RAS: Reflection-Augmented Scaling with In-Context Learning for Executable Cypher Query Generation

Anonymous Authors

Abstract

Inference-time scaling can reduce errors in structured query generation, but methods to allocate the compute for query code generation remains underexplored. We study Text2Cypher, where language models generate Cypher queries that execute against property graph databases. Non-executable queries constitute a distinct syntactic failure separate from semantic inaccuracy: a syntax error triggers a system-generated error message from the database. These error messages are typically discarded at inference time rather than leveraged through in-context learning (ICL). We compare two inference methods: Independent Scaling (IS), which performs memoryless resampling, and Reflection-Augmented Scaling (RAS), which conditions each new attempt on prior execution feedback via ICL. Across three Neo4j datasets and five code-specialized language models, RAS reduces the Query Execution Error Rate by 41–50% at n=5, outperforming IS at 32–38%. Execution errors are not merely failures to discard but actionable feedback, and structuring inference-time compute around them is a more efficient path to executability than scaling independent samples.

Keywords: Graph Databases, Coder Models, Cypher, Query Generation, Knowledge Graphs, Neo4j, Inference Scaling

1 Introduction

Recent work has shown that allocating additional compute at inference time can substantially improve language model performance on complex reasoning and generation tasks Snell et al., 2024; Brown et al., 2024. Techniques such as scaling self-consistency (Wang et al., 2023) and repeated sampling (Brown et al., 2024) increase the probability of producing better outputs without modifying model parameters (Wu et al., 2024).

For an efficient scaling, a complementary line of work studies feedback-driven refinement, where models improve outputs using signals from prior attempts. Methods such as Self-Refine (Madaan et al., 2023), Reflexion (Shinn et al., 2023), and self-debugging(Chen et al., 2024) show that self-critique or execution feedback can guide models toward better generation.

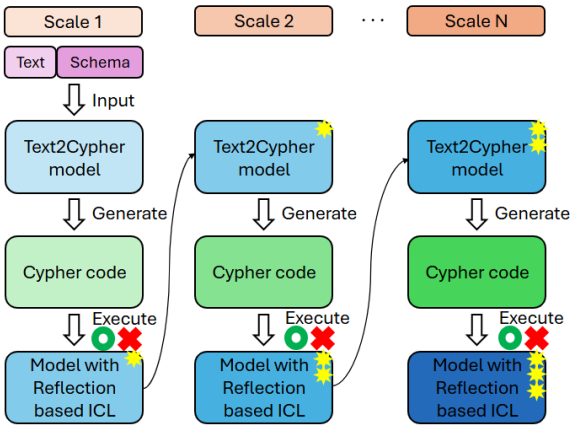

Figure 1: Our method iteratively generates queries and executes them against the database. When execution fails, the system incorporates execution feedback through reflection-based in-context learning (ICL) to refine subsequent generations. Increasing the inference scale expands the reflection context and improves the probability of producing executable queries

We study this question in structured generation, where outputs must satisfy strict syntactic and schema constraints. In query code generation, models frequently produce invalid code due to schema mismatches and compositional complexity (Gao et al., 2024; Liang et al., 2021; Hains et al., 2023). Scaling methods have been invited (Shinn et al., 2023; Chen et al., 2024) to overcome this limitation but they did not focus on the executability, which is prerequisite for the accuracy. We introduce **Reflection-Augmented Scaling (RAS)**, which conditions each new attempt on execution feedback from prior failures, and compare two inference-time strategies for Text2Cypher: **Independent Scaling (IS)**, which repeatedly samples without memory, and **RAS**.

