Vrije Universiteit Amsterdam

Faculty of Science

Study Programme: Artificial Intelligence

Tim Jonatan But

Evaluating & Fine-Tuning LLMs for Natural Language Querying of Code Graphs

Bachelor's Thesis

Supervisor: Dr. Mauricio Verano Merino

Amsterdam, 2025

Evaluating & Fine-Tuning LLMs for Natural Language Querying of Code Graphs

But T. J.
Student, BSc Artificial Intelligence
Vrije Universiteit Amsterdam
Amsterdam, the Netherlands
t.j.but@student.vu.nl

***Abstract*—Translating natural language queries into graph database queries (Text-to-Cypher) has significant practical potential for simplifying complex software dependency analyses. However, existing large language models (LLMs), fine-tuned on general datasets, frequently struggle with specialized schemas like those used by TNO's Renaissance static analysis tool. This study evaluates how general-purpose fine-tuned LLMs perform on such specialized schemas and investigates whether targeted fine-tuning on a small, domain-specific dataset can yield meaningful performance improvements. Our analysis reveals that general fine-tuning often negatively impacts model performance due to domain shift, particularly affecting syntactic correctness. Targeted fine-tuning on a limited dataset (347 query pairs) provided minimal and inconsistent benefits, underscoring that the inherent capability of the foundational model is a stronger predictor of success. Notably, open-source models such as Gemma-3-12B-it demonstrated competitive performance with state-of-the-art proprietary models without additional fine-tuning. These findings suggest that selecting a highly capable base model currently offers greater practical benefits than attempting small-scale fine-tuning: an important consideration for practitioners developing natural-language interfaces for code graph querying.**

***Index Terms*—Large Language Models, Text-to-Cypher, Fine-Tuning, Text-to-Query, Parameter-Efficient Fine-Tuning (PEFT), Code Graph, Dependency Analysis.**

I. Introduction

Software projects spanning millions of lines of code and thousands of components are no new phenomenon, yet developers still face the everyday question of "what else breaks when I touch this file?". Empirical studies show that when teams overlook inter-module dependencies, integration test failures rise and resolution time lengthens [1], [2]. Cataldo et al. found that features containing architectural interactions between features are markedly more prone to integration failures than those whose dependencies are explicitly recognized and coordinated [1]. Additionally, poor alignment between developer coordination and the actual dependency structure has shown to lengthen modification-request resolution time by nearly a third [2]. Consequently, tooling that exposes and queries dependency structure has the potential to alleviate these issues, and enable the safe and rapid evolution of large systems [3].

This work was supported and funded by the Netherlands Organisation for Applied Scientific Research (TNO).

One powerful solution is to represent software dependencies as a code graph, where software entities (e.g., classes, functions, files) become nodes and their relationships (e.g., calls, imports, inheritance) become typed edges. These graphs are often stored in property-graph databases, such as Neo4j, which are designed for efficient traversal using a domain-specific language (DSL) like Cypher [4]. The utility of this approach is well-documented; the expressive power of code graphs has been leveraged to detect 18 previously unknown vulnerabilities in the Linux kernel [5], while their scalability has been proven in performing architectural conformance checks on multimillion-line Java projects [6]. Moreover, for traversal-heavy tasks, graph-based queries can outperform their relational SQL counterparts by up to an order of magnitude [7].

A concrete example of this approach is TNO's Renaissance: a static analysis tool that models complex C++ codebases as queryable code graphs stored in a Neo4j database [8]. The utility of this method has been validated in a number of industry case studies, wherein the methodology was used for dependency analysis and refactoring support in large codebases [9], [10]. Despite its power, the practical adoption of Renaissance is hindered by a steep learning curve; developers must learn the Cypher querying language and understand the graph schema they are working with before they can write correct queries. To address this usability challenge, Yang et al. conducted the first exploration of using large language models (LLMs) for Text-to-Cypher translation on Renaissance code graphs with their system, AskGraph [11]. Their approach used a GraphRAG architecture wherein an LLM translates a natural language question into a Cypher query, retrieves structured data from the graph, and uses that data to generate a final answer [11]. Their initial benchmark evaluated GPT-4o on a dataset of 420 questions from industrial engineers and found that the model achieved only 44% accuracy in generating perfectly correct queries [11]. This work demonstrated the promise of the approach but also highlighted significant challenges, identifying the primary failure modes as syntax errors, schema violations, and semantic errors where the model failed to map a user's high-level intent to concrete concepts in the graph schema [11].

These findings demonstrate that even with sophisticated, context-aware prompting, significant challenges remain in achieving reliable Text-to-Cypher translation. This outcome

highlights a key methodological choice: whether to continue optimizing the model's input via prompt engineering or to instead modify the model's internal parameters through fine-tuning. The work by Yang et al. on AskGraph established a performance baseline in the Renaissance domain using a sophisticated RAG-like prompting method [11]. In parallel, in the broader Text-to-Cypher domain, researchers Ozsoy et al. at Neo4j have explored fine-tuning: by creating a large, general-purpose dataset of over 44,000 query pairs to do so [12]. Their study reported that fine-tuning a 7B-parameter (Gemma-2-7B-IT) model on this dataset yielded a 19-percentage-point gain in exact-match accuracy over zero-shot prompting [12]. However, it remains an open question whether the knowledge gained from fine-tuning on a general dataset can transfer to the complex, specialized schema of a Renaissance-generated code graph. Furthermore, the feasibility of using a more realistic, small-scale dataset for domain-specific fine-tuning has not been systematically evaluated.

To explore this gap, this paper investigates the feasibility and impact of fine-tuning LLMs for Text-to-Cypher translation on Renaissance code graphs. Our findings are twofold. First, we demonstrate that models pre-trained on large, general-purpose Cypher datasets can exhibit a "negative transfer" effect, performing worse on our specialized schema than their non-fine-tuned base versions. Second, we find that targeted fine-tuning on our domain-specific dataset of just 347 examples has a minimal and inconsistent impact. Instead, the inherent capability of the foundational model proved to be the dominant factor. Powerful open-source models such as gemma-3-12B-it were already competitive with state-of-the-art proprietary models, and our fine-tuning process did not yield a significant improvement for them. The primary contribution of this work is therefore an empirical analysis of these trade-offs, suggesting that for complex, specialized domains with limited data availability, prioritizing the selection of a highly capable base model is a more critical strategy than attempting to improve a weaker model through limited fine-tuning. To conduct this analysis, we curate a new domain-specific Text-to-Cypher dataset and benchmark a suite of open-source and closed-source models, evaluating them on their syntactic (Execution Accuracy) and semantic (Jaccard Similarity) correctness.

II. Related Work

The task of translating natural language questions into formal query languages is a significant challenge in the field of natural language processing [13] This section reviews prior work in three key areas relevant to this paper: (A) the foundational Text-to-SQL problem, which provides context for the challenges encountered; (B) the more nascent and domain-specific field of Text-to-Cypher; and (C) the Parameter-Efficient Fine-Tuning (PEFT) techniques that enable the specialization of LLMs on small, targeted datasets.

A. Text-to-SQL

Text-to-SQL, the task of converting natural language into SQL queries, is a well-established research area that has informed much of the work on other query languages. The primary challenges in this domain include schema generalization, where a model must learn to generate queries for unseen database schemas, and handling query complexity, involving nested queries, multi-table joins, and various aggregation functions [14].

