between `Player` and `Hypothesis` should be `*` *before* the player makes an incorrect hypothesis, *after* making a wrong hypothesis the multiplicity should be 0. Based on this reasoning the LLM concluded that the multiplicity in the model was incorrect.

We also investigated why the average recall is 0.78, and we discovered 25 cases, in which the generated sentences were ambiguous because *the modeler did not provide role names for the association*. When a role name is missing, we use `has` as a verb to form the sentence, which often was not precise enough for the LLM.

Composition relationships. For composition relationships, the algorithm achieved an average precision of 1, an average recall of 0.58, and an average F1-score of 0.49. In seven cases, the LLM interpreted that the composed class must be a physical part of the composite class (*physical containment* pattern), and if this was not true, it could not reach a clear conclusion. For example, in the `ATM` example (R14), there is a composition from the `Bank` class to the `Account` class. The relevant description states, "Each bank provides its computer to maintain its accounts and process transactions against them." The generated sentence is "Each bank is made up of accounts" but since a `Bank` is not physically made up of `accounts`, the LLM could not identify this as an alignment. In two other cases, the composition was implicit and spread across multiple sentences in the textual description, making it difficult to determine the relationship from a single sentence (*transitive relationship* pattern).

Generalization/specialization relationships. For inheritance relationships, the algorithm achieved an average precision of 1 and an average recall of 0.85. In the `Employee Management` domain, it missed the alignment between `Off-Shift Worker` and `Worker` classes. This occurred because the textual description used the term "non-shift worker" instead of "off-shift worker" (*synonym* pattern), making it difficult for the algorithm to confirm the alignment. In the `Spy-Robot Game` domain, the algorithm did not detect the alignment between the `Enemy` and `Spaceship` classes since the relationship was stated indirectly in the sentence that describes how to play the game: " 'Jump' actions are used to dodge the aliens on the ground and the empty spaces made by the bombs dropped by the spaceships". The LLM did, however, detect the generalization/specialization relationship between `Enemy` and `Alien`, which is more directly expressed in the sentence.

Enumerations. For enumerations, the algorithm achieves a precision of 1, an average recall of 0.72 and an average F1-score of 0.81. The recall is 1.0 for 6 out of 16 models, while the lowest recall of 0.33 appears for the `Library` example.

In two of the cases, the enumeration type name is not precise enough. For example, in the `Library` domain, the `Status` enum has three enumeration literals: `Borrowed`, `Reserved` and `Renewed`. The sentence in the textual description says "An item can be borrowed, reserved, or renewed to extend a current loan". We experimented with renaming the enumeration type to `Loan Status`, and then the LLM detected an alignment. In two cases, the enumeration literals were not explicitly mentioned. For example, in the `Supermarket` domain, the `Order Status` enumeration has two enumeration literals: `Started` and `Completed`. The sentence in the description says "The delivery process begins when the customer first interacts with the service organization and ends when the delivery of the desired service is completed and the customer exits the process". Here, "started" is not mentioned at all, and "completed" is mentioned implicitly (*implicit state* pattern).

Summary. The detailed analysis for each model element type revealed interesting facts about our algorithm. First, the only mistake our algorithm made was when classifying the multiplicities of association ends. For all other model elements, the precision is 100%. This suggests that our algorithm can even be used in a completely automated setting for model elements that are not association ends. Second, about half of the cases where the algorithm was unable to classify the model element fall into some patterns (synonym, temporal reasoning, multiple associations, physical containment, transitive relationship and implicit state). Hence there is an opportunity for further research to determine whether it is possible to teach the LLM how to overcome these recurring classification difficulties, e.g., by using few-shot prompting or fine tuning.

4.4. Scalability

To study the scalability of our approach, we first discuss the complexity of the classification algorithm illustrated with pseudocode in Algorithm 1. The only operation that has a significant performance cost is the querying of the LLM, which happens at least 3 and at most 19 times *per model element* (outer `for loop` lines 13–47 and *per matched sentence* (inner `for loop` lines 17–33).[7] If we denote the number of model elements with m and the number of sentences in the textual specification with s, then the worst-case complexity of our algorithm is $O(ms)$. This is, however, a very pessimistic worst case, as it assumes that the sentence matcher component matches a model element with *every* sentence. In our general experience, which was also confirmed in our experiment, this only happens with small textual requirements. As the textual requirements grow, different paragraphs and sections talk about different concepts, which allows the sentence matcher to avoid matching model elements with unrelated sentences.

Fortunately, the queries associated with each model element are mutually independent. Even for a given model element, each question is independent of the other ones. In fact, new sessions with the LLM are started for each query on purpose so that the LLM answers each questions based on its inherent knowledge of the domain alone, without being influenced by previous questions. Hence it is actually possible to run *all LLM questions in parallel*, which in theory reduces the cost of querying the LLM to a constant: the maximum time it takes *one* LLM query to complete. Whether in practice the queries actually are evaluated by the LLM in parallel or not depends on the LLM and the user conditions. In our experiments with GPT-4o, we initially noticed some throttling happening at the LLM, because of the token and query limits of our OpenAI account. After using the account for a while, though, we got upgraded to a higher usage tier. At the time of running our experiment, our account had reached usage tier 4, which accepts 10k requests per minute and 2M tokens per minute.

Table 4 shows the execution times for each component when running the experiment with the correct models.[8] The most time is spent in component A, the *NLP Specification Preprocessor*, which took 42 s for the smallest model (R10 with 83 words), and more than 11 min for the largest model (R4 with 710 words). Per word, the fastest pre-processing took place in R26 (0.28 s per word), and the longest in R4 (0.93 s per word).

