

Why Do LLMs Fail at OCL Generation? A Graph Reasoning Perspective

Hamza Attarwala ✉️🏠^{ID}

Polytechnique Montréal, Montréal, QC, Canada

Moataz Chouchen ✉️🏠^{ID}

Concordia University, Montréal, Canada

Mohammad Hamdaqa ✉️🏠^{ID}

Polytechnique Montréal, Montréal, QC, Canada

Omar Alam ✉️🏠^{ID}

Trent University, Peterborough, ON, Canada

Abstract

Background. Large Language Models (LLMs) are increasingly used to generate Object Constraint Language (OCL) constraints from natural language specifications and UML class diagrams. However, existing work mainly focuses on improving accuracy, with limited understanding of why these models fail.

Aims. This study investigates factors associated with LLM failures in OCL generation by examining the task from a graph-reasoning perspective.

Method. We conduct an empirical evaluation using the PathOCL dataset across six LLMs. We analyze the relationship between OCL correctness and UML structural properties (e.g., navigation depth and model complexity), lexical similarity, prompt ordering strategies, and graph-aware prompting.

Results. We find that, within the evaluated dataset and models, OCL generation correctness is negatively associated with navigation depth and structural complexity. Lexical similarity provides limited explanatory power, while textual ordering is associated with differences in performance. Graph-based prompting yields partial improvements but does not eliminate errors involving structural navigation.

Conclusions. The findings are consistent with structural demands being an important contributor to OCL generation difficulty, while not establishing graph reasoning as the sole or primary cause of failure.

2012 ACM Subject Classification Software and its engineering → Unified Modeling Language (UML)

Keywords and phrases Object Constraint Language (OCL), Unified Modelling Language (UML), Model-Driven Engineering (MDE), Large Language Models (LLM), Graph Reasoning

Digital Object Identifier 10.4230/LIPIcs.ESEM.2026.48

Category Technical Track Paper

Supplementary Material *Software:* <https://doi.org/10.5281/zenodo.20279636>

1 Introduction

Large Language Models (LLMs) are increasingly used to automate software engineering tasks that translate natural language (NL) specifications into formal representations. One important task is the generation of Object Constraint Language (OCL) expressions from NL specifications and UML class diagrams [25, 1, 6, 8]. Accurate OCL generation could



© Hamza Attarwala, Moataz Chouchen, Mohammad Hamdaqa, and Omar Alam;
licensed under Creative Commons License CC-BY 4.0

20th International Symposium on Empirical Software Engineering and Measurement (ESEM 2026).

Editors: Robert Feldt, Maria Paasivaara, Daniel Mendez, Stefan Wagner, and Marvin Muñoz Barón;

Article No. 48; pp. 48:1–48:21



Leibniz International Proceedings in Informatics
Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

reduce manual effort in model-driven engineering and improve specification consistency [9, 7]. However, recent studies show that LLMs frequently generate erroneous OCL constraints even when advanced prompting and fine-tuning techniques are used [1, 28, 21].

Prior work has mainly focused on improving OCL generation through prompt engineering, fine-tuning, and dataset construction [1, 2, 28], with most efforts targeting incremental performance improvements. As a result, they provide limited insight into *why* LLMs fail. In particular, little attention has been given to the role of structural reasoning over UML models, despite its fundamental importance for correct OCL generation [35].

A UML class diagram can naturally be represented as a graph, where classes correspond to nodes and associations correspond to edges [19, 43]. Generating OCL constraints often requires multi-hop navigation across this graph to identify relevant entities and relationships. This suggests that OCL generation is not merely a language translation task, but also a graph reasoning problem.

Recent graph reasoning benchmarks show that LLMs struggle with multi-step reasoning over graph structures, especially as graph depth and complexity increase [37]. These findings motivate the following question: *Do similar structural reasoning limitations explain failures in LLM-based OCL generation using UML models?*

To investigate this question, we study how UML structural properties and non-structural lexical cues influence LLM-generated OCL constraints. Specifically, we address the following research questions:

RQ1. How do UML structural properties influence LLM-based OCL generation?

More specifically, **RQ1.1** examines how structural navigation depth affects OCL correctness, while **RQ1.2** investigates whether OCL generation degrades as UML structural complexity increases.

RQ2. To what extent do lexical cues influence LLM-based OCL generation? Here, **RQ2.1** examines how lexical similarity between natural-language specifications and UML elements influences generated OCL, whereas **RQ2.2** investigates whether the ordering of classes and associations in the textual UML representation affects OCL generation.

RQ3. How can OCL generation be improved, and which errors persist? Specifically, **RQ3.1** evaluates whether advanced prompting strategies, such as Chain-of-Thought, Few-shot, and Build-a-Graph, improve OCL generation quality, while **RQ3.2** identifies the types of OCL generation errors that occur across the experimental conditions.

Based on these questions, the core contributions of this paper are fourfold: (1) we conceptualize UML-based OCL generation as a graph reasoning problem and relate observed failures to known LLM limitations in graph reasoning; (2) we empirically evaluate the effects of structural complexity, navigation depth, lexical similarity, and prompt design on OCL generation; (3) we assess advanced prompting strategies, including Chain-of-Thought and Build-a-Graph prompting; and (4) we identify recurring error types, including invalid navigation, incorrect OCL usage, hallucinated content, and syntactically valid but semantically incorrect expressions.

Within the evaluated dataset and models, our results show a negative association between OCL generation correctness and increasing UML structural depth and complexity, consistent with patterns reported in prior graph-reasoning studies. We find limited evidence for a broad lexical-bias explanation: the observed failures cannot be explained by superficial lexical similarity alone, although misleading class-level lexical cues can still be associated with lower correctness in some cases. Advanced prompting strategies provide only partial improvements. Overall, this study shifts the focus from improving OCL generation accuracy toward examining factors associated with LLM failures in model-driven engineering tasks.

2 Motivation

Recent work has shown that Large Language Models (LLMs) struggle with reasoning tasks involving graph-structured data, when the graph is expressed entirely in natural language [37, 15, 22, 29]. Across graph reasoning benchmarks, LLMs often generate fluent intermediate explanations while failing to correctly reason about structural relationships such as connectivity, traversal, and path optimization. These findings are particularly relevant to UML-based OCL generation, since generating a valid OCL constraint frequently requires navigating multiple associations and maintaining consistency with the underlying UML graph structure.

Listing 1 shows one representative example is reported in the NLGraph benchmark [37], which evaluates LLMs on graph reasoning tasks formulated in natural language. In this task, an LLM is asked to identify whether a path between two nodes exist.

Listing 1 Path reasoning failure from [37].

```
Question: "Is there a path, in this undirected graph, that exist
between node 4 and node 5? If yes, give the path."
Graph connections:
0 → 4, 1 → 6, 3 → 5, 2 → 6, 1 → 3, 2 → 5, 5 → 6, 1 → 2
Correct path: No path exist that connect Node 4 and Node 5
LLM output: Yes, a path between Node 4 and Node 5 does exist -
4 → 0 → 6 → 2 → 5
```

The path generated by the model is not valid. The example illustrates an important limitation: the model fails to correctly captures local graph connectivity, and fails to reason about the graph structure. Other work additionally demonstrates that LLM performance is sensitive to how graph information is presented in text. For example, Ge et al. [15] show that simply reordering graph edges in the prompt can substantially change reasoning accuracy, even when the underlying graph remains identical.