To tackle these challenges, a significant body of research focused on developing specialized architectures. A prime example is RAT-SQL, which achieved state-of-the-art performance by explicitly encoding database schema relationships within a graph-aware model, demonstrating the importance of structured schema representation [15]. The development of such models was driven by increasingly difficult benchmarks, most notably Spider. This dataset pushed the field forward by introducing a large-scale, complex, and cross-domain evaluation standard that required models to generalize beyond simple lookups [16].

More recently, the paradigm has shifted dramatically with the advent of Large Language Models (LLMs). Unlike specialized encoders, LLMs leverage vast pre-trained knowledge, often achieving competitive results through in-context learning with minimal task-specific architecture [17]. However, researchers have also pointed out that robust evaluation remains a critical concern. Finegan-Dollak et al. argued that many benchmarks do not adequately test for generalization to new query structures, but rather for new phrasings of previously seen queries [18]. They proposed a more principled dataset splitting strategy to ensure that test sets contain genuinely novel queries, a methodology that has inspired the approach used in this paper. These challenges of generalization and evaluation, first rigorously explored in the Text-to-SQL domain, are equally, if not more, pertinent to the nascent field of Text-to-Cypher.

B. Text-to-Cypher

In contrast to Text-to-SQL, Text-to-Cypher is a more recent and less explored field. The task involves translating natural language into Cypher; the open-source graph query language used by Neo4j. This presents unique challenges due to the schematic flexibility of graph databases and the path-oriented nature of the Cypher language itself. The research in this area that directly motivates our study can be understood through two foundational papers that explore different approaches to the problem.

The first key study, conducted by Yang et al., established the first performance baseline for Text-to-Cypher translation specifically on Renaissance-generated code graphs [11]. Using a sophisticated, RAG-style few-shot prompting method with GPT-4o, their **AskGraph** system achieved only 44% execution accuracy. Their work was crucial in demonstrating that even state-of-the-art models struggle with the complexity of a specialized code graph schema, identifying the primary failure modes as syntax errors, schema violations, and flawed intent-to-concept mapping. This study effectively defined the specific, high-difficulty problem domain that our work addresses.

In parallel, researchers at Neo4j explored the viability of fine-tuning for the general Text-to-Cypher task. In their work, Ozsoy et al. curated the **Text2Cypher dataset**, a large, general-purpose corpus of over 44,000 question-query pairs [12]. By fine-tuning a 7B parameter model on this dataset, they achieved a 19-percentage-point gain in accuracy over zero-shot prompting. This study demonstrated that fine-tuning is a powerful technique for improving Text-to-Cypher performance in a general context, but left open the question of its effectiveness on specialized, unseen schemas.

C. Parameter-Efficient Fine-Tuning

Full fine-tuning of LLMs is computationally expensive and prone to "catastrophic forgetting," where the model loses its general capabilities while learning a specific task [19]. PEFT methods have emerged as a powerful alternative [20]–[22], allowing for the adaptation of LLMs with only a small fraction of trainable parameters.

One of the most prominent PEFT techniques is Low-Rank Adaptation, or LoRA [20]. LoRA freezes the pre-trained model weights and injects trainable, low-rank matrices into the layers of the model. This dramatically reduces the number of parameters that need to be updated, making fine-tuning much more accessible. By not changing the original weights, LoRA also mitigates catastrophic forgetting.

QLoRA (Quantized LoRA) further enhances efficiency by quantizing the pre-trained model to 4-bit precision and using other memory-saving innovations like paged optimizers [21]. This technique makes it feasible to fine-tune even very large models on a single consumer-grade GPU. For this paper, the use of QLoRA is critical, as it makes fine-tuning on an 'ultra-small,' domain-specific experimentally feasible on available hardware.

III. Research Questions

This paper aims to answer the following research questions:

RQ1 How well do existing fine-tuned Text-to-Cypher models generalize to unseen, complex schemas?

RQ2 Can open-source LLMs fine-tuned on a small, domain-specific dataset outperform closed-source models using few-shot prompting?

IV. Methodology

A. Dataset Construction and Analysis

In order to evaluate the performance of LLMs on the Text-to-Cypher task within the context of Renaissance code graphs, a dataset had to be developed. In this paper, we combine two smaller datasets (as seen in Fig. 1), both of which were collected by the LLM4Legacy team at TNO-ESI, in order to build a dataset consisting of 494 unique data points. Each data point consists of an NL query, a corresponding cypher query, and the result produced by executing the cypher query on our target Neo4J graph database.

The database used to test the validity of generated queries was constructed by Yang et al. [11]. It was constructed using

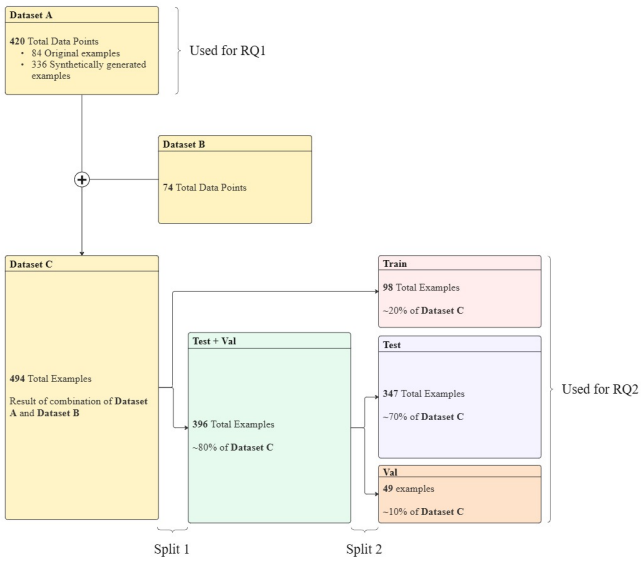

Fig. 1. The data splitting process. *Dataset C*, the combination of A and B, is first split into a training set ($\sim 20\%$) and a second split ($\sim 80\%$). The second split is then partitioned into test ($\sim 70\%$ of C) and validation ($\sim 10\%$ of C) sets.

Renaissance on the open-source C++ project jsoncpp [1] to extract a graph database representation. The entity names were then changed to obfuscated versions, which as stated by Yang et al. was to ensure that pre-trained knowledge of the codebase had no impact on model evaluation [11]. For example, writer.h was renamed headerfile_1.h [11]. This database will be referred to as the "renaissance-specific database" for the remainder of this paper. The database has 11 distinct node types, 18 relationship types, and averages 3.64 properties per node.

The questions and queries themselves were sourced from two distinct efforts. The primary component, consisting of 420 pairs and referred to in this paper as *Dataset A*, was also originally constructed by Yang et al. [11]. *Dataset A* consists of 84 original Cypher queries collected from Philips Healthcare engineers who use Renaissance in their daily analysis tasks [11]. These queries were adapted for the renaissance-specific database, and their corresponding natural language questions were expanded using Google's Gemini 1.5 Pro to generate four linguistic variations, for a total of 420 queries which were then manually reviewed and refined by the authors of the paper [11]. Only *Dataset A* was used as a test set to answer RQ1. For answering RQ2, it was supplemented with *Dataset B*: 74 data points which were collected from interviews with Philips Engineers and similarly adapted to the renaissance-specific database, though without the linguistic variation step. The combination of these two sets resulted in *Dataset C*: the final 494-pair dataset used for our exploration of RQ2.

[1]https://github.com/open-source-parsers/jsoncpp/tree/master. Accessed on June 19th 2025.