The processing time of component B, the *Model Slicer*, is negligible, as for the biggest model it took in total less than 0.6 ms. Hence these times are omitted from Table 4. The performance of component

[7] 5 times for equivalence checks, 9 times for contradiction, and 7 times for inclusion

[8] The running times for the models containing errors are identical, because the number and kind of model elements are the same, and because of the parallelization the inclusion questions are also asked even if the answers of the LLM in the end are not being used for elements that are classified as misaligned.

Table 4
Component execution times (in seconds).

	A		C		D		E		Total	min	max
	Total	/word	Total	/me	Total	/me	Total	/me		/me	/me
R1	115.89	0.45	135.74	4.242	6.93	**0.22**	35.93	1.12	04:54	0:16	0:40
R2	141.65	0.62	9.06	0.377	9.25	0.39	11.79	**0.49**	02:52	0:08	0:12
R3	117.61	0.56	13.24	0.473	9.33	0.33	**103.38**	3.69	04:04	0:14	**1:43**
R4	**657.65**	**0.93**	58.12	2.422	6.92	0.29	52.99	2.21	**12:56**	**0:27**	0:55
R5	286.97	0.70	7.80	0.557	9.35	0.67	25.02	1.79	05:29	0:15	0:26
R6	65.38	0.51	7.92	0.440	4.40	0.24	12.96	0.72	01:31	0:07	0:13
R7	335.06	0.62	47.70	2.074	**11.06**	0.48	42.59	1.85	07:16	0:20	0:44
R8	158.03	0.51	0.23	0.013	7.90	0.46	36.13	2.13	03:22	0:19	0:36
R9	370.80	0.86	0.21	0.013	7.74	0.48	30.78	1.92	06:50	0:16	0:30
R10	**41.71**	0.50	0.12	0.011	7.15	0.65	9.86	0.90	**00:59**	**0:05**	0:10
R11	48.51	0.36	0.05	0.007	5.00	0.71	9.26	1.32	01:03	**0:05**	0:10
R12	95.71	0.80	19.63	1.308	4.86	0.32	16.65	1.11	02:17	0:08	0:18
R13	167.70	0.72	11.54	0.679	7.11	0.42	29.68	1.75	03:36	0:15	0:30
R14	118.15	0.72	14.36	1.436	5.85	0.59	10.29	1.03	02:29	0:08	0:12
R15	115.75	0.52	3.96	0.330	4.33	0.36	20.26	1.69	02:24	0:10	0:20
R16	107.51	0.63	11.36	1.420	3.92	0.49	12.97	1.62	02:16	0:09	0:14
R17	47.80	0.50	0.13	0.010	4.13	0.30	14.57	1.04	01:07	0:07	0:14
R18	121.10	0.82	0.07	0.006	4.32	0.36	25.91	2.16	02:31	0:12	0:26
R19	208.20	0.69	**203.34**	**6.162**	7.85	0.24	63.66	1.93	08:03	0:16	1:09
R20	59.60	0.65	0.64	0.212	**3.34**	**1.11**	**7.54**	2.51	01:11	0:06	**0:08**
R21	111.19	0.29	0.08	0.006	6.60	0.47	14.74	1.05	02:13	0:09	0:15
R22	147.58	0.36	4.05	0.225	5.83	0.32	36.68	2.04	03:14	0:18	0:37
R23	145.30	0.34	0.05	0.006	5.12	0.64	11.90	1.49	02:42	0:08	0:12
R24	103.18	0.33	**0.03**	0.005	4.03	0.81	8.43	1.69	01:56	0:07	0:19
R25	159.95	0.53	1.50	0.150	6.05	0.61	14.70	1.47	03:02	0:07	0:15
R26	81.86	**0.28**	0.07	0.006	7.47	0.57	13.38	1.03	01:43	0:08	0:13
R27	179.43	0.36	0.13	0.019	4.45	0.64	29.20	**4.17**	03:33	0:16	0:29
R28	128.90	0.36	0.14	0.013	4.08	0.37	30.82	2.80	02:44	0:16	0:30
R29	138.50	0.41	0.06	0.006	5.66	0.51	13.24	1.20	02:37	0:08	0:13
R30	77.08	0.31	**0.03**	**0.003**	4.53	0.57	10.44	1.30	01:32	0:06	0:10

C, the *Semantic Matcher*, varies significantly, depending on whether matches are found immediately or whether more elaborate comparison is needed. For several models it computed the matching sentences quasi instantly (0.003 s per model element for R30), whereas it took 6.162 s per model element for R19. In general, though, the matching is fast, with a median of 0.180 s per model element. Component D, the *Sentence Generator*, took minimally 0.22 s per model element in R1, and maximally 1.11 s in R19. Finally, component E, the *Alignment Detector*, thanks to the parallel execution of the prompts, the times range from 7.54 s for the fastest model (R20) to 1 min and 43 s for the longest one (R3). Per model element, the fastest processing was R2 with 0.49 s and the slowest R28 with 4.17 s.

In the end, the fastest model to be processed *in its entirety* took 59 s. That model, R10, contains 11 model elements and its description has 83 words. The longest model to process was R4 with 12 min and 56 s. R4 has 24 model elements and the description is comprised of 710 words. These processing times include the preprocessing step.

In case our approach is used to process *one model element*, it makes sense to look at the measured times by (1) excluding the preprocessing time of component A, as it needs to be run one time only, (2) summing up the average time it takes components B, C and D to process a model element, and (3) using the minimum or maximum processing times that it takes component E to completely process the prompts of one model element. These times are reported in the last two columns in Table 4. We see that the fastest processing times of 5 s per model element were recorded for models R10 and R11, whereas the slowest processing time of 1min 43 s was recorded during the processing of R3.