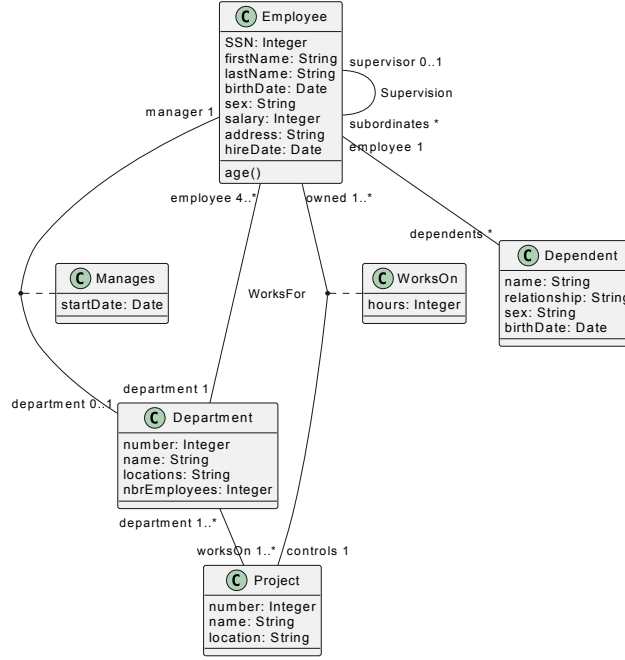
These limitations are directly reflected in OCL generation from UML class diagrams. Although OCL is a formal specification language, producing correct constraints requires reliable navigation over the underlying UML graph, including identifying valid association paths, preserving navigability constraints, and maintaining consistency across multi-step traversals between classes. Listing 2 illustrates this issue through an example generated by *Claude Sonnet 4.5* for the natural language specification: *Only department managers can work less than 5 hours on a project*, based on the UML class diagram in Figure 1, where the resulting OCL expression is incorrect due to failures in maintaining valid structural navigation.

Listing 2 OCL expression generated by LLM.

```
context WorksOn inv: self.hours < 5 implies
self.employee.department.manager = self.employee
```

The generated constraint incorrectly attempts to navigate through an association named `employee` from the `WorksOn` context class. Similar to the NLGraph example, the model produces syntactically plausible output while failing to preserve the underlying graph structure of the problem domain.

Motivated by these findings, RQ1 investigates whether the structural properties of UML class diagrams are associated with OCL generation performance. Prior graph-reasoning studies report declining LLM performance as reasoning paths become longer and graph structures become larger or more densely connected [37, 31]. We therefore examine whether OCL correctness decreases as UML navigation depth and structural complexity increase.



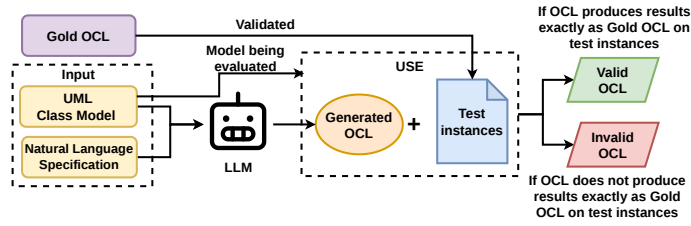
■ **Figure 1** A class diagram from the EmploymentAgency Domain.

RQ2 examines two non-semantic characteristics of the model representation that may also be associated with generation performance. Prior studies suggest that LLM outputs can be affected by superficial lexical cues, such as mention frequency, rather than consistently preserving graph structure [17, 37]. Accordingly, we investigate whether lexical similarity between the natural language specification and UML elements is associated with OCL correctness. Since graph-task performance can also vary when the same structural information is presented in a different textual order [15], we additionally investigate whether alternative orderings of UML classes and associations affect the generated constraints. Finally, motivated by evidence that stepwise processing and intermediate graph construction can partially mitigate failures on graph tasks [37], RQ3 evaluates whether graph-aware and other prompting strategies improve OCL generation and identifies the errors that persist.

3 Methodology

Data. The empirical evaluation conducted in this study uses the dataset introduced in PathOCL [2]. The original dataset consists of 15 UML class models originating from multiple real-world application domains, including airport systems, healthcare, and employment management. For each UML model, the dataset provides three main artifacts: (1) the UML class model itself, represented in PlantUML format, (2) corresponding natural language specifications describing system constraints, and (3) reference gold OCL constraints associated with those specifications.

Since the gold OCL constraints were required for evaluation, two UML models were excluded from the study because they lacked corresponding gold OCL annotations. After filtering, the final dataset comprised 13 UML class models and 115 natural language specifications paired with gold OCL expressions, substantial enough for an empirical study [30, 27, 20]. All UML models were represented in textual form using the PlantUML format [32]. The



■ **Figure 2** Process to categorize LLM-generated OCL expressions as valid or invalid.

gold OCL constraints were used as ground truth references during evaluation. In particular, they were used to validate the correctness of the OCL constraints generated by the evaluated LLMs through test-instance-based verification procedures.

Evaluated Language Models. This study evaluated six large language models (LLMs), covering both closed and open-source models. The evaluated models included Claude Sonnet 4.5 [5], GPT-5 [26], Llama 4 Scout [24], Gemini 2.5 Pro [12], DeepSeek Chat V3.1 [13], and Grok 4 Fast [38]. For all experiments, the temperature was fixed at 0.1 and the maximum generation length was set to 1500 tokens, allowing sufficient output capacity for OCL generation. Additionally, to account for the stochastic nature of LLM decoding, each input specification was evaluated over 10 independent runs per model under identical prompting and decoding settings. This repeated-sampling protocol was used to reduce variance in generation quality and obtain a more stable estimate of model behaviour.

The 10 generated OCL outputs per specification were aggregated using a deterministic majority-based strategy. The outputs were normalized to remove formatting and whitespace differences, and equivalent expressions were grouped and counted. The most frequent expression, whether valid or invalid, was selected as the final prediction. In case of a frequency tie, tied expressions were considered in run order, and the first syntactically valid expression was selected. If all outputs were invalid, the most frequent invalid expression was retained and categorized as invalid during error analysis.

OCL Generation and Evaluation Procedure. The OCL generation and evaluation procedure used in this study is illustrated in Figure 2. For each UML model in the dataset, the model in the text format and its associated natural language specification are provided as inputs to the LLM. Based on these inputs, the LLM generates an OCL expression intended to apply the constraints stated within NL specifications with respect to the provided UML model. The generated OCL expressions are then evaluated using the USE (UML-based Specification Environment) [16] tool.

The evaluation process consists of two stages: syntactic validation and behavioural validation. In the first stage, the generated OCL expressions are checked for syntactic correctness using USE. This step determines whether the generated constraints conform to the grammar and structural rules of OCL. Any generated constraint that cannot be successfully parsed or executed within USE is classified as syntactically invalid.

In the second stage, the generated OCL expressions are evaluated behaviourally to determine whether they correctly capture the intended semantics of the NL specification. For each specification, two manually crafted *object* test instances conforming to the corresponding UML model are created: (1) a positive test instance satisfying the specification, and (2) a counterexample violating the specification. To improve robustness, the test instances include

not only straightforward satisfying and violating cases, but also structurally challenging scenarios designed to expose common OCL generation errors such as incorrect navigation paths, quantifier misuse, cardinality violations, and logical inconsistencies. Prior to evaluation, all test instances are validated against the gold OCL constraints to ensure they correctly represent the intended specification semantics.

Once validated, the same test instances are executed against the LLM-generated OCL expressions using USE. A generated OCL expression is considered behaviourally valid if it produces the same boolean evaluation outcomes as the corresponding gold OCL across all test instances; otherwise, it is classified as behaviourally invalid. This evaluation procedure is applied consistently across invariants, preconditions, and postconditions, enabling uniform comparison between generated and reference OCL constraints.

Experimental Setup and Graph Construction. To investigate how structural properties of UML class diagrams influence OCL generation, each UML model was first transformed into a graph-based representation. Since UML class diagrams naturally encode structural relationships between entities, representing the models as graphs enabled systematic analysis of graph-related properties relevant to OCL generation.

The UML models, represented in PlantUML format, were parsed using regular-expression-based extraction procedures. During this process, classes, attributes, operations, and associations were identified and converted into graph components. Specifically, UML classes were represented as graph nodes, while associations between classes were represented as edges. Attributes and operations associated with a class were stored as node-level properties. Additional metadata describing association types, multiplicities, navigability, and association classes were also preserved within the graph representation as edge properties.