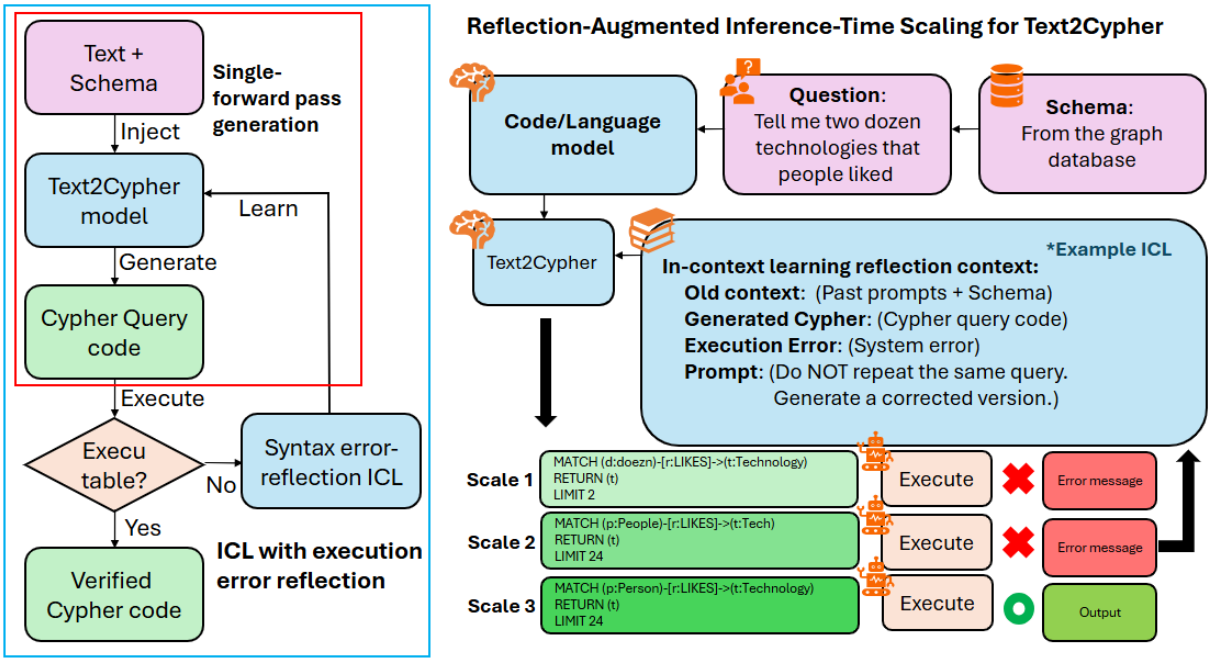

Figure 2: **Reflection-augmented inference-time scaling for Text2Cypher.** Syntax and schema errors surface to users as execution failures that break interactive query workflows, making them a distinct and user-visible failure mode. **Left:** under single-pass generation, a Cypher query is produced and executed against the graph database; execution failures trigger execution-aware reflection via in-context learning. **Right:** as the inference scale increases, the framework iteratively refines query code by conditioning on feedback from prior failures to reduce syntax errors.

Across three Neo4j datasets and five code-specialized language models, RAS consistently outperforms IS under the same compute budget. At $n = 5$, RAS achieves an 41–50% in QER, compared to 32–38% for IS. Our results show that the effectiveness of inference-time scaling depends not only on how much compute is used, but on how inference strategy is integrated.

2 Related Work

Semantic parsing for SQL. Translating natural language into executable database queries has long been studied under the framework of semantic parsing (Berant et al., 2013; Yih et al., 2016). In relational settings, the Text2SQL task has benefited from standardized benchmarks such as Spider (Yu et al., 2018), enabling systematic evaluation and rapid progress with neural and large language model–based approaches (Wang et al., 2020; Scholak et al., 2021; Gao et al., 2024; Pourreza and Rafiei, 2023). Recent work further explores the use of language models as general interfaces to structured databases through in-context learning and agentic workflows (Dong et al., 2023). Despite strong performance on relational schemas, these approaches do not transfer directly to graph query languages, which impose fundamentally different structural constraints.

Semantic parsing for Cypher. Semantic parsing for graph databases remains comparatively underexplored. Graph query languages such as Cypher require explicit reasoning over heterogeneous node and edge properties, direction-sensitive relationships, and compositional multi-hop traversal patterns. Compared with SQL, graph queries introduce additional structural constraints including path pattern matching and schema sparsity, which increase the likelihood of syntax and execution errors. Prior studies investigate natural-language access to knowledge graphs via SPARQL generation (Lan et al., 2021), and more recent work explores graph-aware retrieval and fine-tuning approaches for property graph databases (Clemedtson et al., 2025; Sivasubramaniam et al., 2024). Nevertheless, the dominant inference paradigm remains single forward-pass generation, without leveraging additional inference-time computation to recover from execution failures.