4.4.1. Processing time vs. Processing cost tradeoff

Since the precision of the *Alignment Detector* component is so high, one could consider running the alignment check with all sentences, thus circumventing the *Semantic Matcher* component and as a result increase performance. In this case, only NLP cross-reference resolution needs to be done, which would reduce execution further. On the other hand, the number of queries to the LLM would increase significantly, which would result in higher financial and energy costs.

In our experiment, the *Semantic Matcher* was able to find matches for 446 out of 453 model elements, i.e., for 98.39%. This reduced the number of sentence pairs sent to the *Alignment Detector* from all possible 8634 pairs to only 2132 pairs, i.e., reducing the financial and energy costs by a factor of 4.05.

4.5. Answers to the research questions

Answer to RQ1. To answer RQ1 about correctness, we inspect the overall measured precision, shown on the last row in bold in Table 2 and Table 3. On correct models (i.e., models without misalignments), our approach made predictions on 339 model elements, and only misclassified 2 as misaligned (precision of 0.996). On more realistic models, in our case models containing a minimum of 20% of model elements with mistakes, *all our predictions* were correct (precision of 1.0). In other words, if our approach makes a prediction, the prediction is almost certainly correct.

Answer to RQ2. To answer RQ2 about completeness, we turn to the measured recall. As one can see from the last row in Table 2 and Table 3, the average recall is 0.77 and 0.78, which means that on average we were able to make correct predictions for slightly more than 3 out of 4 model elements.

Answer to RQ3. To answer R3 about scalability, we have discussed the complexity of our approach and have reported on the execution times. In the best-case scenario the complexity of our approach is linear and in the worst-case scenario it is quadratic. In our experiments, checking a single model element takes between 5 seconds to 1 min 43 s, whereas applying the approach to an entire model takes 59 s for the smallest model, while the largest model required 12 m 56 s.

4.6. Threats to validity

4.6.1. Conclusion validity

The fact that LLMs are by nature non-deterministic [24] is another threat to conclusion validity. To mitigate this problem, we prompt the LLM and ask about the same pair of model element and matching sentence several times with different questions.

4.6.2. Construct validity

In this study, alignment quality is quantified using precision, recall, and F1, which may not fully capture the relevance or practical usefulness of alignments. For instance, all alignments are treated as equally important, despite the fact that they might have different relevance and impact. Furthermore, the dataset that we have used for evaluation might include specifications written by stakeholders with different levels of domain expertise. This implies variation in the amount and precision of technical jargon, which may affect how reliably our approach detects semantic (mis-)alignments. To shed more light on the practical usefulness of our approach, we plan to carry out an empirical study with human participants in the future.

4.6.3. Internal validity

There are several aspects that could affect the results presented in this paper. The correctness and completeness of our approach have been studied using a third-party dataset [19]. Although the size of the requirements and models in the dataset are relatively small, they cover different and diverse domains as well as a large part of the concepts that can be found in domain models. While the use of a third-party dataset reduces subjectivity and bias, we assumed the models derived from the descriptions to be correct. Any errors, inaccuracies or missing elements in the models may have affected our evaluation. Furthermore, because of the heterogeneity of the domain models in the dataset, we required the help of a human to create some models and transform structured text into class diagrams. The alignments derived from these artifacts may reflect subjective judgments.

4.6.4. External validity

The use of a third-party dataset covering diverse domains and the fact that it includes positive and negative examples (i.e., correct models and models with errors) and models with different sizes gives us some confidence that our results can be generalized to other cases. However, we cannot generalize our results for large specifications and domain models as we know there is some performance penalty when the size of these grows and further research in needed in this line. Finally, we only used one LLM hence we cannot generalize our results to other LLMs.

5. Discussion on applicability

The evaluation results for our approach are very promising, but since the precision is not 100% and the running time is non-negligible, thought must be given on how to best integrate the approach into model-driven development. This section revisits the results, puts them in perspective and, if applicable, provides some ideas on how the algorithm could be improved.

5.1. Correctness

The correctness of the algorithm is excellent, with a precision of 1 for all models and all model elements except for R4, where two multiplicities of associations were incorrectly classified as misaligned. As a result, our approach could be used effectively within a modeling tool *as a modeling assistant*: model elements classified as aligned by our approach would be highlighted as "verified correct", whereas model elements classified as misaligned could be highlighted as "suspected incorrect". In a follow-up step we would run more experiments to confirm that the classification of our algorithm is 100% correct for the following model element categories: attributes, inheritance relationships and enumerations. If this is confirmed, then the algorithm could even be used for their automated validation.

However, because our misalignment precision is not 100%, we imagine that our approach would need to allow the modeler to consult the sentence in the textual specification that contradicts what was modeled, as well as read the reasoning provided by the LLM for flagging the model element as misaligned. The modeler can then decide on their own whether indeed the model element is misaligned, or whether to discard the warning of the assistant.

5.2. Completeness

The recall for all model element categories combined is currently above 0.75, which means that the modeler would see confirmation indications for more than 75% of their model elements if our approach is used as part of an assistant. We believe that it might be possible to improve this result further by specifically targeting the situations that we identified in our experiment in which our approach was not able to provide a classification.

For associations, the most common reason for not being able to provide a classification is that the modeler did not provide role names. Forcing the modeler to put role names on association ends would increase recall considerably.

In many other cases, the LLM was not able to provide a clear answer because domain models usually do not model time constraints, whereas sentences in problem descriptions often describe situations that evolve over time. For example, a garage might provide services during weekdays, but not on weekends. In these situations, the LLM would apply temporal reasoning and doubt whether the sentence in the problem description "A garage provides services to customers during weekdays, e.g., changing the oil or the tires." aligns with the sentence "A garage can provide services" (which implies "a garage can *always* provide services") generated from the domain model. It would be interesting to explore whether the prompts could be modified (or augmented with few shot prompting) to instruct the LLM to not apply temporal reasoning.