This graph transformation enabled the extraction of structural properties required for the empirical analysis. In particular, breadth-first search (BFS) traversal was used to analyze connectivity and navigation relationships between classes. Using the generated graph representation, several structural metrics were computed for each UML model, including: 1) number of classes, 2) number of associations, 3) number of attributes, 4) number of operations, and 5) navigation depth between related classes. These graph-derived metrics were subsequently used as explanatory variables in the statistical analysis to investigate their relationship with OCL generation correctness.

Statistical Analysis. To analyze the relationship between UML structural properties and OCL generation correctness, this study employed Generalized Estimating Equations (GEE) [18] model. The dependent variable in the analysis was whether a generated OCL constraint was behaviourally correct according to the evaluation procedure described in Section 3.

A standard logistic regression model was considered, but not used because one of its key assumptions, namely the independence of observations, was violated in the experimental setting. Specifically, multiple natural language specifications were associated with the same UML class model. Since structural features such as the number of classes, associations, attributes, and graph connectivity were extracted at the UML-model level, multiple observations originating from the same UML model shared identical structural characteristics. Consequently, the generated samples were not statistically independent.

Hence, to account for this intra-model correlation, the GEE model was employed. GEE extends generalized linear models by explicitly modelling correlations among grouped observations while still estimating population-level effects. Using GEE allowed us to more reliably estimate the influence of structural UML properties on OCL generation performance while accounting for dependencies between specifications derived from the same UML model.

■ **Table 1** Effect of navigation depth on OCL correctness.

(a) Correctness by navigation depth.

Navigation depth	Clusters	Generated	Correct	Correctness
0	18	360	277	76.9%
1	54	1080	777	71.9%
2	27	486	227	46.7%
3	7	126	8	6.3%

(b) GEE regression results.

Predictor	Coef.	OR	SE	p-value
Navigation depth	-0.937	0.392	0.218	0.0031

4 Results

4.1 RQ1: Effect of UML Structural Properties on OCL Generation

RQ1.1: Effect of Navigation Depth. To investigate whether deeper UML navigation affects OCL generation, we operationalize navigation depth from the reference OCL constraints. In OCL, a navigation expression corresponds to a sequence of association traversals starting from the context class of the UML model, where each role access in a **self**-based expression represents one step in the UML graph. Navigation depth is defined as the maximum length of any such navigation chain in the gold OCL, capturing the extent of structural traversal required to express a constraint.

This metric captures the structural traversal required to express a constraint: shallow constraints involve only local attributes or directly associated classes, whereas deeper constraints require navigation across multiple related classes. Although we extract the metric from the OCL expression itself, it directly reflects the underlying UML graph structure, since each role access corresponds to traversal along an association in the class diagram.

To evaluate the association between navigation depth and OCL generation correctness, the navigation-depth feature was incorporated as a predictor variable within the GEE model. Table 1a presents the observed correctness rates across different navigation depths. The results indicate a consistent decline in correctness as navigation depth increases. Constraints with navigation depths of 0 and 1 achieved correctness rates of 76.9% and 71.9%, respectively. In contrast, correctness decreased to 46.7% for depth-2 constraints and further declined to 6.3% for depth-3 constraints. Although this pattern is consistent with longer structural traversals being more difficult, the depth-3 estimate should be interpreted as suggestive because it is based on only seven specification clusters.

The GEE analysis further indicates that this relationship is statistically significant. Navigation depth produced a negative coefficient of -0.937 , with an odds ratio of 0.392 and a p-value of 0.0031 (Table 1b). Since correctness was used as the binary outcome variable, the negative coefficient indicates that greater navigation depth is associated with a lower probability of generating a behaviourally correct OCL constraint. Interpreted in terms of odds ratios, a one-standard-deviation increase in navigation depth is associated with an approximate 60.8% reduction in the odds of generating a correct OCL expression.

The observed effect remained stable across alternative correlation assumptions. When the GEE model was refitted using an independence correlation structure, navigation depth remained negative and statistically significant, with coefficient -0.980 , odds ratio 0.375, and $p = 0.0021$. A similar pattern was also observed consistently across all evaluated LLMs, where navigation depth produced negative and statistically significant coefficients for each model individually.

Within the evaluated dataset and models, the results provide evidence of a negative association between navigation depth and LLM-based OCL generation correctness. As the required structural traversal through the UML model increases, the observed likelihood

■ **Table 2** Effect of UML structural complexity on OCL correctness.

Structural metric	Coefficient	Odds ratio	95% CI	p-value
Number of classes	-0.514	0.598	[-0.756, -0.272]	< 0.0012
Number of associations	-0.523	0.593	[-0.771, -0.274]	< 0.0024
Attributes and operations	-0.347	0.707	[-0.779, 0.085]	0.115

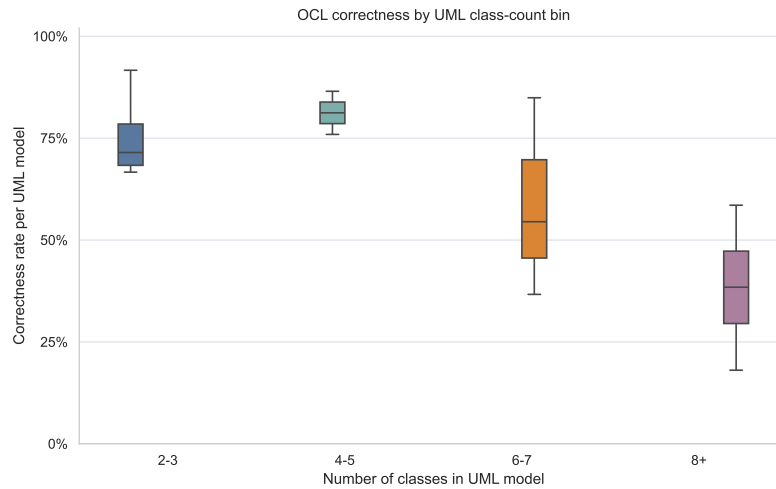
of generating a correct OCL constraint decreases. This pattern is consistent with the graph-reasoning perspective motivating this study and suggests that difficulty maintaining longer structural navigation paths may contribute to OCL generation failures. However, the particularly low depth-3 correctness rate remains suggestive rather than conclusive because that category contains only seven specification clusters.

RQ1.2: Effect of Structural Complexity. We define UML structural complexity as the size and connectivity of the UML class model that the LLM must reason over when generating an OCL constraint. In graph terms, structurally simple model contains fewer possible classes and navigation alternatives, whereas a structurally complex model contains more classes, more associations, and more model elements that may need to be selected, ignored, or navigated correctly. In this study, we operationalized structural complexity using three model-level metrics: the number of classes, the number of associations, and the total number of attributes and operations.

For each UML model, these metrics were extracted from the graph-based class diagram. Since all specifications belonging to the same UML model share the same structural-complexity values, we evaluated the relationship between each complexity metric and OCL correctness using a logistic GEE model with correctness as the binary outcome. To avoid instability caused by strong correlations among structural metrics, we fitted one GEE model per complexity metric. Each feature was standardized before fitting, so the coefficients represent the effect of a one-standard-deviation increase in the corresponding complexity measure.

Table 2 reports the results. The number of classes has a statistically significant negative association with OCL correctness ($\beta = -0.514$, $p < 0.0012$). The corresponding odds ratio is 0.598, indicating that a one-standard-deviation increase in the number of classes is associated with an approximately 40.2% decrease in the odds of generating a correct OCL expression. Similarly, the number of associations also has a statistically significant negative association with correctness ($\beta = -0.523$, $p < 0.0024$), with an odds ratio of 0.593. This corresponds to an approximately 40.7% decrease in the odds of correctness for a one-standard-deviation increase in associations.