Inference-time scaling via independent resampling. An alternative to improving single-pass generation is to allocate additional inference-time computation through repeated sampling. Test-time compute scaling increases the effective reasoning

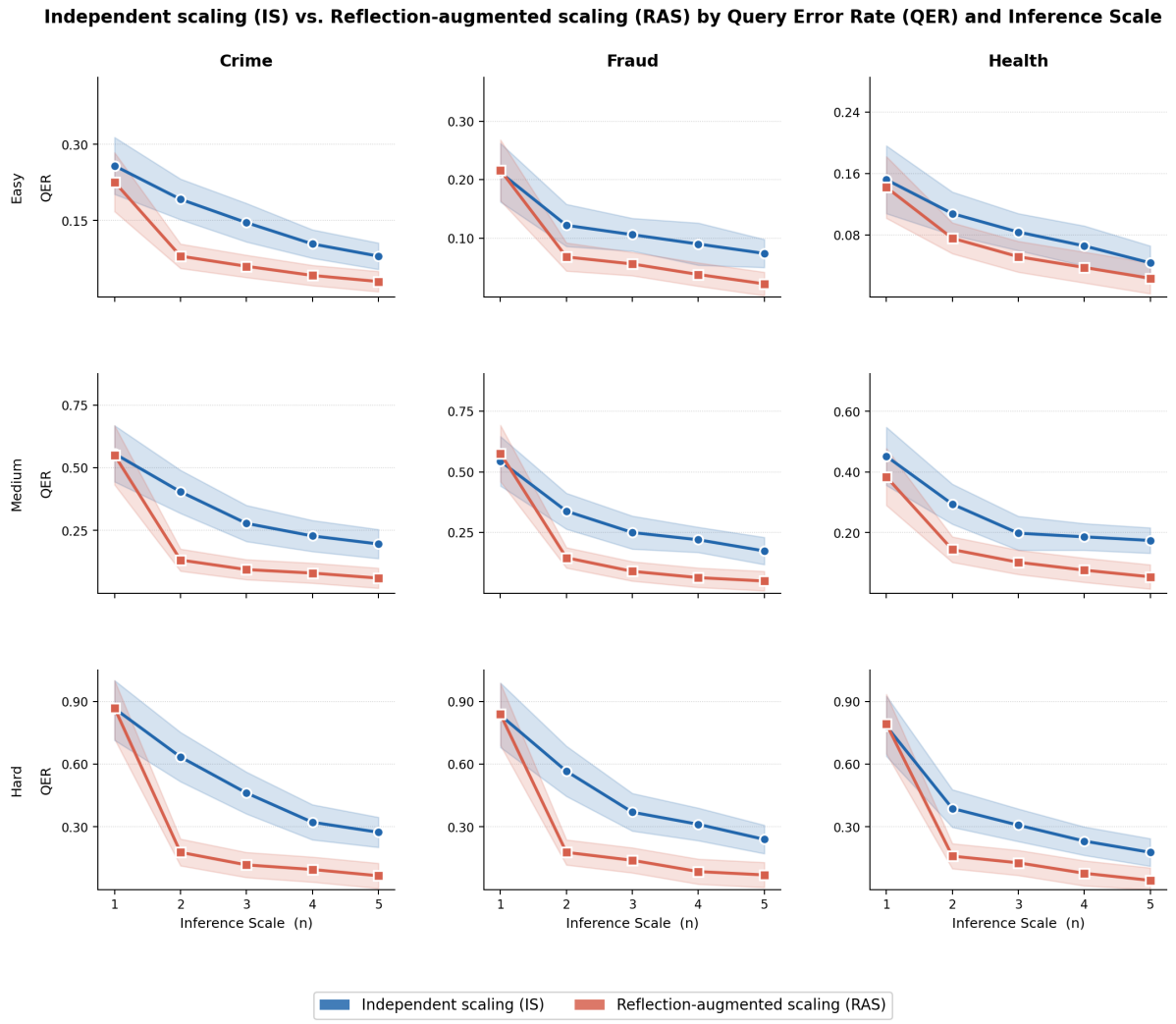

Figure 3: Query Error Rate (QER) across three graph datasets under inference-time scaling. Q@1 denotes baseline single-pass error (n=1), while IS@5 and RAS@5 report error after scaling to n=5 using simple re-run and reflection-augmented strategies. Reflection consistently achieves larger absolute reductions in QER. Values are averaged across three query complexities.