5.3. Scalability

In practice, domain modeling is a relatively slow and iterative activity [25]. We believe that the delay introduced by our approach should not affect the experience too negatively when used as part of a modeling assistant that runs in the background. In such a scenario, the preprocessing component A would run only once on the problem description. Components B to E would then produce misalignment annotations as they become available, always progressively calculated on single model elements as the modeler adds them to the model. With this low volume of calls to the LLM, we would be expecting processing times close to the fastest measured times, i.e., below 10 s per model element. This would be similar to background compilation or test execution in an IDE, which also produces warnings or test results incrementally as they become available.

We could also imagine our approach being used in an offline mode, for example, by running overnight to establish traceability links between textual requirements and domain models, or tag the model with a quality estimate that can be used to decide on whether human inspection is required or not. In this case, where speed is not essential, one could even imagine running component E in sequential mode. This would save LLM cost (i.e., energy and money), because as soon as a model element can be classified because the relative majority for a prompt has been reached, the remaining queries for that model element would not have to be sent to the LLM. These savings can be considerable. For example, running the error-free R17 model in parallel mode resulted in costs of US $5.82, whereas running it in sequential mode only cost US $1.79. However, while the parallel execution time for R17 was 1 m 7 s, the sequential execution time was 24 m 44 s.

Finally, it remains to be seen whether our approach is effective on textual requirements and models that are orders of magnitude bigger. Since the cost of querying the LLM is significant, the performance of the algorithm could be improved significantly by using local LLMs as they become more available or smaller LLMs (i.e., LLMs with less parameters such as GPT-4o mini). Even if local or smaller LLMs have limited reasoning capabilities compared to their larger counterparts, ongoing research and advanced fine-tuning methods are significantly enhancing their performance on complex tasks [26,27]. We would also have to investigate the performance cost of the NLP preprocessing step and sentence matcher component as the textual specifications get significantly larger. Since the precision of component E is very high, we could explore replacing those components by a much simpler matcher.

5.4. Modeler experience

Since our approach can be integrated in an assistant, the requirements and expectations from novice modelers and expert modelers might differ. For instance, empirical evidence [28] shows that less experienced modelers often welcome stronger and more explicit guidance, whereas experts typically favor subtle, on-demand support. The integration of the approach into an assistant should take these aspects into account and offer different models of assistance.

5.5. Limitations

Our approach is currently not able to detect certain errors, e.g., using the wrong type when defining attributes.

Missing or unnecessary model elements can also not be detected. Similarly, misplaced attributes can also not be detected, as they are treated as additional ones. Therefore, in order to be able to detect the full spectrum of mistakes, a modeling assistant would have to combine our approach with some of the existing techniques mentioned in the related work.

6. Related work

In this paper, we use NLP and LLMs to verify the alignment between domain model elements and textual specifications. While no prior work directly addresses this specific verification task, several research areas provide relevant context: (semi-)automated domain modeling approaches, LLM applications in domain modeling, schema and ontology matching techniques, and domain model evaluation methods. However, these areas differ from our approach in their objectives, methods, or scope, as we detail below.

6.1. (Semi-)automated domain modeling

Manual construction of domain models from textual specifications is time-consuming and error-prone. Existing research addresses these challenges through recommendations or extraction approaches.

Recommendation-based approaches assist modelers complete partial domain models. For instance, Weyssow et al. [29] fine-tuned a pre-trained RoBERTa [30] language model to recommend meaningful domain concepts relevant to the given modeling context. Burgueño et al. [31] segmented the partial domain model into attributes and relationships, and used a pretrained language model to recommend domain concepts for model completion. Rocco et al. [32] provided recommendations by considering the existing portion of the metamodel as active context, using a graph representation to encode relationships among metamodel artifacts and generating recommendations with a context-aware collaborative filtering technique [33].

Extraction-based approaches generate domain models from textual descriptions. Yang et al. [2] and Saini et al. [3] use a hybrid approach by combining NLP rules with task-specific machine learning models to extract model elements, evaluating outputs against reference solutions. Both approaches evaluate the domain model extracted by the machine learning model against a predefined ‘golden truth’ model. Unlike other methods, Saini et al. acknowledged multiple valid modeling solutions for the same textual description and incorporated interactive user feedback for modeling decisions

In contrast to these approaches, our proposed method in this paper does not aim to automatically extract domain models from textual specifications or to recommend new model elements to the modeler. Instead, we evaluate the alignment of the model elements that the modeler uses in their partial domain model. We classify these elements as correct or incorrect using the textual domain specification as the ‘golden truth’.

6.2. LLMs for domain modeling

As LLMs continue to advance, their benefits are becoming more apparent in various aspects of the development processes. Consequently, model-based engineering (MDE) is increasingly integrating diverse LLMs for a range of modeling tasks, including automated domain modeling [34]. Several studies have incorporated LLMs to extract domain model elements from given domain specifications, without any human interaction or traditional supervised training on a specific domain or task [6–8].

For instance, Chen et al. [34] conduct a comparative study on the use of LLMs for fully automated domain modeling. They assess GPT-3.5 and GPT-4, employing various prompt engineering techniques such as zero-shot, N-shot, and Chain-of-Thought (CoT) prompting on a dataset containing ten diverse domain modeling examples with reference solutions created by modeling experts. They experiment with five cases: one zero-shot prompt, two 1-shot prompts, one 2-shot prompt, and one prompt with CoT reasoning. Their findings show that, while LLMs demonstrate strong domain understanding capabilities, they remain impractical for full automation. The top-performing GPT-4 achieves F1 scores of 0.76 for class generation, 0.61 for attribute generation, and 0.34 for relationship generation. Moreover, the F1-score exhibits higher precision and lower recall, so LLM-retrieved domain elements are often reliable, but many elements are missing. Additionally they observe that generated domain models rarely follow modeling best practices. Furthermore, adding examples to the prompt improves LLM performance in retrieving classes and relations, but including reasoning steps can also decrease performance.