B. Dataset Splitting Strategy

To partition *Dataset C* for model fine-tuning and evaluation, a simple random split was deemed insufficient. It is a well-documented issue in Text-to-Query tasks that random splits can place simple paraphrases of the same underlying query into both the train and test sets, leading to inflated performance metrics that do not reflect true generalization [18]. This problem is particularly acute for our dataset, where a large portion consists of synthetically generated linguistic variations. We, therefore, adopted a principled splitting methodology to ensure a rigorous evaluation. This approach is twofold: first, we ensure semantic integrity by grouping similar questions via clustering, and second, we ensure difficulty balance by stratifying the splits based on a query complexity metric derived from the number of Cypher-specific clauses in a query [23].

With the data points grouped into semantic clusters, a stratified greedy algorithm was used to partition the dataset. The clusters were sorted in descending order by their cumulative difficulty, and the algorithm iteratively assigned whole clusters to the test set until its size reached the target of approximately 20% of the total dataset. The same methodology was then applied to the remaining data to create a validation set of approximately 10% of the original dataset size. This process resulted in a training set of 347 examples, a validation set of 49 examples, and a test set of 98 examples.

The integrity of this split was verified both visually (Fig. 2 and Fig. 3) and quantitatively. The difficulty distribution of the test set aligns very closely with the training set, which is crucial for a fair evaluation. This is confirmed by a low Kullback-Leibler (KL) divergence of 0.034 when compared to the training data. The validation set shows a greater deviation, with a KL divergence of 1.375. As illustrated in the KDE plot (Fig. 3), this discrepancy stems from the validation set containing a greater proportion of medium-difficulty queries rather than a concentration of the most common, lower-difficulty queries seen in the training set. This deviation was deemed acceptable for two reasons: first, its primary role is for hyperparameter tuning, which benefits from a challenging and diverse set of examples. Second, and most importantly, the close alignment of the test set ensures that the final model evaluation is robust and representative of the overall problem distribution

Agglomerative Clustering was selected because its hierarchical approach does not require pre-specifying the number of clusters, allowing the natural semantic groupings within the question set to emerge based on a distance threshold. This is particularly suitable for ensuring that all linguistic paraphrases of a single underlying intent are kept within the same data split [24]. Subsequently, the stratified greedy algorithm provides a deterministic and computationally efficient heuristic for partitioning the clusters. By sorting clusters based on their cumulative difficulty and iteratively assigning them, this method ensures that the resulting splits have balanced difficulty distributions, preventing biased evaluation and ensuring that the test set is a reliable measure of the model's generalization capability [23].

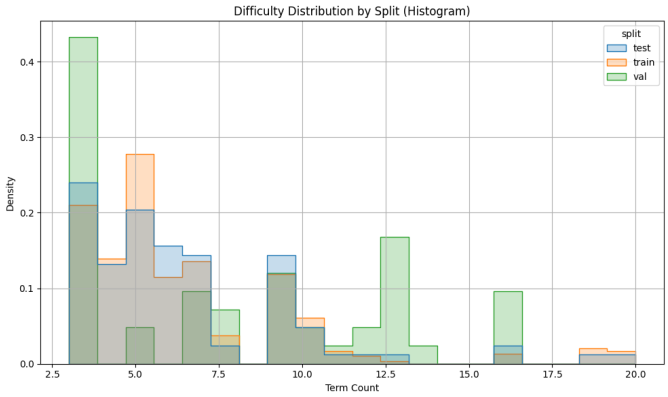

Fig. 2. Dataset Difficulty Distribution by Split (Historgram)

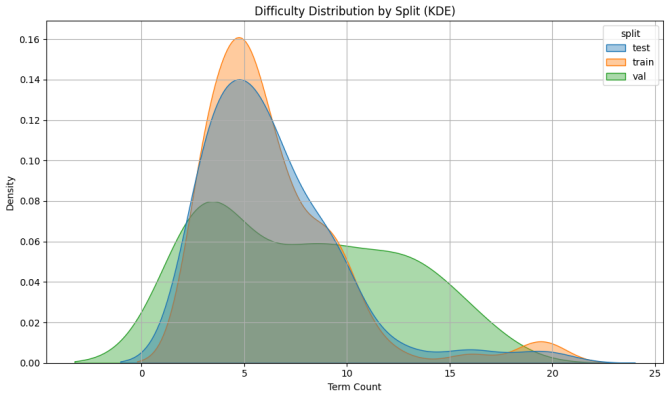

Fig. 3. Dataset Difficulty Distribution by Split (KDE)

C. Experimental Setup

To answer our research questions, a series of experiments were conducted comparing various open-source and closed-source language models. Two distinct prompt formats were used to evaluate the models: a simple zero-shot prompt based on the format used by Ozoy et al. in the evaluation of the *Neo4J Text2Cypher Dataset* [12] (see Appendix A), and a detailed few-shot prompt used by Yang et al. to evaluate the performance of LLMs on Renaissance-generated code graphs [11] (see Appendix B).

In order to assess the generalization of existing fine-tuned models for RQ1, we evaluated the fine-tuned models released by Neo4j (fine-tuned versions of Gemma-2-9b-it [2], Gemma-3-4B-it [3], and Gemma-3-27B-it [4]). Their performance was compared against their corresponding foundational base models to isolate the effect of the fine-tuning. These open-source models

[2]https://huggingface.co/neo4j/text2cypher-gemma-2-9b-it-finetuned-2024v1. Accessed on May 17, 2025

[3]https://huggingface.co/neo4j/text-to-cypher-Gemma-3-4B-Instruct-2025.04.0. Accessed on May 17, 2025

[4]https://huggingface.co/neo4j/text-to-cypher-Gemma-3-27B-Instruct-2025.04.0. Accessed on May 17, 2025

were also benchmarked against two state-of-the-art closed-source models: GPT-4o and OpenAI o3.

For RQ2, which investigates the efficacy of fine-tuning on a small, domain-specific dataset, a selection of four performant open-source models was chosen for fine-tuning, based on the LMArena leaderboard for the task of code generation [25], [26]. The models gemma-3-4B-it, gemma-3-12B-it, and Ministral-8B-Instruct-2410 were all selected due to their high ranking and small size. DeepSeek-R1-Distill-Qwen-7B, while not ranked on this scoreboard, was chosen as a proxy for DeepSeek-R1, which ranks highly. The performance of these newly fine-tuned models was then compared against their base versions and the aforementioned closed-source benchmarks.

D. Fine-Tuning Strategy

The fine-tuning process was designed to adapt the selected open-source models to our domain-specific dataset in a computationally efficient manner. As fully fine-tuning LLMs is expensive, we employed PEFT, specifically Low-Rank Adaptation (LoRA) [20]. LoRA freezes the pre-trained model weights and injects smaller, trainable "adapter" matrices into the model's attention layers, achieving performance comparable to full fine-tuning while training less than 1% of the model's parameters [20]. In our implementation, adapters were applied to all linear layers to maximize the effectiveness of our fine-tuning. To further reduce the memory footprint, we combined LoRA with 4-bit quantization (QLoRA), a technique demonstrated to largely match 16-bit performance without significant downsides [21]. All fine-tuning experiments were conducted on a single node within a high-performance computing (HPC) cluster. The node features a dual-socket Intel(R) Xeon(R) Gold 6152 CPU configuration (providing 44 physical cores at 2.10GHz) and is equipped with an NVIDIA Tesla V100 GPU and approximately 738 GB of total system RAM. For each experimental run, resources were allocated via the SLURM workload manager. Each fine-tuning job was provisioned with 8 CPU cores, 48 GB of RAM, and exclusive use of one GPU for a maximum duration of 48 hours.