In contrast, the number of attributes and operations does not show a statistically significant independent effect ($\beta = -0.347$, $p = 0.115$). Although the coefficient is negative, the confidence interval crosses zero. This suggests that the amount of local class-level information alone is not sufficient to explain variation in OCL correctness. One possible reason is that attributes and operations are highly correlated with other model-size measures, especially the number of classes and associations, making their independent contribution harder to isolate. Figure 3 further illustrates the relationship between UML structural complexity and OCL generation correctness using class-count bins as a proxy for model size. The observed distributions are consistent with the GEE results, showing a clear decline in correctness as the number of classes increases. UML models containing two to three classes achieve an average correctness of approximately 75.3%, whereas models containing eight or more classes achieve an average correctness of only 38.4%, representing nearly a 49% relative reduction.



■ **Figure 3** Distribution of OCL correctness across UML class-count bins.

The box plots further show that smaller UML models (2–4 classes) generally achieve higher median correctness scores, although with greater variability across specifications and prompting configurations. As model size increases, the median correctness decreases consistently across successive bins, while the distributions become increasingly concentrated near lower correctness values. In particular, the largest complexity bins (8+ classes) contain comparatively fewer high-performing observations, indicating lower observed OCL generation correctness for structurally larger models. This descriptive pattern is consistent with the negative associations identified by the regression analysis.

4.2 RQ2: Influence of Lexical and Ordering Biases

RQ2.1: Lexical Similarity Effects. This research question investigates whether lexical similarity between the natural language specification and UML model elements is associated with the performance of LLM-based OCL generation. To evaluate this hypothesis, we compare the natural language specification against UML elements that appear in the gold OCL constraint and against UML elements that do not appear in the gold OCL constraint. First, we extract the UML elements referenced by the gold OCL constraint, including the context class, referenced classes, terminal attributes, and navigated association roles. These elements form the *gold* element set, representing the UML elements required to express the constraint. All remaining classes, attributes, and roles in the UML model are treated as *non-gold* elements, representing potential distractor elements.

We then compute several lexical similarity features using a token-level Jaccard similarity metric. The gold-alignment features measure how strongly the natural language specification matches the UML elements referenced in the gold OCL constraint. Conversely, the non-gold similarity features measure similarity between the specification and UML elements not used in the gold constraint. We also compute a lexical trap score, defined as the maximum similarity between the specification and any non-gold UML element. Finally, we compute a distance-weighted trap score, which assigns higher weight to lexically similar non-gold classes that are structurally farther from the gold navigation path.

■ **Table 3** GEE results for lexical-similarity hypothesis features.

Feature	Expected direction	Coefficient	Odds ratio	p-value
Lexical trap score	Negative	0.610	1.841	0.1134
Gold alignment	Positive	0.061	1.063	0.7681
Distance-weighted trap	Negative	0.340	1.405	0.0458
Similarity to non-gold attributes	Negative	-0.514	0.598	0.0927

We hypothesize that if LLM-based OCL generation is influenced by lexical similarity, then higher similarity to gold elements is positively associated with correctness, whereas higher similarity to non-gold elements is negatively associated with correctness. Accordingly, gold-alignment features are expected to have positive coefficients, while lexical trap features are expected to have negative coefficients.

Table 3 summarizes the GEE results for the main lexical-similarity hypothesis features. The overall lexical trap score was not statistically significant ($\beta = 0.610$, $p = 0.1134$), and its coefficient was positive rather than negative. Gold alignment was also not significant ($\beta = 0.061$, $p = 0.7681$). Similarly, the distance-weighted trap score was statistically significant but in the opposite direction from the hypothesis ($\beta = 0.340$, $p = 0.0458$), indicating that it does not support the proposed lexical-trap explanation. Similarity to non-gold attributes showed a negative trend, but it was not statistically significant ($\beta = -0.514$, $p = 0.0927$). These results do not support the broad lexical-bias hypothesis. The expected pattern, where similarity to irrelevant UML elements consistently reduces correctness and similarity to gold elements improves correctness, was not observed. Across the main hypothesis tests, none of the four lexical-similarity features showed the expected statistically significant effect.

However, the broader model reveals a more nuanced result. Similarity to non-gold classes was statistically significant and negative ($\beta = -1.141$, odds ratio = 0.319, $p = 0.0006$). This suggests that class-level lexical distractors may still matter. When the natural language specification is lexically similar to classes that are not part of the gold OCL expression, the likelihood of generating a correct constraint decreases. This effect was also stable in the sensitivity analysis using an independence correlation structure ($\beta = -1.132$, odds ratio = 0.323, $p = 0.0007$).

Thus, the results provide limited rather than general support for lexical bias. We do not find evidence that LLMs rely primarily on lexical similarity across all UML element types. In particular, gold alignment, lexical trap score, and distance-weighted trap score do not behave as predicted. At the same time, the significant effect of non-gold class similarity indicates that lexical overlap with irrelevant class names can still interfere with OCL generation. This suggests that LLM failures cannot be explained by lexical similarity alone, and instead, lexical cues may interact with structural reasoning, especially when the model must choose among competing candidate classes in the UML graph.

Therefore, for RQ2.1, we conclude that lexical similarity is not a sufficient explanation for OCL generation errors. The results reject the broad lexical-trap hypothesis, but they leave room for a narrower class-level lexical distraction effect. This finding is consistent with the results from several graph reasoning paper [37, 42] where LLMs appear to perform some reasoning over the UML model structure, but their generation can still be affected by misleading textual cues in specific cases.

■ **Table 4** Distribution of OCL generation errors across prompting strategies and evaluated LLMs.

Model	Prompting Strategy	Correct	Hal.	Nav.	Log	Inc.	Inv.
Claude Sonnet 4.5	Classes-first	82	2	10	4	8	9
	Associations-first	80	–	12	3	7	13
	Breadth-first search (BFS)	81	1	14	3	6	10
	Depth-first search (DFS)	80	–	14	3	4	14
	Leaf-first	79	1	13	4	5	13
GPT-5	Classes-first	80	5	11	8	6	5
	Associations-first	80	5	11	2	3	14
	Breadth-first search (BFS)	77	1	11	2	6	18
	Depth-first search (DFS)	77	6	14	2	4	12
	Leaf-first	82	6	11	4	4	8
Llama 4 Scout	Classes-first	71	3	20	9	4	8
	Associations-first	57	6	18	6	4	24
	Breadth-first search (BFS)	65	1	17	2	3	27
	Depth-first search (DFS)	53	3	19	5	7	28
	Leaf-first	54	3	20	5	7	26
Gemini 2.5 Pro	Classes-first	79	3	12	8	7	6
	Associations-first	78	3	10	3	9	12
	Breadth-first search (BFS)	73	5	12	2	7	16
	Depth-first search (DFS)	74	3	12	3	11	12
	Leaf-first	85	2	11	1	4	12
DeepSeek v3.1	Classes-first	30	2	12	3	4	64
	Associations-first	85	1	12	6	2	9
	Breadth-first search (BFS)	81	2	12	5	2	13
	Depth-first search (DFS)	79	1	7	8	3	17
	Leaf-first	75	1	18	4	4	13
Grok-4-Fast	Classes-first	71	8	8	5	10	13
	Associations-first	82	3	9	4	4	13
	Breadth-first search (BFS)	75	6	13	4	3	14
	Depth-first search (DFS)	78	1	14	5	1	16
	Leaf-first	72	1	16	4	11	11

Hal: Hallucination, Nav: Navigation, Log: Logic, Inc: Incorrect Ops, Inv: Invalid Ops

RQ2.2: UML Ordering Effects. To investigate whether the textual ordering of UML elements influences OCL generation, we evaluated five strategies for linearizing UML class diagrams before providing them to the LLM. These include *classes-first*, *associations-first*, *breadth-first search (BFS)*, *depth-first search (DFS)* inspired by graph-reasoning studies [15] and a proposed *leaf-first* traversal. The classes-first and associations-first strategies grouped UML elements by type, whereas BFS, DFS, and leaf-first generated traversal-based ordering over the UML graph structure. The proposed leaf-first strategy begins from terminal classes and progressively moves toward more central classes, motivated by the observation that many OCL constraints ultimately reference terminal attributes or entities located near leaf regions of the UML graph. For BFS and DFS, traversal was performed over the dual graph representation following [15] to ensure that all UML associations were included in the final prompt. These transformations modified only the textual presentation order of the UML model and did not alter its underlying semantics or topology.