The same authors extend their research in [6] by proposing a multi-step automated domain modeling approach that extracts model elements from problem descriptions using GPT-4. Their approach includes step-by-step instructions and follows an iterative process. They provide a task description to outline the overall domain model generation task, a natural language modeling problem description to specify the problem domain, and an example store with few-shot examples that define model elements and patterns for the multi-step iterative generator (MIG).

Compared to the average results of their previous single-step approach, the multi-step method improves the F1-score for class identification by 22.71% (from 0.6280 to 0.7706) and for relationship identification by 75.18% (from 0.1781 to 0.3120). The F1-score for attributes remains similar, while the F1-score for identifying the Player-Role pattern improves by 10.39%.

Prokop et al. [7] explore the automation of domain modeling using pre-trained large language models (LLMs) by presenting an experimental LLM-based conceptual modeling assistant that collaborates with a human expert. The assistant offers modeling suggestions based on a given textual description of the domain. They use zero-shot and N-shot prompting techniques. Given a textual description and a partial model M, they first query the LLM to generate all classes. For each class, they query the LLM for attributes, and in the next step, they query for associations. A human expert evaluates each suggestion provided by the LLM before integrating it into the domain model. They experiment with the Mistral-7B and Llama-3 8B models. Their experiments show that Llama generally performs better than Mistral, but Mistral with Retrieval-Augmented Generation (RAG) outperforms Llama without RAG.

Chaaben et al. [8] address the completion of partial models using a few-shot prompting technique with the GPT-3 model. Their approach supports both static and dynamic diagram completion. They begin by semantically mapping the domain concepts from the partial model into a structured text representation. In the next step, they query the LLM using this structured representation along with a set of few-shot examples to generate relevant domain concepts.

The recommendation process proceeds in multiple steps. First, they ask the LLM to suggest class names. Next, for each class, they prompt

the LLM to suggest attributes, providing only the attributes from the partial model in the input prompt. Finally, they query for associations between each pair of classes, again using only the relevant information from the partial model in the prompt for that specific association. Their results show a precision of 0.56 and a recall of 0.45 for class name generation, a recall of 0.7 for attribute suggestions, and an accuracy of 0.64 for identifying associations.

Unlike these approaches that process entire textual requirements and use LLMs to generate or recommend relevant model elements, our method partitions the model into small model slices. In addition, our method uses LLMs to compare pairs of sentences, asking for simple yes/no answers. Our method reduces the complexity faced by the LLM, as it only needs to semantically compare two sentences. Our process is fully automated and does not require any human interaction. Additionally, we use zero-shot prompting, eliminating the need to design domain-agnostic sample examples for few-shot prompt settings.

Unlike these approaches that process entire requirements to generate or recommend elements, our method partitions models into small slices and uses LLMs to perform semantic comparisons between specification sentences and model element descriptions. This reduces LLM complexity, requires no human interaction, and uses zero-shot prompting without domain-specific examples.

6.3. Schema and ontology matching

Schema and ontology matching-based methods address the challenge of identifying semantic correspondences between heterogeneous data structures. While these methods share conceptual similarities with our work in terms of semantic alignment, the problem formulation, artifacts involved, and verification objectives differ substantially.

The ontology matching literature distinguishes between two primary methodological approaches. Schema-based matching techniques rely on structural and terminological features of the ontology, such as class labels, property names, and hierarchical relationships between concepts [35,36]. In contrast, instance-based matching methods use entity-level information, including the specific classes and properties instantiated for individual entities, to infer semantic correspondences [37–39]. Both approaches aim to establish mappings between elements of two distinct ontological structures.

Recent work has explored the integration of large language models and pre-trained embeddings to enhance semantic matching capabilities. For instance, Hao et al. propose a system that introduces heterogeneous graph layers to incorporate both the local structure and the global context into concept embeddings using SBERT embedding method [40]. Wang proposes a knowledge graph embedding technique for filtering the candidates generated by a language model. Other approaches use pre-trained SBERT model for encoding labels [41,42]. Ahmed et al. propose prompting LLMs with fragments of the source and target metamodels, identifying correspondences through an iterative process. The fragments to be provided in the prompt are identified based on an initial mapping derived from their elements' definitions [43].

The above approaches differ from our approach in several fundamental aspects. First, ontology matching addresses a correspondence identification problem between two independently constructed and equally valid schemas or metamodels, i.e., a symmetric model-to-model alignment task. In contrast, our work formulates an asymmetric verification problem where domain model elements are evaluated against textual specifications that serve as the ground truth reference. Second, while ontology matching seeks to discover and establish mappings between corresponding elements across two complete structural representations, our approach performs correctness assessment of individual model elements relative to textual specification fragments, yielding binary classification outcomes rather than correspondence sets. These distinctions highlight that while both areas involve semantic alignment, they address fundamentally different problems with distinct inputs, objectives, and evaluation criteria.

6.4. Evaluation of domain models

Evaluating domain models for their coherence and conciseness is a time consuming task. Several approaches propose methods to evaluate domain models in comparison to a 'golden truth' domain model that represents a reference solution. Bian et al. [44] propose a grading algorithm to match student solutions with the template solution using syntactic, semantic, and structural matching strategies. Boubekeur et al. [45] propose an approach that uses simple heuristics and machine learning techniques to evaluate and categorize student submissions of domain models based on their quality.