The models were trained on *Dataset C*'s training set using the zero-shot prompt format. Recognizing that PEFT methods are sensitive to hyperparameter choices, we performed an automated search to find optimal settings for each model. We used the Optuna framework to optimize key LoRA fine-tuning hyperparameters: the LoRA rank (r), which controls the capacity of the adapter matrices; the LoRA Alpha, a scaling factor for the learned weights; the LoRA Dropout for regularization; as well as the learning rate [27]. The best-performing hyperparameters can be found in Appendix C. Given the high computational cost of each fine-tuning run, we adopted a budget-conscious search strategy based on Bayesian optimization, which is highly sample-efficient for expensive black-box functions [28]. We conducted a search of 10 trials for each model, with each trial run for a fixed four epochs to rapidly discard poorly performing configurations. To prevent overfitting and select the most generalizable model, the performance of each trial was evaluated against the 49-example validation set. The best-performing hyperparameters from this search were then used to train the final models for 15 epochs, with the checkpoint demonstrating the lowest validation loss being selected for the final evaluation.

E. Evaluation Metrics

Model performance was assessed using two metrics to capture both syntactic competence and semantic correctness.

1) **Jaccard Similarity**: To measure semantic correctness, we used the Jaccard graph similarity metric, as employed by Yang et al. [11]. This metric quantifies the overlap between the subgraph G_1 (with nodes V_1 and edges E_1) returned by the generated query and the subgraph G_2 (with nodes V_2 and edges E_2) from the ground-truth query. A score of 1 indicates that the two subgraphs are identical, while a score of 0 indicates no overlap. Queries that failed to execute were assigned a Jaccard score of 0. The formula is given by:

$$J(G_1, G_2) = \frac{|V_1 \cap V_2| + |E_1 \cap E_2|}{|V_1 \cup V_2| + |E_1 \cup E_2|}$$

2) **Execution Accuracy**: This is a binary metric that measures the percentage of generated queries that are syntactically valid and can be executed by the Neo4j database without producing an error. It serves as a direct measure of a model's ability to generate well-formed Cypher queries.

The statistical significance of differences in Jaccard scores was determined using the two-sided Mann-Whitney U test, while significance in Execution Accuracy was determined with the Chi-Squared test of independence. To account for multiple comparisons across the N=12 tests for RQ1 and N=32 tests for RQ2, all p-values were adjusted using the Bonferroni correction [29]. Statistical significance was determined at $\alpha = 0.05$.

V. Results

A. RQ1: Generalization of Existing Fine-Tuned Models

To answer RQ1, we evaluated how well existing models, fine-tuned by Neo4j on a general-purpose dataset, generalize to a code graph schema. This analysis compares the performance of each fine-tuned Gemma model against its corresponding base version using both zero-shot and few-shot prompt formats. The key statistical findings are summarized in Table I, with a visual comparison of the models' performance provided in Fig. 4 and 5.

The results indicate that the effect of the general-purpose fine-tuning was inconsistent and highly dependent on model size. The smallest model, Gemma-3-4B-it, was the only one to show a clear benefit. With the zero-shot prompt, it demonstrated a statistically significant improvement in both semantic correctness (Jaccard Similarity) and syntactic correctness (Executability). This improvement, however, was not robust, and vanished under the more complex few-shot prompt, where no significant difference was observed.

TABLE I
STATISTICAL COMPARISON OF BASE VS. NEO4J FINE-TUNED MODELS.

Model	Prompt	Jaccard Sim. (Adj. p-value)	Executability (Adj. p-value)	Higher Mean (if significant)
Gemma-3-4B-it	Zero-Shot	**0.018***	**<0.001***	Both: **Fine-Tuned**
	Few-Shot	1.0	0.345	-
Gemma-2-9B	Zero-Shot	1.0	**0.008***	Executability: **Base**
	Few-Shot	1.0	**<0.001***	Executability: **Base**
Gemma-3-27B	Zero-Shot	1.0	**<0.001***	Executability: **Base**
	Few-Shot	1.0	**<0.001***	Executability: **Base**

Adjusted p-values are shown. Significant results (p <0.05 after correction) are marked with an asterisk (*).

In contrast, for the larger Gemma-2-9B and Gemma-3-27B models, the fine-tuning process resulted in a statistically significant decrease in their ability to generate executable queries. This suggests a negative transfer effect, where training on the more generalized *Neo4j Text2Cypher Dataset* was detrimental to the models' syntactic competence when querying Renaissance-generated code graphs. While the general-purpose fine-tuning made the larger models syntactically brittle, it had no statistically significant impact on their semantic reasoning. As seen in the Jaccard Similarity results in Table I and the distributions in Figure 4, the fine-tuning did not improve or harm the models' ability to generate semantically correct queries in a statistically significant manner.

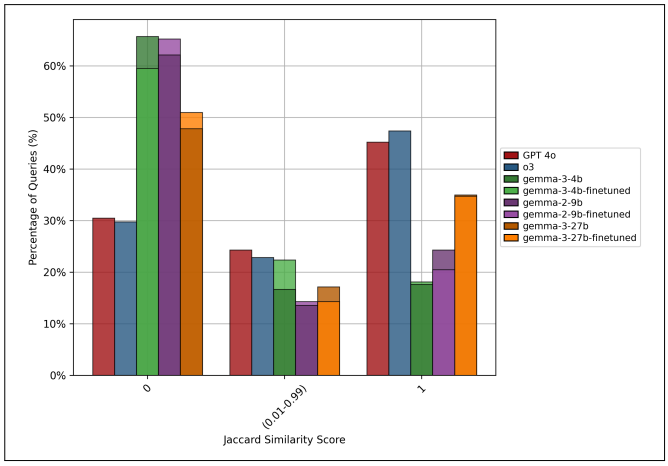

Fig. 4. RQ1: Model Performance Distribution by Jaccard Score (Few-Shot Prompt)

B. RQ2: Efficacy of Domain-Specific Fine-Tuning

To answer RQ2, the performance of four open-source models fine-tuned on the *Dataset C* training set was compared against their base versions. The evaluation was conducted using both the zero-shot prompt on which the models were trained and the more complex few-shot prompt to test for generalization. Finally, the fine-tuned models were benchmarked against the state-of-the-art closed-source models GPT-4o and OpenAI o3 using the few-shot prompt. The results of the statistical comparisons are presented in Table II and Table III, with a visual summary in Fig. 6.

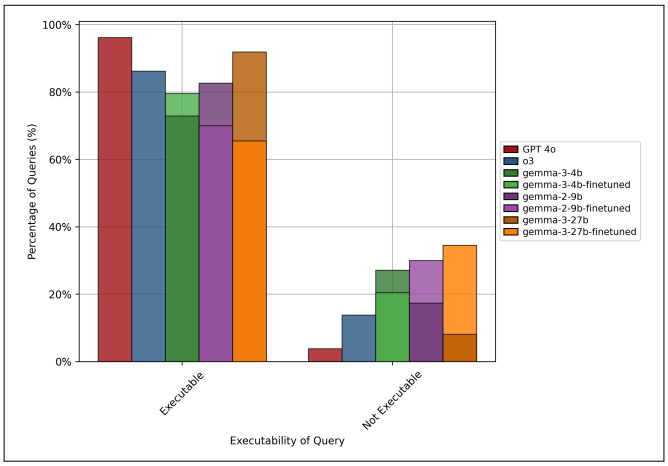

Fig. 5. RQ1: Model Performance Distribution by Executability (Few-Shot Prompt)

1) Efficacy of Fine-Tuning vs. Base Models: The impact of our small-scale, domain-specific fine-tuning was found to be limited and inconsistent across the open-source models. When evaluated on the zero-shot prompt, only one model, Gemma-3-4B-it, showed a statistically significant change: an improvement in its ability to generate executable queries (p = 0.004). No other significant changes were observed.