Table 4 reports the effect of different textual ordering of the UML class diagram on OCL generation correctness and error distributions across all evaluated LLMs. The results show that LLM performance is sensitive to the textual ordering of UML elements, even though the underlying UML graph remains unchanged. This finding is consistent with prior graph-reasoning studies [15], suggesting that LLMs are influenced not only by structural content itself, but also by how that structure is linearized within the prompt.

Across most evaluated models, no single ordering strategy consistently dominated all others. Nevertheless, several important trends emerge. First, traversal-based orderings such as BFS and DFS often increased the number of navigation and invalid-generation errors relative to simpler serialization strategies. For example, GPT-5 produced 11 navigation errors under the classes-first representation, compared to 14 under DFS ordering. Similarly, Claude Sonnet 4.5 exhibited an increase in invalid outputs from 9 under classes-first ordering

to 14 under DFS ordering. These results suggest that traversal-based serializations may introduce longer or less locally coherent prompt structures, making it more difficult for the model to maintain consistent navigation paths during OCL generation.

Second, the results indicate that textual order can substantially affect syntactic validity. This effect is particularly visible for DeepSeek v3.1. Under the conventional classes-first representation, DeepSeek achieved only 30 correct generations out of 115 evaluated specifications, while producing 64 invalid OCL expressions. In contrast, under associations-first ordering, correctness increased dramatically to 85 correct generations, with invalid outputs decreasing to only 9. Similar improvements were observed for BFS and DFS orderings. Manual inspection revealed that many invalid generations under the classes-first configuration omitted required OCL context declarations entirely. For example:

```
Project.allInstances()->forAll(p | p.department.locations
->includes(p.location))
```

Although structurally plausible, such expressions are not valid OCL invariants because they lack an explicit `context` declaration. One possible explanation is that the classes-first ordering encourages DeepSeek to generate constraints in a more query-like or declarative style rather than following the expected invariant structure. In contrast, traversal-based and associations-first representations may provide stronger relational cues that better align with the model’s learned patterns for structured constraint generation. However, because this behaviour was not consistently observed across the other evaluated LLMs, the exact cause remains unclear and requires further investigation.

The proposed leaf-first ordering produced mixed but noteworthy results. For GPT-5 and Gemini 2.5 Pro, leaf-first ordering achieved the highest correctness among all evaluated strategies, reaching 82 and 85 correct generations respectively. This partially supports the intuition underlying the leaf-first design: presenting structurally terminal classes earlier in the prompt may help the model more easily identify terminal attributes and navigation targets frequently referenced in OCL constraints. However, this behaviour was not universal across models. For Llama 4 Scout and Grok-4-Fast, leaf-first ordering did not improve correctness and in some cases increased navigation-related errors. These results suggest that while leaf-first ordering may help certain models prioritize relevant structural regions of the UML graph, its effectiveness likely depends on how individual LLMs internally process long structured contexts.

More broadly, the results indicate that OCL generation is sensitive to prompt serialization structure even when the semantic content of the UML model remains identical. The observed differences across ordering strategies support the hypothesis that LLMs do not reason over UML graphs in a fully structure-invariant manner. Instead, the textual presentation of structural information itself influences how effectively the model can maintain consistent graph navigation and constraint construction during generation.

4.3 RQ3: Prompting Strategies and Error Analysis

RQ3.1: Effectiveness of Prompting Strategies. To answer RQ3.1, we evaluated four prompting strategies: zero-shot, Chain-of-Thought (CoT), few-shot, and graph-based prompting. The first three follow standard prompting paradigms for code and formal-language generation, while the graph-based variant explicitly instructs the model to reconstruct the UML diagram as a graph before generating OCL, with the aim of improving structural reasoning and navigation consistency, and reduce graph-related reasoning failures during OCL generation.

■ **Table 5** Distribution of OCL generation errors across prompting strategies and evaluated LLMs.

Model	Prompting Strategy	Correct	Hal.	Nav.	Log	Inc.	Inv.
Claude Sonnet 4.5	Zero-shot	82	2	10	4	8	9
	Few-shot	73	4	12	7	6	13
	CoT	81	5	9	8	4	8
	Graph-based	83	–	9	5	6	12
GPT-5	Zero-shot	80	5	11	8	6	5
	Few-shot	82	5	5	8	4	7
	CoT	85	5	5	10	4	6
	Graph-based	91	1	4	4	5	10
Llama 4 Scout	Zero-shot	71	3	20	9	4	8
	Few-shot	67	1	18	8	3	16
	CoT	69	6	16	11	5	8
	Graph-based	60	15	11	5	8	16
Gemini 2.5 Pro	Zero-shot	79	3	12	8	7	6
	Few-shot	78	7	9	7	7	7
	CoT	81	4	13	8	5	4
	Graph-based	86	4	15	2	4	4
DeepSeek v3.1	Zero-shot	30	2	12	3	4	64
	Few-shot	67	11	10	9	5	13
	CoT	41	9	10	8	3	40
	Graph-based	80	8	10	3	2	12
Grok-4-Fast	Zero-shot	71	8	8	5	10	13
	Few-shot	76	5	11	5	8	10
	CoT	73	8	9	9	11	5
	Graph-based	86	4	12	3	3	7

Hal: Hallucination, Nav: Navigation, Log: Logic, Inc: Incorrect Ops, Inv: Invalid Ops

For evaluation, a generated OCL constraint was considered *correct* only if it was both syntactically valid and behaviourally consistent with the corresponding gold OCL according to the test-instance-based evaluation procedure described earlier. Incorrect generations were further categorized into five major error types: hallucinated UML elements, navigation errors, logical inconsistencies, incorrect OCL operation usage, and syntactic invalidity. We use the class-first based textual structure when appending the UML class diagram to the prompts.

Table 5 summarizes the correctness and error distributions across prompting strategies and evaluated LLMs. Graph-based prompting consistently achieved the highest or near-highest correctness across most evaluated models. GPT-5 improved from 80 correct constraints under zero-shot prompting to 91 under graph-based prompting, representing the highest overall performance. Similarly, Grok-4-Fast improved from 71 to 86 correct constraints, while DeepSeek-v3.1 exhibited the largest relative improvement, increasing from only 30 correct constraints under zero-shot prompting to 80 correct constraints using graph-based prompting.

The results further show that graph-based prompting substantially reduced several structural reasoning failures. In particular, GPT-5 reduced navigation-related errors from 11 under zero-shot prompting to 4 under graph-based prompting, while hallucination errors decreased from 5 to 1. Similar reductions in navigation and hallucination errors were observed for Claude Sonnet 4.5 and Grok-4-Fast. These findings suggest that explicitly reconstructing the UML model as a graph may help LLMs better maintain structural consistency during OCL generation.

Compared to graph-based prompting, Chain-of-Thought and few-shot prompting produced mixed results. Although CoT prompting occasionally improved correctness, the gains were generally smaller and less consistent across models. Few-shot prompting similarly showed moderate improvements for some models but occasionally introduced additional invalid or hallucinated outputs. For example, Claude Sonnet 4.5 decreased from 82 correct constraints under zero-shot prompting to 73 under few-shot prompting, while invalid outputs increased from 9 to 13.

Despite the overall improvements, graph-based prompting did not uniformly reduce all error categories. Some models continued to exhibit navigation failures and syntactic invalidity even after explicit graph construction. Furthermore, for Llama 4 Scout, graph-based prompting produced lower correctness than the simpler prompting strategies, suggesting that additional structural reasoning steps may increase reasoning instability for certain models.