As already mention in Section 6.1, Yang and Sahraoui [2] propose an approach to generate domain models automatically using natural language patterns and machine learning. The authors then evaluate these generated models with reference solutions using exact matching, relaxed matching, and general matching over classes and relationships.

Singh et al. [46] develop a mistake detection system (MDS) to identify different types of mistake, such as missing elements, extra elements, incorrect elements, and incorrect application of modeling patterns. The MDS compares a student solution with a correct solution provided by the instructor by iterating over all the elements in both solutions and performing various checks. Kasaei et al. [47] propose an approach for equivalence checking to detect semantically similar concepts within domain models. Their approach uses the combination of pre-trained model and dictionary to identify equivalent elements.

In contrast, our approach does not require a reference model solution, because it considers the given textual specification as the 'golden truth' for comparison and evaluation. However, our approach can currently only detect errors with respect to wrong model elements. Missing or unnecessary model elements are currently not detected.

7. Conclusions

In this paper, we presented an approach to detect alignments and misalignments between a textual specification written in natural language and an existing, potentially partial domain model expressed as a UML class diagram. Our approach first uses NLP techniques to preprocess the textual specification. Next, we split the domain model into minimal model slices focusing on one particular model element. Finally, we generate a sentence in natural language for each of the model elements in focus, and query an LLM to discover alignments or misalignments between this sentence and selected sentences from the textual specification that talk about that model element. Ultimately, our approach classifies every model element as *aligned* with the textual specification, *misaligned* with the textual specification, or leaves it *unclassified*.

We evaluated our approach on a publicly available set of 30 requirements specifications from a broad range of domains. Our results show nearly perfect precision for detecting alignments and misalignments. Moreover, we achieve an average recall of 77% for alignments and 78% for misalignments.

We identified several ways in which our approach could be extended. First, we could investigate whether it is possible to refine the prompts for dealing with multiplicities to avoid misclassifications when there are multiple associations between the same two classes, as well as to avoid that the LLM interprets temporal constraints expressed in the text as contradictions. Additionally, since our detailed analysis of the LLM responses detected several patterns (synonym, temporal reasoning, multiple associations, physical containment, transitive relationship and implicit state) that caused the LLM to not be able to classify model elements with enough certainty, we are planning to explore few-shot prompting and/or fine-tuning techniques to train the LLM to recognize these patterns and avoid them. We also want to extend our approach to be able to handle additional model elements. Finally, future work could extend this research to other structural and behavioral UML diagram types.

Regarding scalability, we are planning to investigate ways of speeding up the execution, e.g., using a locally hosted and fine-tuned LLM, as well as experimenting with significantly larger requirements and models.

Last but not least, we would like to use our approach to implement an assistant for a modeling tool in order to be able to perform a user study.

CRediT authorship contribution statement

Shwetali Shimangaud: Writing – review & editing, Writing – original draft, Software, Methodology, Investigation. **Lola Burgueño:** Writing – review & editing, Writing – original draft, Visualization, Validation, Software, Project administration, Investigation, Funding acquisition, Conceptualization. **Jörg Kienzle:** Writing – review & editing, Writing – original draft, Supervision, Software, Project administration, Methodology, Investigation, Conceptualization. **Rijul Saini:** Writing – review & editing, Writing – original draft, Software, Investigation.

Declaration of competing interest

The authors declare the following financial interests/personal relationships which may be considered as potential competing interests:

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

Acknowledgments

We would like to thank Philipp-Lorenz Glasser for this support with the web tool to obtain the complexity metrics presented in the appendix of this paper. This project has been funded by the Spanish Ministry of Science, Innovation, and Universities under contract PID2021-125527NB-I00 and Universidad de Málaga under project JA.B1-17 PPRO-B1-2023-037. Funding for open access charge: Universidad de Málaga / CBUA.

Appendix A. Supplementary data

Supplementary material related to this article can be found online at https://doi.org/10.1016/j.infsof.2026.108154.

Data availability

Data will be made available on request.

References

[1] C. Arora, M. Sabetzadeh, L.C. Briand, An empirical study on the potential usefulness of domain models for completeness checking of requirements, Empir. Softw. Engg. 24 (4) (2019) 2509–2539, http://dx.doi.org/10.1007/s10664-019-09693-x.

[2] S. Yang, H. Sahraoui, Towards automatically extracting UML class diagrams from natural language specifications, in: Proceedings of the 25th International Conference on Model Driven Engineering Languages and Systems: Companion Proceedings, 2022, pp. 396–403.

[3] R. Saini, G. Mussbacher, J.L. Guo, J. Kienzle, Machine learning-based incremental learning in interactive domain modelling, in: Proceedings of the 25th International Conference on Model Driven Engineering Languages and Systems, 2022, pp. 176–186.

[4] M. Robeer, G. Lucassen, J.M.E. Van Der Werf, F. Dalpiaz, S. Brinkkemper, Automated extraction of conceptual models from user stories via NLP, in: 2016 IEEE 24th International Requirements Engineering Conference, RE, IEEE, 2016, pp. 196–205.

[5] J. Francu, P. Hnetynka, Automated generation of implementation from textual system requirements, in: Software Engineering Techniques: Third IFIP TC 2 Central and East European Conference, CEE-SET 2008, Brno, Czech Republic, October 13-15, 2008, Revised Selected Papers 3, Springer, 2011, pp. 34–47.

[6] Y. Yang, B. Chen, K. Chen, G. Mussbacher, D. Varró, Multi-step iterative automated domain modeling with large language models, in: Proceedings of the ACM/IEEE 27th International Conference on Model Driven Engineering Languages and Systems, in: MODELS Companion '24, ACM, New York, NY, USA, 2024, pp. 587–595, http://dx.doi.org/10.1145/3652620.3687807.