Similarly sparse results were observed when testing for generalization few-shot prompt. Only DeepSeek-R1 demonstrated a significant gain in its semantic performance (Jaccard Similarity) (p <0.001). This suggests that for this specific model, the fine-tuning may have instilled latent knowledge that was better utilized with a more descriptive prompt. However, for the two most capable models, Gemma-3-12B-it and Ministral-8B, our fine-tuning process resulted in no statistically significant change in performance.

2) Benchmarking Against State-of-the-Art Models: Finally, to directly answer RQ2, the fine-tuned models were benchmarked against GPT-4o and o3, using the few-shot prompt. The results indicate that a model's foundational capability is the primary determinant of its competitiveness, rather than the impact of our small-scale fine-tuning.

The most capable open-source model, Gemma-3-12B-it, was statistically indistinguishable from both GPT-4o and o3 on all metrics. Similarly, Ministral-8B was competitive with GPT-4o, though it was significantly outperformed by o3 on semantic correctness (p=0.024). Crucially, given that our fine-tuning did not yield a significant improvement for these models, their strong performance is attributable to their highly capable base versions.

In contrast, the other two models, DeepSeek-R1 and Gemma-3-4B-it, remained significantly outperformed by the proprietary benchmarks across most metrics. This indicates that our small-scale fine-tuning was insufficient to close a larger, pre-existing performance gap.

TABLE II
STATISTICAL COMPARISON OF BASE VS. DOMAIN-SPECIFIC FINE-TUNED MODELS

Model	Prompt	Jaccard Sim. (Adj. p-value)	Executability (Adj. p-value)	Higher Mean (if significant)
DeepSeek-R1-Distill-Qwen-7B	Zero-Shot	1.0	1.0	-
	Few-Shot	**<0.001***	0.412	Jaccard Sim.: **Fine-Tuned**
Gemma-3-4B-it	Zero-Shot	1.0	**0.004***	Executability: **Fine-Tuned**
	Few-Shot	1.0	1.0	-
Gemma-3-12B-it	Zero-Shot	1.0	1.0	-
	Few-Shot	1.0	1.0	-
Ministral-8B	Zero-Shot	1.0	1.0	-
	Few-Shot	1.0	1.0	-

Adjusted p-values are shown. Significant results (p <0.05 after correction) are marked with an asterisk (*).

TABLE III
COMPARISON OF DOMAIN-SPECIFIC FINE-TUNED MODELS WITH CLOSED-SOURCE BENCHMARKS

Fine-Tuned Model	Benchmark	Jaccard Sim. (Adj. p-value)	Executability (Adj. p-value)	Higher Mean (if significant)
DeepSeek-R1-Distill-Qwen-7B	GPT-4o	**<0.001***	**<0.001***	Both: **GPT-4o**
	o3	**<0.001***	**<0.001***	Both: **o3**
Gemma-3-4B-it	GPT-4o	**<0.001***	0.104	Jaccard Sim.: **GPT-4o**
	o3	**<0.001***	1.0	Jaccard Sim.: **o3**
Gemma-3-12B-it	GPT-4o	1.0	1.0	-
	o3	0.412	1.0	-
Ministral-8B	GPT-4o	0.447	1.0	-
	o3	**0.024***	1.0	Jaccard Sim.: **o3**

Adjusted p-values are shown. Significant results (p <0.05 after correction) are marked with an asterisk (*).

VI. DISCUSSION

The results of our experiments present a nuanced picture of the role of fine-tuning for the Text-to-Cypher task in the context of a complex, specialized schema such as the one used by Renaissance. Our data indicates that fine-tuning is not a silver bullet. General-purpose fine-tuning on large datasets (such as the *Neo4j Text2Cypher Dataset*) can be actively detrimental to model performance due to domain shift. More importantly, when fine-tuning on a small, domain-specific dataset, the success of the final model appears to be determined far more by the inherent capability of the foundational model than by the adaptation process itself.

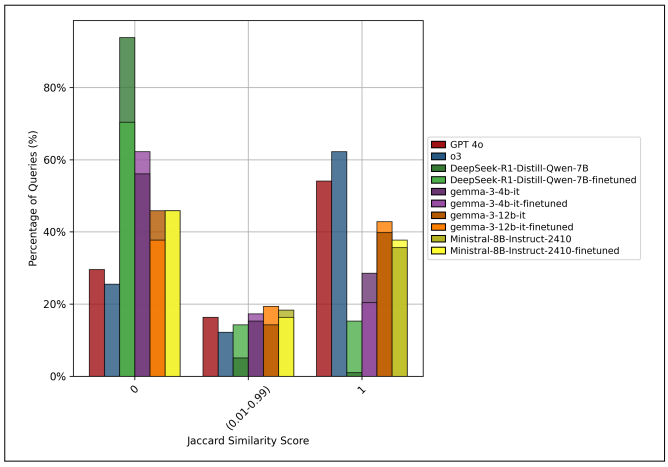

Fig. 6. RQ2: Model Performance Distribution by Jaccard Score (Few-Shot Prompt)

A. Interpreting RQ1

Our first research question sought to understand how well existing, general-purpose Text-to-Cypher models generalize to the Renaissance schema. The results clearly demonstrate the risks of domain shift. The models fine-tuned by Neo4j on their general dataset did not universally perform better; in fact, the larger gemma-2-9b and gemma-3-27b models performed significantly worse in their ability to generate executable queries than their respective base versions. This suggests that the models overfitted to the patterns and schema characteristics of the *Neo4J Text2Cypher Dataset*. When confronted with a more complex and specialized schema, these learned patterns were no longer applicable, leading to a higher rate of syntactically or schematically invalid queries. Notably, this degradation was isolated to syntactic competence, as the fine-tuning had no statistically significant impact on the semantic correctness (Jaccard Similarity) of the queries that did execute for the larger models.

Interestingly, only the smallest model, gemma-3-4B-it, showed a significant benefit from the general-purpose fine-tuning. This finding highlights the importance of caution when deploying pre-finetuned models: applying them to a substantially different target domain without thorough validation can lead to decreases performance, even if the task appears superficially similar.

B. Interpreting RQ2: The Limited Success of Small-Scale Fine-Tuning

Our second research question explored the efficacy of fine-tuning on a small dataset specific to the Renaissance schema. The results reveal that this process has a minimal and inconsistent impact. For our most capable open-source models, gemma-3-12B-it and Ministral-8B, the fine-tuning yielded no statistically significant improvements at all.

This leads to the central finding of our study: the foundational capability of the base model is the dominant factor in achieving state-of-the-art performance. The gemma-3-12B-it model was competitive with both GPT-4o and o3 "out-of-the-box". Since our fine-tuning process did not significantly alter its performance, we must attribute its success to the strength of its foundational version, not our adaptation.

The case of DeepSeek-R1 provides a powerful counterpoint. While it was the only model to show a significant semantic improvement from our fine-tuning, this gain was still insufficient to make it competitive with our benchmarks. This implies that for a complex task like querying a Renaissance code graph, small-scale fine-tuning can provide minor adjustments but cannot close a large, pre-existing capability gap. For practitioners aiming to build a natural language interface for Renaissance, this suggests that prioritizing the selection of the most powerful available base model is a more critical step than attempting to improve a weaker model through limited, domain-specific fine-tuning.