In answer to RQ3.1, prompting strategies that explicitly encourage graph construction and structural processing achieved higher observed correctness for several evaluated models, including GPT-5, Gemini 2.5 Pro, DeepSeek v3.1, and Grok-4-Fast. However, graph-based prompting did not improve performance for every model, and its effectiveness varied substantially across LLM families. Explicit structural decomposition therefore provides only a partial and model-dependent improvement and is insufficient by itself to resolve OCL generation failures.

RQ3.2: Analysis of OCL Generation Errors. To better understand the limitations of LLM-based OCL generation, we manually analyzed incorrect outputs and categorized the observed failures into five recurring error types: navigation errors, incorrect use of OCL operations, hallucinated model content, logical errors, and invalid or non-OCL constructs. These categories were derived through iterative inspection of generated constraints across all evaluated prompting strategies and models. The results indicate that many failures are closely related to structural reasoning over UML models, rather than purely syntactic generation mistakes. We explain the errors using the class diagram illustrated in Figure 1.

Navigation Errors. Navigation errors occur when a generated OCL expression attempts to access model elements through invalid or nonexistent association paths. Although such expressions are often syntactically valid, they violate the structural relationships defined in the UML model and therefore fail during semantic validation or behavioural evaluation. *Example:* Given the natural language requirement, “The start date of an employee as manager of a department must be greater than his/her hire date,” one model generated the following OCL constraint:

```
context Manages inv: self.startDate > self.employee.hireDate
```

However, the class `Manages` contains no association end named `employee`; the correct navigation should use `manager`. Consequently, the navigation path `self.employee` is invalid. This example illustrates how LLMs may incorrectly traverse the UML structure despite producing syntactically plausible OCL.

Incorrect Use of OCL Operations. This category includes errors in which OCL collection or iterator operations are applied incorrectly. Common examples include invoking collection operators on non-collection values, omitting required iterator expressions, or confusing attribute navigation (`.`) with collection navigation (`->`). *Example:* Given the natural language requirement, “The SSN of employees is an identifier (or key),” the following OCL expression was generated:

```
context Employee inv ssnUnique:
Employee.allInstances().SSN->isUnique()
```

The `isUnique()` operation requires an iterator expression specifying the uniqueness criterion for each element in the collection. Since no iterator body is provided, the expression is invalid. This example demonstrates that LLMs frequently struggle with the semantic constraints imposed by OCL collection operators.

Hallucinated Model Content. LLMs occasionally generate associations, attributes, or operations that do not exist in the UML model. These hallucinated elements are often semantically plausible but unsupported by the provided diagram, indicating reliance on learned language priors rather than faithful structural reasoning. *Example:* Given the requirement, “The location of a project must be in one of the locations of its department,” the following OCL constraint was produced:

```
context Project inv locationConstraint:
self.controllingDepartment.locations->includes(self.location)
```

In this example, the association `controllingDepartment` does not exist in the UML model and is fabricated by the LLM. The generated expression therefore references a non-existent navigation path.

Logic Errors. Logic errors arise when a generated OCL expression is syntactically valid and structurally consistent with the UML model, but does not correctly encode the intended semantics of the natural language specification. These errors are typically identified during behavioural evaluation using test instances. *Example:* Given the natural language requirement, “The manager of a department must be an employee of the department,” one generated constraint was:

```
context Department inv ManagerIsEmployee: self.manager.department =
self
```

Although the expression is syntactically valid, it does not correctly enforce the intended constraint. Specifically, the generated OCL only checks whether the manager references the current department, rather than verifying that the manager belongs to the department’s employee collection. Consequently, invalid instances may still satisfy the constraint.

Invalid or Non-OCL Constructs. Some generated expressions contain operations, symbols, or functions that are not part of the OCL language. These errors typically result in parsing failures and indicate that the model is transferring assumptions from general-purpose programming languages into OCL generation. *Example:* Given the requirement, “The location of a project must be in one of the locations of its department,” one model produced the following constraint:

```
context Project inv: self.department.locations.split(',')
->exists(loc | loc.trim() = self.location)
```

In this example, the operation `trim()` is not defined in standard OCL. The generated expression therefore contains non-OCL constructs and cannot be executed by the USE validator.

These observed error categories are consistent with structural demands contributing to some OCL generation failures. In particular, navigation errors and hallucinated associations show that the evaluated LLMs do not always preserve the relationships defined by the underlying UML graph, especially when constraints require multi-step navigation across related classes. While prompting strategies such as graph-based prompting reduce some of these failures, the remaining errors suggest that reliable structural navigation continues to be an important challenge within the evaluated setting.

5 Discussion

These results extend prior graph-reasoning findings to a formal software-engineering task: increasing path length and changing graph serialization are reflected not only in lower task accuracy, but also in domain-specific failures such as invalid OCL navigation and hallucinated

UML elements. In particular, the observed relationships involving navigation depth, UML complexity, and textual ordering are consistent with structural task demands contributing to generation difficulty. At the same time, the comparatively weaker and less consistent lexical-similarity effects suggest that the observed failures cannot be explained purely through shallow textual matching or token-level associations. These findings motivate a broader discussion of how structural demands, graph representation, and prompt organization may shape formal specification generation with LLMs.

More broadly, the structural metrics evaluated in this paper represent only a subset of the possible UML properties that may influence OCL generation. While this study focused on navigation depth, class count, association count, and overall model size, many additional graph characteristics may also affect LLM reasoning performance. Examples include node degree distributions, multiplicity constraints, inheritance depth, association types, graph density, centrality measures, and the presence of highly connected hub classes. Similarly, the textual ordering strategies explored in this work represent only a small subset of the possible linearizations of UML graph structures. Since the number of valid serialization strategies grows combinatorially with graph size, exhaustively evaluating all possible orderings is infeasible. Instead, the strategies evaluated here were selected based on prior graph-reasoning literature [15, 17, 42] and to provide representative examples of structurally different prompt organizations.

Consequently, this work should be viewed as an initial empirical investigation into the relationship between UML graph structure and LLM-based OCL generation. Within the evaluated dataset and models, the results identify associations between OCL correctness and structural properties, textual order, and graph-oriented prompting; they do not establish these factors as the sole or primary causes of failure. We therefore view this study as a foundation for future work that can examine these relationships using larger datasets and stronger controls, while providing directions for graph-sensitive prompting, UML structural analysis, and structure-aware evaluation of LLM-based formal specification generation.

6 Threats to Validity

Construct Validity. OCL correctness was assessed using behavioural agreement with gold constraints on manually constructed test instances. While practical, behavioural equivalence on finite cases does not guarantee full semantic equivalence. Correct formulations may be misclassified and vice versa. To mitigate this, test instances included both satisfying and violating cases, including edge scenarios. Additionally, navigation depth was used as a proxy for structural complexity, but it does not capture all aspects of reasoning difficulty (e.g., branching or predicate complexity). Lexical similarity measures were also limited to surface-level string comparisons.

Internal Validity. Observed relationships between UML properties and correctness do not imply causality, as other factors (e.g., logical complexity) may contribute. Model outputs may also vary due to prompting strategies and stochastic decoding, although consistent settings and repeated sampling were used to reduce this effect.

External Validity. The study uses the PathOCL dataset (13 models, 115 specifications), which may not represent the full diversity of industrial UML/OCL usage. The focus on class-diagram invariants limits generalizability to more complex or large-scale settings. The small number of models reflects the scarcity of suitable public datasets with aligned UML,

natural language, and OCL. Synthetic datasets were avoided due to potential realism and bias concerns, though this limits scale. Future work should validate findings on larger and more diverse datasets.

Conclusion Validity. The use of GEE accounts for correlated observations, but the small number of UML models may limit statistical power. Results were consistent across sensitivity analyses.