[7] D. Prokop, Š. Stenchlák, P. Škoda, J. Klímek, M. Nečaskỳ, Enhancing domain modeling with pre-trained large language models: An automated assistant for domain modelers, in: International Conference on Conceptual Modeling, Springer, 2024, pp. 235–253.

[8] M.B. Chaaben, L. Burgueño, H. Sahraoui, Towards using few-shot prompt learning for automating model completion, in: 2023 IEEE/ACM 45th International Conference on Software Engineering: New Ideas and Emerging Results, ICSE-NIER, IEEE, 2023, pp. 7–12.

[9] J. Silva, Q. Ma, J. Cabot, P. Kelsen, H.A. Proper, Application of the tree-of-thoughts framework to LLM-enabled domain modeling, in: W. Maass, H. Han, H. Yasar, N. Multari (Eds.), Conceptual Modeling, Springer Nature Switzerland, Cham, 2025, pp. 94–111.

[10] T.J.T.F. on Computing Curricula, Curriculum guidelines for undergraduate degree programs in software engineering, 2015.

[11] A.N. Kumar, R.K. Raj, S.G. Aly, M.D. Anderson, B.A. Becker, R.L. Blumenthal, E. Eaton, S.L. Epstein, M. Goldweber, P. Jalote, D. Lea, M. Oudshoorn, M. Pias, S. Reiser, C. Servin, R. Simha, T. Winters, Q. Xiang, Computer Science Curricula 2023, ACM, New York, NY, USA, 2024.

[12] R. Saini, G. Mussbacher, J.L. Guo, J. Kienzle, Automated, interactive, and traceable domain modelling empowered by artificial intelligence, Softw. Syst. Model. 21 (3) (2022) 1015–1045.

[13] I. Montani, M. Honnibal, spaCy: Industrial-strength natural language processing in Python, 2018, (Accessed 18 March 2025).

[14] C. Arora, M. Sabetzadeh, L. Briand, F. Zimmer, Extracting domain models from natural-language requirements: approach and industrial evaluation, in: Proceedings of the ACM/IEEE 19th International Conference on Model Driven Engineering Languages and Systems, MODELS '16, ACM, New York, NY, USA, 2016, pp. 250–260, http://dx.doi.org/10.1145/2976767.2976769.

[15] OMG, OMG Unified Modeling Language, Version 2.5, Tech. rep. Object Management Group, 2015.

[16] L. Chen, A. Avizienis, N-Version programming: A fault-tolerance approach to reliability of software operation, in: Twenty-Fifth International Symposium on Fault-Tolerant Computing, 1995, ' Highlights from Twenty-Five Years', 1995, pp. 113–119.

[17] S. Shigmangaud, L. Burgueño, J. Kienzle, R. Saini, Software repository for the paper: Detecting semantic alignments between textual specifications and domain models, 2025, https://github.com/atenearesearchgroup/semantic-alignment-detection. (Accessed 18 October 2025).

[18] M. Schiedermeier, B. Li, R. Languay, G. Freitag, Q. Wu, J. Kienzle, H. Ali, I. Gauthier, G. Mussbacher, Multi-Language Support in TouchCORE, in: 2021 ACM/IEEE International Conference on Model Driven Engineering Languages and Systems Companion, MODELS-C, 2021, pp. 625–629, http://dx.doi.org/10.1109/MODELS-C53483.2021.00096.

[19] F. Bozyigit, T. Bardakci, A. Khalilipour, M. Challenger, G. Ramackers, Ö. Babur, M.R.V. Chaudron, Generating domain models from natural language text using NLP: a benchmark dataset and experimental comparison of tools, Softw. Syst. Model. 23 (6) (2024/12/01) 1493–1511, http://dx.doi.org/10.1007/s10270-024-01176-y.

[20] IEEE, IEEE standard classification for software anomalies, 2010, pp. 1–23, http://dx.doi.org/10.1109/IEEESTD.2010.5399061, IEEE Std 1044-2009 (Revision IEEE Std 1044-1993).

[21] M.F. Granda, N. Condori-Fernández, T.E. Vos, O. Pastor, What do we know about the defect types detected in conceptual models? in: 2015 IEEE 9th International Conference on Research Challenges in Information Science, RCIS, 2015, pp. 88–99, http://dx.doi.org/10.1109/RCIS.2015.7128867.

[22] M.F. Granda, N. Condori-Fernández, T.E.J. Vos, O. Pastor, Mutation operators for UML class diagrams, in: S. Nurcan, P. Soffer, M. Bajec, J. Eder (Eds.), Advanced Information Systems Engineering, Springer International Publishing, Cham, 2016, pp. 325–341.

[23] P.-L. Glaser, L. Burgueño, D. Bork, A benchmarking framework for model datasets, 2026, arXiv:2603.05250, URL https://arxiv.org/abs/2603.05250.

[24] Y. Song, G. Wang, S. Li, B.Y. Lin, The good, the bad, and the greedy: Evaluation of LLMs should not ignore non-determinism, 2024, arXiv:2407.10457.

[25] S.J. Ali, I. Reinhartz-Berger, D. Bork, How are LLMs used for conceptual modeling? An exploratory study on interaction behavior and user perception, in: W. Maass, H. Han, H. Yasar, N. Multari (Eds.), Conceptual Modeling, Springer Nature Switzerland, Cham, 2025, pp. 257–275.

[26] Y. Tian, Y. Han, X. Chen, W. Wang, N.V. Chawla, Beyond answers: Transferring reasoning capabilities to smaller LLMs using multi-teacher knowledge distillation, 2024, arXiv:2402.04616.