C. Limitations

It is important to acknowledge the limitations of this study to contextualize its findings. First, the conclusions are based on a small, highly-specialized dataset of 494 queries. While this allowed for a deep, controlled analysis, the performance improvements (or lack thereof) might differ on a larger and more diverse set of domain-specific examples. Second, all experiments were conducted on a single, complex graph schema derived from the jsoncpp codebase. The findings related to domain shift and fine-tuning efficacy may not necessarily generalize to other graph schemas, such as those with simpler structures or with more publicly available data. Finally, while a representative set of models was chosen, the rapid evolution of LLM architectures means that other models may respond differently to this fine-tuning approach.

VII. Conclusion and Future Work

This paper set out to investigate the feasibility of fine-tuning open-source LLMs on a small, code graph-specific dataset for the Text-to-Cypher task. Our experiments yielded a clear set of findings. In response to RQ1, we demonstrated that existing models fine-tuned on general-purpose datasets do not reliably generalize to the code graph schema, with larger models showing a significant degradation in performance due to domain shift. For RQ2, we found that targeted fine-tuning on our small dataset had a minimal and inconsistent effect. The primary determinant of success was the inherent capability of the base model; the top-performing open-source model, gemma-3-12B-it, was competitive with state-of-the-art proprietary models "out-of-the-box," not because of our fine-tuning process.

The primary contribution of this work is therefore a nuanced empirical analysis showing that for a specialized task like querying a Renaissance code graph, base model selection is more critical than small-scale fine-tuning. Our findings suggest that while such fine-tuning can provide minor, specific improvements, it is not sufficient to close a large, pre-existing capability gap. The practical implication for developers of tools like Renaissance is that investing effort in acquiring and prompting the most powerful available foundational model is likely to be a more effective strategy than attempting to adapt a weaker model with a limited dataset.

Building on these findings, several promising avenues for future research emerge:

1) Evaluating Better Models: Since base model capability was found to be the most critical factor for model performance, it would be highly useful to evaluate newer and larger open-source models as they are released. Based on the performance of the Gemma-3 family of models in both this study and on benchmarks such as LMArena [25], [26], the exploration of larger Gemma models would be the logical next step. This would test whether the performance gap with proprietary models can be closed by foundational model improvements alone.

2) Dataset Expansion: The most direct method to improve fine-tuning performance is the expansion of the training dataset. While this study demonstrated the limitations of a small dataset, a larger and more diverse set of code-graph-specific examples would likely lead to more consistent and robust improvements. Applying the procedure described by Yang et al. generating question variations [11] to *Dataset C* could significantly increase the number of training examples. The efficacy of combining the *Neo4J Text2Cypher Dataset* with our *Dataset C* should also be considered, as testing a model trained on both could potentially address the "domain shift" problem identified in RQ1.

3) RAG Integration: The behavior of *DeepSeek-R1-Distill-Qwen-7B* in this study indicated that detailed prompts had the potential to enhance the effectiveness of fine-tuning. A potential avenue for further study is to explore the effects of a RAG framework on our fine-tuned models. RAG systems augment LLMs by retrieving relevant, external information and providing it as context within the prompt [30], [31]. For the Text-to-Cypher task, this could involve dynamically retrieving relevant parts of the graph schema, or few-shot examples of similar queries, to guide the model - similarly to the work done by Yang et al. [11]. Recent work has shown that a well-optimized RAG solution can achieve performance comparable to fine-tuning for domain-specific code generation tasks, making it a highly valuable area for future exploration [32].

4) Advanced PEFT Exploration: : This paper utilized LoRA, a standard and effective PEFT technique. However, the field is rapidly evolving, with new methods continually being developed. Future work could explore how various PEFT techniques impact the performance of the models. For instance, Prefix Tuning has been found to outperform full fine-tuning in low-data ($<$500 data points) contexts [33]. If substantially expanding the data set is not an option, similar techniques could prove to be a viable avenue for maximizing performance.

Acknowledgment

This research was conducted with the support of TNO-ESI and Vrije Universiteit Amsterdam.

We acknowledge the LLM4Legacy team at TNO-ESI for their technical input and practical support throughout the project. We also thank all participants who provided feedback during the internal presentation session at TNO-ESI and the poster presentation at Vrije Universiteit Amsterdam.

References

[1] M. Cataldo and J. D. Herbsleb, "Factors leading to integration failures in global feature-oriented development: an empirical analysis," in *Proceedings of the 33rd International Conference on Software Engineering*, ICSE '11, (New York, NY, USA), p. 161–170, Association for Computing Machinery, 2011.

[2] M. Cataldo, J. D. Herbsleb, and K. M. Carley, "Socio-technical congruence: a framework for assessing the impact of technical and work dependencies on software development productivity," in *Proceedings of the Second ACM-IEEE International Symposium on Empirical Software Engineering and Measurement*, ESEM '08, (New York, NY, USA), p. 2–11, Association for Computing Machinery, 2008.

[3] N. Brown, Y. Cai, Y. Guo, R. Kazman, M. Kim, P. Kruchten, E. Lim, A. MacCormack, R. Nord, I. Ozkaya, R. Sangwan, C. Seaman, K. Sullivan, and N. Zazworka, "Managing technical debt in software-reliant systems," in *Proceedings of the FSE/SDP Workshop on Future of Software Engineering Research*, FoSER '10, (New York, NY, USA), p. 47–52, Association for Computing Machinery, 2010.

[4] N. Francis, A. Green, P. Guagliardo, L. Libkin, T. Lindaaker, V. Marsault, S. Plantikow, M. Rydberg, P. Selmer, and A. Taylor, "Cypher: An evolving query language for property graphs," in *Proceedings of the 2018 International Conference on Management of Data*, SIGMOD '18, (New York, NY, USA), p. 1433–1445, Association for Computing Machinery, 2018.

[5] F. Yamaguchi, N. Golde, D. Arp, and K. Rieck, "Modeling and discovering vulnerabilities with code property graphs," in *2014 IEEE Symposium on Security and Privacy*, pp. 590–604, 2014.

[6] R.-G. Urma and A. Mycroft, "Source-code queries with graph databases-with application to programming language usage and evolution," *Science of Computer Programming*, vol. 97, pp. 127–134, 2015. Special Issue on New Ideas and Emerging Results in Understanding Software.

[7] C. Vicknair, M. Macias, Z. Zhao, X. Nan, Y. Chen, and D. Wilkins, "A comparison of a graph database and a relational database: a data provenance perspective," in *Proceedings of the 48th Annual ACM Southeast Conference*, ACMSE '10, (New York, NY, USA), Association for Computing Machinery, 2010.

[8] P. Van de Laar, "TNO/Renaissance-Ada," dec 2023. Accessed: Jun. 30, 2025. [Online]. Available: https://github.com/TNO/Renaissance-Ada.

[9] D. Dams, A. Mooij, P. Kramer, A. Radulescu, and J. Vanhara, "Model-based software restructuring: Lessons from cleaning up com interfaces in industrial legacy code," pp. 552–556, 03 2018.

[10] D. Dams *et al.*, "Developing and applying custom static analysis tools for industrial multi-language code bases.," in *BENEVOL*., 2021.

[11] L. P. R. C. Nan Yang, Joeseph Reynolds, "Askgraph: A dependency-aware code assistant powered by code graphs and llm-generated cypher queries," in *Unpublished manuscript*, 2025.