budget without modifying model parameters, and recent studies show that such strategies can yield substantial gains on complex reasoning tasks (Snell et al., 2024; Brown et al., 2024; Wu et al., 2024). Self-consistency (Wang et al., 2023) demonstrates that sampling multiple reasoning paths and selecting by majority vote consistently outperforms greedy decoding. In structured query generation, repeated sampling reduces error rates by exploiting stochastic variation across independent outputs. However, independent resampling maintains a fixed conditioning distribution over the input question and database schema, without incorporating evidence from prior execution failures. Consequently, repeated samples tend to reproduce similar structural error modes — such as schema misalignment, incorrect relationship directionality, or invalid aggregation constructs — limiting the gains achievable through scaling alone.

Execution-grounded reflection and iterative refinement. A more targeted strategy is to condition subsequent generations on feedback from earlier failures, enabling iterative correction rather than independent exploration. Self-Refine (Madaan et al., 2023) and Reflexion (Shinn et al., 2023) propose self-refinement loops in which models critique and revise their own outputs, or maintain episodic memory of unsuccessful attempts in agentic reasoning settings. Self-Debugging (Chen et al., 2024) demonstrates that program execution feedback can guide iterative repair on Text2SQL tasks, improving performance particularly on harder queries. While these methods highlight the value of feedback-driven reasoning, they have not been systematically studied in the context of schema-constrained Cypher generation against live prop-

erty graph databases. Furthermore, prior work does not directly compare independent resampling and feedback-driven refinement under controlled inference-time compute budgets. Our work addresses these gaps by formalizing execution-aware reflection for Text2Cypher generation and providing a systematic empirical comparison of both strategies across diverse property graph datasets and query complexity levels.

3 Problem Formulation and Modeling

3.1 Text2Cypher as Structured Graph Query Generation

We formulate Text2Cypher as a structured semantic parsing task that maps a natural language question to an executable Cypher query. Let $x \in \mathcal{X}$ denote a natural language input and $\mathcal{S}$ denote the database schema. The objective is to generate a query $q \in \mathcal{Q}$, where $\mathcal{Q}$ denotes the space of syntactically valid Cypher queries over a database instance G consistent with $\mathcal{S}$.

We model query generation as conditional structured generation (Scholak et al., 2021; Wang et al., 2020):

$$q \sim p_\theta(q \mid x, \mathcal{S}), \tag{1}$$

where p_θ denotes the conditional distribution induced by an autoregressive language model with parameters θ, defining a probability measure over token sequences representing candidate Cypher programs. Syntactic validity and executability are evaluated only after full decoding. Unlike unconstrained text generation, Cypher queries must satisfy strict structural constraints imposed by the graph schema; small deviations — such as incorrect relationship directionality or references to invalid node properties — result in immediate execution failure.

3.2 Executability and Accuracy

Executability and accuracy measure two conceptually distinct aspects of a generated query. *Executability* refers to whether a query can be successfully processed by the database engine without raising syntax or schema errors. *Accuracy*, in contrast, measures semantic correctness — whether the execution result matches the ground-truth answer.

A query may be executable yet semantically incorrect (e.g., due to a missing filter or an incorrect aggregation). However, a non-executable query cannot be accurate, since it fails before producing any result. This structural asymmetry motivates treating execution failure as a distinct and separable source of error.

	$A = 1$	$A = 0$
$E = 1$	P_{11}	P_{10}
$E = 0$	—	P_{00}

Table 1: Joint distribution of executability (E) and semantic accuracy (A). The cell P_{01} is undefined since a non-executable query cannot produce a correct result.

We formalize this distinction via two binary random variables defined for a generated query:

$$E \in \{0, 1\} \quad \text{(syntax)}, \qquad A \in \{0, 1\} \quad \text{(semantic)}$$

where $E = 1$ denotes a successfully executed query and $E = 0$ denotes execution failure due to syntactic or schema violations (e.g., undefined node labels, invalid relationship types, or malformed path syntax), while $A = 1$ denotes a result matching the ground-truth answer and $A = 0$ denotes a semantic error. Crucially, E captures *structural validity* and A captures *semantic fidelity*: a query may satisfy $E = 1$ while failing $A = 1$, but $E = 0$ guarantees $A = 0$ since no result is produced.