[27] M.J.J. Bucher, M. Martini, Fine-tuned 'Small' LLMs (still) significantly outperform zero-shot generative AI models in text classification, 2024, arXiv:2406.08660.

[28] M.B. Chaaben, L. Burgueño, I. David, H. Sahraoui, On the utility of domain modeling assistance with large language models, ACM Trans. Softw. Eng. Methodol. (ISSN: 1049-331X) 35 (4) (2026) http://dx.doi.org/10.1145/3744920.

[29] M. Weyssow, H. Sahraoui, E. Syriani, Recommending metamodel concepts during modeling activities with pre-trained language models, Softw. Syst. Model. 21 (3) (2022) 1071–1089.
[30] Y. Liu, M. Ott, N. Goyal, J. Du, M. Joshi, D. Chen, O. Levy, M. Lewis, L. Zettlemoyer, V. Stoyanov, RoBERTa: A robustly optimized BERT pretraining approach, 2019, arXiv:1907.11692.
[31] L. Burgueño, R. Clarisó, S. Gérard, S. Li, J. Cabot, An NLP-based architecture for the autocompletion of partial domain models, in: International Conference on Advanced Information Systems Engineering, Springer, 2021, pp. 91–106.
[32] J. Di Rocco, D. Di Ruscio, C. Di Sipio, P.T. Nguyen, A. Pierantonio, MemoRec: a recommender system for assisting modelers in specifying metamodels, Softw. Syst. Model. 22 (1) (2023) 203–223.
[33] J.B. Schafer, D. Frankowski, J. Herlocker, S. Sen, Collaborative filtering recommender systems, in: The Adaptive Web: Methods and Strategies of Web Personalization, Springer, 2007, pp. 291–324.
[34] K. Chen, Y. Yang, B. Chen, J.A. Hernández López, G. Mussbacher, D. Varró, Automated domain modeling with large language models: A comparative study, in: 2023 ACM/IEEE 26th International Conference on Model Driven Engineering Languages and Systems, MODELS, 2023, pp. 162–172, http://dx.doi.org/10.1109/MODELS58315.2023.00037.
[35] A. Sharma, A. Patel, S. Jain, LSMatch and LSMatch-multilingual results for OAEI, 2022, OM@ ISWC.
[36] D. Faria, M.C. Silva, P. Cotovio, P. Eugénio, C. Pesquita, Matcha and Matcha-DL results for OAEI 2022, in: OM@ ISWC, 2022, pp. 197–201.
[37] D. Ayala, I. Hernández, D. Ruiz, E. Rahm, Towards the smart use of embedding and instance features for property matching, in: 2021 IEEE 37th International Conference on Data Engineering, ICDE, IEEE, 2021, pp. 2111–2116.
[38] H. Belhadi, K. Akli-Astouati, Y. Djenouri, J.C.-W. Lin, Data mining-based approach for ontology matching problem, Appl. Intell. 50 (4) (2020) 1204–1221.
[39] O. Fallatah, Z. Zhang, F. Hopfgartner, The impact of imbalanced class distribution on knowledge graphs matching., in: OM@ ISWC, 2022, pp. 1–12.
[40] J. Hao, C. Lei, V. Efthymiou, A. Quamar, F. Özcan, Y. Sun, W. Wang, Medto: Medical data to ontology matching using hybrid graph neural networks, in: Proceedings of the 27th ACM SIGKDD Conference on Knowledge Discovery & Data Mining, 2021, pp. 2946–2954.
[41] D. Kossack, N. Borg, L. Knorr, J. Portisch, TOM matcher results for OAEI 2021, in: CEUR Workshop Proceedings, vol. 3063, RWTH Aachen, 2022, pp. 193–198.
[42] Y. Peng, M. Alam, T. Bonald, Ontology matching using textual class descriptions, in: International Workshop on Ontology Matching, 2023.
[43] N. Ahmed, H.C. Kwok, M. Hamdaqa, W.K.G. Assunção, SMATCH-M-LLM: Semantic similarity in metamodel matching with large language models, in: 2025 IEEE/ACM 22nd International Conference on Mining Software Repositories, MSR, 2025, pp. 199–210, http://dx.doi.org/10.1109/MSR66628.2025.00040.
[44] W. Bian, O. Alam, J. Kienzle, Automated grading of class diagrams, in: 2019 ACM/IEEE 22nd International Conference on Model Driven Engineering Languages and Systems Companion, MODELS-C, 2019, pp. 700–709, http://dx.doi.org/10.1109/MODELS-C.2019.00106.
[45] Y. Boubekeur, G. Mussbacher, S. McIntosh, Automatic assessment of students' software models using a simple heuristic and machine learning, in: Proceedings of the 23rd ACM/IEEE International Conference on Model Driven Engineering Languages and Systems: Companion Proceedings, MODELS '20, ACM, New York, NY, USA, 2020, pp. 1–10, http://dx.doi.org/10.1145/3417990.3418741.
[46] P. Singh, Y. Boubekeur, G. Mussbacher, Detecting mistakes in a domain model, in: Proceedings of the 25th International Conference on Model Driven Engineering Languages and Systems: Companion Proceedings, MODELS '22, ACM, New York, NY, USA, 2022, pp. 257–266, http://dx.doi.org/10.1145/3550356.3561583.
[47] M.-S. Kasaei, M. Sharbaf, A. Fatemi, B. Zamani, AI-driven approach to detect equivalent elements within domain models, in: 2024 15th International Conference on Information and Knowledge Technology, IKT, 2024, pp. 26–30, http://dx.doi.org/10.1109/IKT65497.2024.10892739.