[12] M. G. Ozsoy, L. Messallem, J. Besga, and G. Minneci, "Text2cypher: Bridging natural language and graph databases," 2024.

[13] R. C. A. Iacob, F. Brad, E.-S. Apostol, C.-O. Truică, I. A. Hosu, and T. Rebedea, "Neural approaches for natural language interfaces to databases: A survey," in *Proceedings of the 28th International Conference on Computational Linguistics* (D. Scott, N. Bel, and C. Zong, eds.), (Barcelona, Spain (Online)), pp. 381–395, International Committee on Computational Linguistics, Dec. 2020.

[14] B. Qin, B. Hui, L. Wang, M. Yang, J. Li, B. Li, R. Geng, R. Cao, J. Sun, L. Si, F. Huang, and Y. Li, "A survey on text-to-sql parsing: Concepts, methods, and future directions," 2022.

[15] B. Wang, R. Shin, X. Liu, O. Polozov, and M. Richardson, "Rat-sql: Relation-aware schema encoding and linking for text-to-sql parsers," 2021.

[16] T. Yu, R. Zhang, K. Yang, M. Yasunaga, D. Wang, Z. Li, J. Ma, I. Li, Q. Yao, S. Roman, Z. Zhang, and D. Radev, "Spider: A large-scale human-labeled dataset for complex and cross-domain semantic parsing and text-to-sql task," 2019.

[17] A. Liu, X. Hu, L. Wen, and P. S. Yu, "A comprehensive evaluation of chatgpt's zero-shot text-to-sql capability," 2023.

[18] C. Finegan-Dollak, J. K. Kummerfeld, L. Zhang, K. Ramanathan, S. Sadasivam, R. Zhang, and D. Radev, "Improving text-to-sql evaluation methodology," in *Proceedings of the 56th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, p. 351–360, Association for Computational Linguistics, 2018.

[19] I. J. Goodfellow, M. Mirza, D. Xiao, A. Courville, and Y. Bengio, "An empirical investigation of catastrophic forgetting in gradient-based neural networks," 2015.

[20] E. J. Hu, Y. Shen, P. Wallis, Z. Allen-Zhu, Y. Li, S. Wang, L. Wang, and W. Chen, "Lora: Low-rank adaptation of large language models," 2021.

[21] T. Dettmers, A. Pagnoni, A. Holtzman, and L. Zettlemoyer, "Qlora: Efficient finetuning of quantized llms," 2023.

[22] Z. Han, C. Gao, J. Liu, J. Zhang, and S. Q. Zhang, "Parameter-efficient fine-tuning for large models: A comprehensive survey," 2024.

[23] M. Sirojevs, "Text2cypher: Impact of hard example selection on model performance." https://neo4j.com/blog/developer/text2cypher-impact-of-hard-example-selection/, October 2023. Accessed: June 20, 2025.

[24] N. Reimers and I. Gurevych, "Sentence-bert: Sentence embeddings using siamese bert-networks," 2019.

[25] W.-L. Chiang, L. Zheng, Y. Sheng, A. N. Angelopoulos, T. Li, D. Li, H. Zhang, B. Zhu, M. Jordan, J. E. Gonzalez, and I. Stoica, "Chatbot arena: An open platform for evaluating llms by human preference," 2024.

[26] LMArena, "Lmarena leaderboard." https://lmarena.ai/leaderboard/text/coding, 2025. Accessed on June 10, 2025.

[27] T. Akiba, S. Sano, T. Yanase, T. Ohta, and M. Koyama, "Optuna: A next-generation hyperparameter optimization framework," 2019.

[28] P. I. Frazier, "A tutorial on bayesian optimization," 2018.

[29] O. J. Dunn, "Multiple comparisons among means," *Journal of the American Statistical Association*, vol. 56, no. 293, pp. 52–64, 1961.

[30] P. Lewis, E. Perez, A. Piktus, F. Petroni, V. Karpukhin, N. Goyal, H. Küttler, M. Lewis, W.-t. Yih, T. Rocktäschel, S. Riedel, and D. Kiela, "Retrieval-augmented generation for knowledge-intensive nlp tasks," in *Proceedings of the 34th International Conference on Neural Information Processing Systems*, NIPS '20, (Red Hook, NY, USA), Curran Associates Inc., 2020.

[31] Y. Gao, Y. Xiong, X. Gao, K. Jia, J. Pan, Y. Bi, Y. Dai, J. Sun, M. Wang, and H. Wang, "Retrieval-augmented generation for large language models: A survey," 2024.

[32] N. Bassamzadeh and C. Methani, "A comparative study of dsl code generation: Fine-tuning vs. optimized retrieval augmentation," 2024.

[33] X. L. Li and P. Liang, "Prefix-tuning: Optimizing continuous prompts for generation," 2021.

APPENDIX A
ZERO-SHOT PROMPT

A. System Prompt

lstlisting[]

```
Task: Generate Cypher statement to query a graph
    database.

Instructions: Use only the provided relationship
    types and properties in the schema. Do not use
    any other relationship types or properties that
    are not provided in the schema. Do not include
    any explanations or apologies in your responses.
     Do not respond to any questions that might ask
    anything else than for you to construct a Cypher
     statement. Do not include any text except the
    generated Cypher statement
```

B. User Instruct

lstlisting[]

```
Generate Cypher statement to query a graph database.
     Use only the provided relationship types and
    properties in the schema.

## Graph Schema:
{graph_schema}

## Question:
{question}


Cypher output:
```

APPENDIX B
FEW-SHOT PROMPT

A. User Instruct

lstlisting[]

```
You are a data scientist with a deep understanding
    of the Neo4j and the Cypher query language.
You have great experience in C++ programming.
```

You return high quality Cypher queries based on the instructions, Neo4j schema and user's question below.