We define the joint probabilities as

$$P_{ij} = \mathbb{P}(E = i,\ A = j),$$

summarized in Table 1. Since a non-executable query cannot be accurate, P_{01} is undefined. The execution failure probability is therefore $\mathbb{P}(E = 0) = P_{00}$, and this is the signal used to trigger the reflection loop in Algorithm 2. The objective of the RAS algorithm is to decrease P_{00}, thereby increasing the total executable mass $P(E = 1) = P_{11} + P_{10}$. This paper focuses on reducing execution failure rather than directly optimizing the allocation between semantically correct executions (P_{11}) and semantically incorrect but executable queries (P_{10}).

3.3 Query Execution Error

Given a generated query q and a fixed database instance G, execution is deterministic. We write

$$(r,\ m) = \mathcal{E}(q, G), \tag{2}$$

where r is the query result returned by the database engine and m is the accompanying execution status message (e.g., a success confirmation or a structured error diagnostic). Accuracy A is evaluated against r, while executability E is determined by m: a query fails ($E = 0$) if and only if m indicates an error, in which case r is undefined.

Algorithm 1 Independent Scaling (IS)

Require: Input x, schema $\mathcal{S}$, graph G, budget T
$q_{\text{last}} \leftarrow \varnothing$
for $t = 1$ **to** T **do**
  $q_t \sim p_\theta(\cdot \mid x, \mathcal{S})$
  $q_{\text{last}} \leftarrow q_t$
  $m_t \leftarrow \mathcal{E}(q_t, G)$
  if $m_t \notin \mathcal{M}_{\text{error}}$ **then**
    return q_t
  end if
end for
return q_{last}

We define the *Query Execution Error* (QEE) indicator as

$$\text{QEE}(q, G) \;:=\; \mathbf{1}[m \in \mathcal{M}_{\text{error}}]\,, \qquad (3)$$

where $\mathcal{M}_{\text{error}}$ denotes the set of error-class status messages. Given an input distribution $\mathcal{D}$ over x, the *Query Execution Error Rate* (QER) is

$$\text{QER} \;:=\; \mathbb{E}_{x\sim\mathcal{D}}\, \mathbb{E}_{q\sim p_\theta(\cdot|x,\mathcal{S})}[\text{QEE}(q, G)]\,. \quad (4)$$

Thus QER equals $\mathbb{P}(E = 0)$ and is estimated empirically as the fraction of queries that fail execution across all inputs.

From a system perspective, executability acts as a hard constraint on downstream correctness: when E = 0, no result is produced, and the retrieval stage in RAG pipelines fails entirely. Consequently, reducing P(E = 0) is not only a syntactic objective but a prerequisite for enabling semantically grounded responses.

3.4 Inference-Time Scaling Strategies

We study two inference-time strategies under a fixed compute budget T: (i) *Independent Scaling* (IS) and (ii) *Reflection-Augmented Scaling* (RAS). Both allocate additional test-time computation without updating model parameters.

Independent Scaling (IS). IS repeatedly samples from the same conditional distribution $p_\theta(\cdot \mid x, \mathcal{S})$ until an executable query is obtained or the budget T is exhausted, as formalized in Algorithm 1. Each sample is generated independently, without incorporating information from prior execution failures(Wang et al., 2023; Brown et al., 2024). If no executable query is found within T attempts, the final sampled query is returned as a fallback.

Reflection-Augmented Scaling (RAS). RAS augments the generation context with execution feedback from prior failed attempts, enabling iterative targeted correction (Madaan et al., 2023; Shinn et al., 2023). After each execution failure, the failed query and its error message are appended to the conditioning context $\mathcal{C}$. Subsequent generations condition on this expanded context, allowing the model to identify and correct specific structural errors. Algorithm 2 formalizes this procedure. As with IS, if no executable query is produced within T attempts, the final generated query is returned as a fallback.