7 Related Work

Automatically generating OCL constraints from natural language specifications has been studied widely within model-driven engineering research [8]. Early approaches primarily relied on rule-based transformations, template matching, and syntactic parsing techniques to translate restricted forms of natural language into OCL expressions [6, 35]. These approaches typically required carefully structured input specifications and were limited in their ability to generalize across domains and linguistic variations. More recent work has explored the use of Large Language Models (LLMs) for OCL generation [33, 21, 39]. Abukhalaf et al. [1] investigated prompt engineering strategies for Codex-based OCL generation and demonstrated that prompting design significantly influences generation quality. PathOCL [2] further introduced path-based prompt augmentation strategies intended to improve structural navigation during OCL generation. Other recent work has explored dataset construction and fine-tuning approaches for improving LLM-based OCL generation [28, 23, 21]. These studies primarily focus on improving generation accuracy through prompting or training strategies.

Furthermore, recent research has shown that LLMs struggle with graph reasoning tasks when graph structure is represented in natural language form [36, 41, 42, 17, 40]. The NLGraph benchmark [37] demonstrated that LLM performance degrades substantially on tasks such as shortest-path reasoning, connectivity analysis, and Hamiltonian path problems as graph complexity increases. Subsequent work further showed that LLM reasoning is highly sensitive to graph serialization and descriptive ordering [15].

Other studies have proposed methods for improving graph reasoning capabilities [11]. GraphOtter [22] explored graph-evolving reasoning strategies for complex reasoning tasks, while Peng et al. [29] investigated reinforcement-learning-based approaches for improving graph reasoning generalization. These studies collectively suggest that current LLMs often struggle to maintain consistent structural reasoning over multi-hop graph relationships. Additionally, representing UML class diagrams as graphs has been widely studied in model-driven engineering and formal verification research. Prior work has explored transformations from UML models into graph transformation systems for analysis and verification purposes [19, 43, 4]. These graph-based representations enable reasoning about structural relationships, navigability, and model consistency.

OCL validation itself has also been extensively studied [34]. USE [16] provides executable validation support for UML and OCL models, while other work has explored constraint solving and automated test-data generation for OCL specifications [3, 14, 10]. In contrast to prior work, we investigate factors associated with LLM failures in OCL generation by relating them to the structural demands of navigating UML graphs. We show that common errors, such as invalid paths and hallucinated associations, resemble error patterns reported in graph-reasoning tasks, but within a formal software engineering context. Together with the statistical results, these observations suggest that UML structural properties are relevant to LLM performance without establishing them as the exclusive cause of failure.

8 Conclusion and Future Work

This paper examined why Large Language Models (LLMs) struggle to generate correct OCL constraints from natural language and UML class diagrams. We approached this problem from a graph-reasoning perspective, analyzing how UML structure influences generation correctness.

Using the PathOCL dataset, we evaluated multiple LLMs and analyzed the relationships between correctness and navigation depth, structural complexity, lexical similarity, and prompting strategies. Within the evaluated dataset and models, correctness declined as navigation depth and UML complexity increased, and constraints involving multi-hop traversal achieved lower observed correctness. These findings are consistent with patterns reported in prior graph-reasoning studies.

The tested lexical factors provided limited explanatory power, indicating that the observed failures cannot be explained by superficial text matching alone. While graph-aware prompting produced modest improvements for several models, common errors persisted, including invalid navigation, hallucinated associations, and logical inconsistencies. These results are consistent with OCL generation involving both linguistic translation and structural navigation demands.

Future work should validate these findings on larger and more diverse UML/OCL datasets using robustness methods such as cluster bootstrap, permutation tests, or mixed-effects models. Multivariable or matched analyses could better isolate the effects of navigation depth and model size from factors such as NL/OCL length, predicate complexity, quantifiers, and collection operations. The semantic oracle could also be strengthened through larger generated test suites, constraint solving, and mutation analysis. Further studies should examine alternative similarity measures, statistically evaluate prompting improvements and their interaction with UML serialization, and explore validity-aware generation, constrained decoding, and iterative repair using parser, type-checker, and behavioural feedback alongside graph-aware and neuro-symbolic approaches.

Data Availability Statement

To support reproducibility and comply with the conference open science policy, the datasets, prompts, and evaluation artifacts used in this study have been made publicly available at DOI: <https://doi.org/10.5281/zenodo.20279636>.

References

- 1 Seif Abukhalaf, Mohammad Hamdaqa, and Foutse Khomh. On codex prompt engineering for ocl generation: an empirical study. In *2023 IEEE/ACM 20th International Conference on Mining Software Repositories (MSR)*, pages 148–157. IEEE, 2023. doi:10.1109/MSR59073.2023.00033.
- 2 Seif Abukhalaf, Mohammad Hamdaqa, and Foutse Khomh. Pathocl: path-based prompt augmentation for ocl generation with gpt-4. In *Proceedings of the 2024 IEEE/ACM First International Conference on AI Foundation Models and Software Engineering*, pages 108–118, 2024. doi:10.1145/3650105.3652290.
- 3 Shaikat Ali, Muhammad Zohaib Iqbal, Andrea Arcuri, and Lionel C Briand. Generating test data from ocl constraints with search techniques. *IEEE Transactions on Software Engineering*, 39(10):1376–1402, 2013. doi:10.1109/TSE.2013.17.
- 4 Omar Raad Alsammak and Ashraf Abdulmunim Abdulmajeed. Effective analysis and classification of uml class diagrams: Literature review. In *AIP Conference Proceedings*, volume 3393, page 060011. AIP Publishing LLC, 2026.
- 5 Anthropic. Claude sonnet 4.5. <https://www.anthropic.com/news/claude-sonnet-4-5>, 2025. Accessed: Nov. 13, 2025.

- 6 Imran Sarwar Bajwa, Behzad Bordbar, and Mark G Lee. Ocl constraints generation from natural language specification. In *2010 14th IEEE International Enterprise Distributed Object Computing Conference*, pages 204–213. IEEE, 2010. doi:10.1109/EDOC.2010.33.
- 7 Jordi Cabot, Robert Clarisó, and Daniel Riera. On the verification of uml/ocl class diagrams using constraint programming. *Journal of Systems and Software*, 93:1–23, 2014. doi:10.1016/J.JSS.2014.03.023.
- 8 Jordi Cabot, David Delgado, and Lola Burgueño. Combining ocl and natural language: a call for a community effort. In *Proceedings of the 25th International Conference on Model Driven Engineering Languages and Systems: Companion Proceedings*, pages 908–912, 2022. doi:10.1145/3550356.3561542.
- 9 Jordi Cabot and Martin Gogolla. Object constraint language (ocl): a definitive guide. In *International school on formal methods for the design of computer, communication and software systems*, pages 58–90. Springer, 2012. doi:10.1007/978-3-642-30982-3_3.
- 10 Jordi Cabot and Ernest Teniente. Constraint support in mda tools: A survey. In *European Conference on Model Driven Architecture-Foundations and Applications*, pages 256–267. Springer, 2006. doi:10.1007/11787044_20.
- 11 Ziwei Chai, Tianjie Zhang, Liang Wu, Kaiqiang Han, Xiaohai Hu, Xuanwen Huang, and Yang Yang. Graphllm: Boosting graph reasoning ability of large language model. *IEEE Transactions on Big Data*, 2025.
- 12 Google DeepMind. Gemini 2.5 pro. <https://deepmind.google/en/models/gemini/pro>, 2025. Accessed: Nov. 13, 2025.
- 13 DeepSeek-AI. Deepseek-v3 technical report. *arXiv preprint arXiv:2412.19437*, 2024. doi:10.48550/arXiv.2412.19437.
- 14 Enrico Franconi, Alessandro Mosca, Xavier Oriol, Guillem Rull, and Ernest Teniente. Ocl fo: first-order expressive ocl constraints for efficient integrity checking. *Software & Systems Modeling*, 18(4):2655–2678, 2019. doi:10.1007/S10270-018-0688-Z.
- 15 Yuyao Ge, Shenghua Liu, Baolong Bi, Yiwei Wang, Lingrui Mei, Wenjie Feng, Lizhe Chen, and Xueqi Cheng. Can graph descriptive order affect solving graph problems with llms? In *Proceedings of the 63rd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 6404–6420, 2025. doi:10.18653/V1/2025.ACL-LONG.321.
- 16 Martin Gogolla, Fabian Büttner, and Mark Richters. Use: A uml-based specification environment for validating uml and ocl. *Science of Computer Programming*, 69(1-3):27–34, 2007. doi:10.1016/J.SCICP.2007.01.013.
- 17 Haoyu Han, Yaochen Xie, Hui Liu, Xianfeng Tang, Sreyashi Nag, William Headden, Yang Li, Chen Luo, Shuiwang Ji, Qi He, et al. Reasoning with graphs: Structuring implicit knowledge to enhance llms reasoning. In *Findings of the Association for Computational Linguistics: ACL 2025*, pages 25698–25714, 2025.
- 18 James W Hardin and Joseph M Hilbe. *Generalized estimating equations*. chapman and hall/CRC, 2002.
- 19 Karsten Hölscher, Paul Ziemann, and Martin Gogolla. On translating uml models into graph transformation systems. *Journal of Visual Languages & Computing*, 17(1):78–105, 2006. doi:10.1016/J.JVLC.2005.11.001.
- 20 Zhongzhan Huang, Guoming Ling, Shanshan Zhong, Hefeng Wu, and Liang Lin. Minilongbench: the low-cost long context understanding benchmark for large language models. In *Proceedings of the 63rd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 11442–11460, 2025. doi:10.18653/V1/2025.ACL-LONG.560.
- 21 Kevin Chenhao Li, Vahid Zolfaghari, Nenad Petrovic, Fengjunjie Pan, and Alois Knoll. Optimizing retrieval augmented generation for object constraint language. *arXiv preprint arXiv:2505.13129*, 2025. doi:10.48550/arXiv.2505.13129.
- 22 Qianlong Li, Chen Huang, Shuai Li, Yuanxin Xiang, Deng Xiong, and Wenqiang Lei. Graphotter: Evolving llm-based graph reasoning for complex table question answering. In *Proceedings of the 31st International Conference on Computational Linguistics*, pages 5486–5506, 2025. URL: <https://aclanthology.org/2025.coling-main.368/>.