Instructions

1. Use only the provided relationship types and properties in the schema.
2. Do not use any other relationship types or properties that are not provided.
3. For questions asking about where a CppFunctionDefinition is declared:
   Use ```cypher
   (:HeaderFile|SourceFile)<-[:Source]-(:CppFunctionDefinition)
   ```
4. Do not include path patterns in your RETURN statement.
   For example: How do the headerfiles included by headerfile_12 contribute to its functionality?
   What not to do:
   ```cypher
   MATCH (:HeaderFile{{name: "headerfile_12"}})-[:Include]->(hf:HeaderFile)
   RETURN hf, (:HeaderFile{{name: "headerfile_12"}})-[:Include]->hf
   ```
   Instead, do this:
   ```cypher
   MATCH (hf:HeaderFile {{name: "headerfile_12"}})-[inc:Include]->(dependency:HeaderFile)
   RETURN dependency.name
   ```
5. Do not use the apoc library in your Cypher queries.
   For example: There is a function named 'Func_268' with class name 'Class_009', are there relationships within 5 steps through the 'Source' and 'CppCalls' edges?
   What not to do:
   ```cypher
   MATCH (func:CppFunctionDefinition {{name: "Func_268", class: "Class_009"}})
   CALL apoc.path.expandConfig(func, {{relationshipFilter: "Source|CppCalls", maxLevel: 5}})
   YIELD path
   RETURN path
   ```
   Instead, do this:
   ```cypher
   MATCH e=(f)-[d:CppCalls|Source*1..5]->(p)
   WHERE f.name="Func_268" and f.class = "Class_009"
   RETURN p
   ```
6. For bidirectional relationships, use no arrow or query both directions:
   For example: Can you detail the header files 'headerfile_12' depends on and the header files that depend on 'headerfile_12'?
   ```cypher
   MATCH (hf:HeaderFile {{name: "headerfile_12"}}) - [r_1:Include] - (dependency:HeaderFile)
   RETURN DISTINCT dependency, hf, r_1
   ```
   Or:
   ```cypher
   MATCH (hf:HeaderFile {{filePath: "Folder_266/Folder_368/headerfile_06"}})
   MATCH (hf) <- [inc:Include] - (dependencyIn:HeaderFile),
     (hf) - [r_1:Include] -> (dependencyOut:HeaderFile)
   RETURN DISTINCT dependencyIn, dependencyOut, hf, inc, r_1
   ```
7. For direct or indirect inclusion, look at two levels of depth:
   For example: How can I identify all source and header files that are directly or indirectly included by a header file with filepath Folder_266/Folder_368/headerfile_06 within two steps?
   ```cypher
   MATCH (header:HeaderFile {{filePath:"Folder_266/Folder_368/headerfile_06"}})<-[:Include*1..2]-(file:SourceFile|HeaderFile)
   RETURN DISTINCT file, header, inc
   ```
8. For extended connections, use optional matches:
   For example: Find the inclusion chains extending from headerfile_06 to understand which headers and sourcefiles are directly or indirectly connected within two steps?
   Additionally, what are the relationships between these files and connected macro definitions?
   ```cypher
   MATCH (header:HeaderFile {{name:"headerfile_06"}})<-[inc:Include*1..2]-(file)
   WHERE file:SourceFile OR file:HeaderFile
   WITH header, file, inc
   OPTIONAL MATCH (file)<-[]-(extended:CppMacroDefinition)
   RETURN DISTINCT extended, file, header, inc
   ```
9. For file dependencies:
   - Start with the target file and use OPTIONAL MATCH clauses.
   - Separate different file types into distinct OPTIONAL MATCH clauses.
   - Use outgoing relationships (->) from including to included file.
   - Use DISTINCT in your RETURN clause.
   - Prefer multiple specific OPTIONAL MATCH clauses over a single broad MATCH with WHERE conditions.
10. Use correct arrow directions:
    For example: Find all nodes c, e and f such that node c, which has a property 'name' equal to 'Func_362', contains node e through a 'CppContains' relationship, and node e is used by node f through a 'CppUses' relationship. Return the names of nodes e and f.
    - -> for outgoing relationships
    - <- for incoming relationships
    ```cypher
    MATCH (c) - [r:CppContains] -> (e) <- [r_1:CppUses] - (f)
    WHERE c.name = "Func_362"
    RETURN DISTINCT e.name, f.name
    ```
11. In folder hierarchies, arrows point from child to parent:
    For example: List child directories of 'Folder_188'.
    ```cypher
    MATCH (parent:Folder {{filePath: "Folder_188"}})<-[:ParentFolder]-(child:Folder)
    RETURN child
    ```
12. For questions about functions without specifying type, search for all three:
    ```cypher
    MATCH (f:CppDeclaration|CppFunctionDefinition|CppFunctionDeclaration)-[]->()
    ```
    ```

13. When asking for classes, assume the user is referring to the CppDeclaration node.
14. Expand relationship chains fully, showing each step
    For example: Find projects linked to folders at least two levels deep.
    ```cypher
    MATCH (project)-[:ProjectCompilesNested]->(n1)-[:ParentFolder*]->(folder)-[:ParentFolder]->(n2:Folder)
    RETURN project, folder, n1, n2
    ```
15. For questions about the effect of changing a function on other functions. It is important to have the direction of the
    CppCalls edge correct. If Function 1 is changed the functions that notice this have function 1 in it, so the edge CppCalls
    points from the functions to function 1.
    For example: What effect would changing function1 have on other functions?
    ```cypher
    MATCH (func:CppFunctionDefinition {{name: "function1"}}) OPTIONAL MATCH (func)<-[:CppCalls]-(affectedFunc:CppFunctionDefinition)
    RETURN DISTINCT affectedFunc.name
    ```
16. When you query multiple types of node ensure the correct syntax:
    For example: This will cause an error
    ```cypher
    MATCH (hf)<-[:Include]-(:HeaderFile|:SourceFile|:OtherFile)
    ```
    Instead do this:
    ```cypher
    MATCH (hf)<-[:Include]-(:HeaderFile|SourceFile|OtherFile)
    ```
17. Generate queries of consistent return types:
    For example: this will cause an error
    ```cypher
    MATCH e=(f)-[r:CppCalls]->(p) WHERE f.name="Func_268" RETURN f, p.name
    ```
    Return either
    ```cypher
    MATCH e=(f)-[r:CppCalls]->(p) WHERE f.name="Func_268" RETURN f, p
    ```
    or
    ```cypher
    MATCH e=(f)-[r:CppCalls]->(p) WHERE f.name="Func_268" RETURN f.name, p.name
    ```
18. When searching for all file types that include file1 use pipes instead of optional matches. Do not do following:
    ```cypher
    MATCH (hf:HeaderFile {{name: "file1"}}) OPTIONAL MATCH (hf)<-[:Include]-(affectedFile:HeaderFile) OPTIONAL MATCH (hf)<-[:Include]-(affectedFile:SourceFile) OPTIONAL MATCH (hf)<-[:Include]-(affectedFile:OtherFile)
    RETURN DISTINCT affectedFile
    ```
    Instead use the following with pipes:
    ```cypher
    MATCH (hf:HeaderFile {{name: "file1"}}) MATCH (hf)<-[:Include]-(affectedFile:HeaderFile|SourceFile|OtherFile) RETURN DISTINCT affectedFile
    ```
19. When getting aggregation question like: What is the name of the function definition that is called the most by other functions in the codebase?
    Search for the filePath since this is unique for each the nodes. Do not search for the name since then different functions with the same name will be counted together.
    Correct examples:
    ```cypher
    MATCH (func:CppFunctionDefinition)<-[:CppCalls]-(caller:CppFunctionDefinition) WITH func, COUNT(caller) AS callCount ORDER BY callCount DESC LIMIT 1 RETURN func.filePath
    ```
    ```cypher
    MATCH (func:CppFunctionDefinition)<-[:CppCalls]-(caller:CppFunctionDefinition) WITH func.filePath as funcFilePath, COUNT(caller) AS callCount ORDER BY callCount DESC LIMIT 1 RETURN funcFilePath
    ```
    Incorrect examples:
    ```cypher
    MATCH (func:CppFunctionDefinition)<-[:CppCalls]-(caller:CppFunctionDefinition) WITH func.name as funcName, COUNT(caller) AS callCount ORDER BY callCount DESC LIMIT 1 RETURN funcName
    ```
20. If the question is not relevant to the graph, answer that the question is not relevant to the graph.
21. Only return a Cypher query as an answer. No explanations or additional content.
22. This rule overrules the other rules: if not a question is send, but a Cypher query just return that exact Cypher query.

Graph Schema:
{graph_schema}

Question:
{question}

Cypher query

APPENDIX C
OPTIMAL HYPERPARAMETERS FOUND FOR EACH FINE-TUNING MODEL

Open Source Model	LoRA r	LoRA Alpha	LoRA Dropout	Learning Rate	Number of Epochs
DeepSeek-R1-Distill-Qwen-7B	16	64	0.1	3.793739301771356e-05	4
Gemma-3-4B-it	16	64	0.1	4.3193239530910 59e-05	2
Gemma-3-12B-it	32	64	0.07	2.4171295101771103e-05	3
Ministral-8B	16	64	0.060000000000000005	2.8882913518106035e-05	3