Algorithm 2 Reflection-Augmented Scaling (RAS)

Require: Input x, schema $\mathcal{S}$, graph G, budget T
$\mathcal{C} \leftarrow (x, \mathcal{S})$
$q_{\text{last}} \leftarrow \varnothing$
for $t = 1$ **to** T **do**
  $q_t \sim p_\theta(\cdot \mid \mathcal{C})$
  $q_{\text{last}} \leftarrow q_t$
  $m_t \leftarrow \mathcal{E}(q_t, G)$
  if $m_t \notin \mathcal{M}_{\text{error}}$ **then**
    return q_t
  end if
  $\mathcal{C} \leftarrow \mathcal{C} \cdot (q_t, m_t)$ ▷ Append failure to context
end for
return q_{last}

Dataset	Nodes	Edges	Domain
Healthcare	11,381	61,453	Clinical Records
Fraud	332,973	980,098	Financial Transactions
Crime	61,521	105,840	Public Safety

Table 2: Statistics of the Neo4j property-graph datasets used in our Text2Cypher evaluation. The datasets vary substantially in scale and structural density, enabling experiments across heterogeneous schema structures.

4 Experimental Setup

4.1 Datasets

We evaluate our approach on three publicly available Neo4j property graph databases spanning diverse scales and structural characteristics: *Healthcare*, *Fraud*, and *Crime* (Table 2). These datasets vary substantially in graph size, schema complexity, and query topology, enabling robustness assessment across heterogeneous graph structures.

Healthcare is derived from FDA adverse event reporting data and contains heterogeneous biomedical entities including drugs, patients, and reported reactions. Queries over this graph frequently require multi-entity joins and attribute-based filtering, stressing schema alignment and type consistency during generation. **Fraud** represents a large-scale synthetic financial transaction network characterized by high connectivity and long relational paths. The dense topology increases the likelihood of structural errors in generated Cypher queries,

particularly in aggregation and multi-hop traversal patterns. **Crime** follows the investigative POLE (Person–Object–Location–Event) data model used in law-enforcement analytics, exhibiting moderate scale but complex relationship directionality and path constraints.

4.2 Scope of Evaluation

This experiment evaluates execution reliability rather than full semantic correctness. Our metric, QER, measures whether a generated Cypher query can be executed successfully against the target GDB. This choice is deliberate: in graph-based RAG and agentic database systems, non-executable queries represent a hard failure mode because no evidence can be retrieved for downstream reasoning before measuring accuracy. To put it another way, executability is a necessary condition for accuracy despite it is not a sufficient condition. Although query code executability is not a substitute for semantic accuracy, it is a required precondition.

4.3 Question Generation

For each dataset, natural language questions are generated at three complexity levels (*Easy*, *Medium*, and *Hard*), conditioned on the graph schema and representative subgraph samples. Full prompt templates and representative examples are provided in the Appendix.

4.4 Text2Cypher

We implement Text2Cypher using LangChain's `GraphCypherQAChain`, which constructs a structured generation prompt by injecting the database schema $\mathcal{S}$ retrieved directly from the Neo4j instance. To isolate the query generation stage and focus directly on QEE, we do not evaluate downstream answer for accuracy. Each generated query is executed against the live database instance to determine executability as defined in §3.2.

4.5 Models

We evaluate five open-weight, code-specialized language models: `CodeLlama-7B` and `CodeLlama-13B` (Rozière et al., 2023), `DeepSeek-Coder-6.7B` (Guo et al., 2024), `Qwen2.5-Coder-7B` (Hui et al., 2024), and `StarCoder2-7B` (Lozhkov et al., 2024). These models are pretrained on large-scale source code corpora, inducing a structural prior toward syntactically well-formed token sequences. We deliberately focus on small language models (SLMs) to reduce the latency of the inference stage. All models are evaluated under identical schema injection, decoding configuration, and execution-validation settings within the same `GraphCypherQAChain` pipeline.

4.6 Scaled Context Construction via In-Context Learning

We instantiate the two inference methods (Algorithms 1 and 2) at scale. For our method, Alg. 2, the initial context $\mathcal{C}_0 = (x, \mathcal{S})$ is constructed by injecting the schema $\mathcal{S}$ directly from the Neo4j instance into a structured Cypher generation prompt. For the RAS, each failed attempt appends the generated query q_t and its execution error m_t to the prompt, along with a corrective instruction, forming $\mathcal{C}_t = \mathcal{C}_{t-1} \cup \{q_t, m_t\}$. This accumulated context is embedded into the prompt, preserving compatibility with the system architecture. Alg. 1 keeps generating answers until it succeeds. At T=1, both strategies are based on the identical single-pass baseline, but the scaling with RAS keeps adding query code with system errors.