- 23 Christoph Mayr-Dorn, Anmol Bilal, Cosmina Ratiu, and Alexander Egyed. Generating quality assurance constraints from natural language with llms. *Journal of Software: Evolution and Process*, 37(11):e70062, 2025. doi:10.1002/SMR.70062.
- 24 Meta AI. Introducing LLaMA 4: Advancing multimodal intelligence. <https://ai.meta.com/blog/llama-4-multimodal-intelligence/>, 2024. Accessed: 2025-12-29.
- 25 Object Management Group. Object constraint language (ocl), version 2.4. Technical report, OMG, 2014. URL: <https://www.omg.org/spec/OCL/2.4/>.
- 26 OpenAI. Gpt-5. <https://chat.openai.com>, 2025. Accessed: Nov. 13, 2025.
- 27 Lorenzo Pacchiardi, Lucy G Cheke, and José Hernández-Orallo. 100 instances is all you need: predicting the success of a new llm on unseen data by testing on a few instances. *arXiv preprint arXiv:2409.03563*, 2024. doi:10.48550/arXiv.2409.03563.
- 28 Fengjunjie Pan, Vahid Zolfaghari, Long Wen, Nenad Petrovic, Jianjie Lin, and Alois Knoll. Generative ai for ocl constraint generation: Dataset collection and llm fine-tuning. In *2024 IEEE International Symposium on Systems Engineering (ISSE)*, pages 1–8. IEEE, 2024.
- 29 Miao Peng, Nuo Chen, Zongrui Suo, and Jia Li. Rewarding graph reasoning process makes llms more generalized reasoners. In *Proceedings of the 31st ACM SIGKDD Conference on Knowledge Discovery and Data Mining V. 2*, pages 2257–2268, 2025. doi:10.1145/3711896.3737109.
- 30 Felipe Maia Polo, Lucas Weber, Leshem Choshen, Yuekai Sun, Gongjun Xu, and Mikhail Yurochkin. tinybenchmarks: evaluating llms with fewer examples. In *Proceedings of the 41st International Conference on Machine Learning*, pages 34303–34326, 2024. URL: <https://proceedings.mlr.press/v235/maia-polo24a.html>.
- 31 Revanth Rameshkumar, Jimson Huang, Yunxin Sun, Fei Xia, and Abulhair Saparov. Reasoning models reason well, until they don't. In *Proceedings of the 14th International Joint Conference on Natural Language Processing and the 4th Conference of the Asia-Pacific Chapter of the Association for Computational Linguistics*, pages 936–956, 2025. doi:10.18653/V1/2025.IJCNLP-LONG.52.
- 32 Arnaud Roques. Plantuml. <https://plantuml.com/>, 2026. Accessed: 2026-05-09.
- 33 Hanan Abdulwahab Siala and Kevin Lano. Using llms to extract ocl specifications from java and python programs: an empirical study. *OCL '25, STAF*, 2025.
- 34 Mathias Soeken, Robert Wille, Mirco Kuhlmann, Martin Gogolla, and Rolf Drechsler. Verifying uml/ocl models using boolean satisfiability. In *2010 Design, Automation & Test in Europe Conference & Exhibition (DATE 2010)*, pages 1341–1344. IEEE, 2010. doi:10.1109/DATE.2010.5457017.
- 35 Li Tan, Zongyuan Yang, and Jinkui Xie. Ocl constraints automatic generation for uml class diagram. In *2010 IEEE International Conference on Software Engineering and Service Sciences*, pages 392–395. IEEE, 2010.
- 36 Anton Tsitsulin, Bryan Perozzi, Bahare Fatemi, and Jonathan J Halcrow. Graph reasoning with llms (greal). In *Proceedings of the 30th ACM SIGKDD conference on knowledge discovery and data mining*, pages 6424–6425, 2024. doi:10.1145/3637528.3671448.
- 37 Heng Wang, Shangbin Feng, Tianxing He, Zhaoxuan Tan, Xiaochuang Han, and Yulia Tsvetkov. Can language models solve graph problems in natural language? *Advances in Neural Information Processing Systems*, 36:30840–30861, 2023.
- 38 xAI. Grok-4-fast. <https://docs.x.ai/docs/models/grok-4-fast-reasoning>, 2025. Accessed: Nov. 13, 2025.
- 39 Yilong Yang, Yibo Liu, Tianshu Bao, Weiru Wang, Nan Niu, and Yongfeng Yin. Deepocl: A deep neural network for object constraint language generation from unrestricted nature language. *CAAI Transactions on Intelligence Technology*, 9(1):250–263, 2024. doi:10.1049/CIT2.12207.
- 40 Zike Yuan, Ming Liu, Hui Wang, and Bing Qin. Ma-gts: A multi-agent framework for solving complex graph problems in real-world applications. In *Proceedings of the 2025 Conference on Empirical Methods in Natural Language Processing*, pages 19297–19315, 2025.

- 41 Qifan Zhang, Nuo Chen, Zehua Li, Miao Peng, Jing Tang, and Jia Li. Improving LLMs' generalized reasoning abilities by graph problems. In *Second Conference on Language Modeling*, 2025. URL: <https://openreview.net/forum?id=6vMRcaYbU7>.
- 42 Yizhuo Zhang, Heng Wang, Shangbin Feng, Zhaoxuan Tan, Xiaochuang Han, Tianxing He, and Yulia Tsvetkov. Can llm graph reasoning generalize beyond pattern memorization? In *Findings of the Association for Computational Linguistics: EMNLP 2024*, pages 2289–2305, 2024. doi:10.18653/V1/2024.FINDINGS-EMNLP.127.
- 43 Paul Ziemann, Karsten Hölscher, and Martin Gogolla. From uml models to graph transformation systems. *Electronic Notes in Theoretical Computer Science*, 127(4):17–33, 2005. doi:10.1016/J.ENTCS.2004.10.025.