4.7 Hyperparameters

Each experimental configuration — defined by a dataset, model, complexity level, and inference strategy — is evaluated over 128 independent runs. We report the mean QER along with its standard deviation σ_{QER} across runs. All decoding is performed at temperature $\tau = 0.9$ to introduce sufficient stochastic diversity across trials while maintaining generation quality (Wang et al., 2023). The inference scale is fixed at $T = 5$, selected via multi-objective optimization (MOO) on the trade-off between error reduction, latency and compute cost with the optimality identified at the knee point of Pareto frontiers (Jung et al., 2025).

Evaluation is based on the query execution that has been introduced in Table 3.2. Additional information about the prompts by complexity and results by complexity are provided in Appendix A.1 and Appendix A.2.

5 Results

5.1 Inference Scaling Consistently Reduces QER

Table 3 reports QER under single-shot generation (n=1) and scaled inference (n=5) for both IS and RAS. Across all datasets and models, increasing the inference scale reduces QER regardless of the

Model	Crime					Fraud					Healthcare				
	Q@1	IS@5	RAS@5	ΔIS	ΔRAS	Q@1	IS@5	RAS@5	ΔIS	ΔRAS	Q@1	IS@5	RAS@5	ΔIS	ΔRAS
CodeLlama-13B	0.40	0.14	0.05	0.26	**0.35**	0.38	0.12	0.03	0.26	**0.35**	0.32	0.08	0.03	0.24	**0.29**
DeepSeek-6.7B	0.46	0.18	0.04	0.28	**0.42**	0.44	0.13	0.05	0.31	**0.39**	0.37	0.13	0.04	0.24	**0.33**
CodeLlama-7B	0.58	0.19	0.05	0.39	**0.53**	0.56	0.17	0.06	0.39	**0.50**	0.43	0.13	0.05	0.30	**0.38**
Qwen2.5-7B	0.65	0.21	0.06	0.44	**0.59**	0.63	0.19	0.04	0.44	**0.59**	0.51	0.14	0.04	0.37	**0.47**
StarCoder2-7B	0.68	0.20	0.06	0.48	**0.62**	0.69	0.20	0.05	0.49	**0.64**	0.63	0.17	0.05	0.46	**0.58**

Table 3: Query Execution Error Rate (QER) across three graph datasets and five coder models. Q@1 denotes baseline error at a single generation (n=1), computed as the average of IS@1 and RAS@1 across Easy, Medium, and Hard complexity levels. IS@5 and RAS@5 denote error after test-time scaling to n=5 using IS (Simple Retrials) and RAS (Reflection-Augmented Scaling). ΔIS and ΔRAS represent reductions in QER relative to Q@1. The average values are rounded up into two decimal points. Detailed experimenta result is in Appendix 4

inference methods. The Q@1 is the average of the two methods, and both showed a similar error rate at the initial stage because the ICL of RAS is not applicable at RAS@1.

5.2 RAS Outperforms Independent Scaling

At scale n=1, both strategies shows similar QER, as no execution feedback has yet been incorporated. However, at n=5, RAS achieves QER of 3–6% across most configurations, representing a reduction of more than 10% over IS. This gap demonstrates that execution feedback constitutes a qualitatively stronger inference-time signal than independent resampling. Rather than simply retrying query generations, RAS actively incorporates information from prior execution failures to steer subsequent generation toward executable queries. The advantage of RAS over IS is consistent across all datasets, model architectures, and input complexities.

5.3 QER Scaling Trajectories Differ Between Strategies

Beyond endpoint comparisons, Figure 3 reveals a difference in how the two strategies reduce QER across the inference budget. IS exhibits an approximately linear decrease in QER as the scale grows from 1 to 5, suggesting that each additional independent trial contributes a roughly constant marginal gain in executability. RAS, by contrast, follows a concave trajectory: the sharpest drop occurs at n=2, the first point at which execution feedback from a failed attempt becomes available, after which further improvements taper gradually. This suggests that the primary benefit of execution-grounded reflection is concentrated in the earliest feedback step, while subsequent iterations yield diminishing returns.

5.4 RAS is Robust Across Datasets, Models, and Query Complexities

A critical requirement for any inference-time strategy is that its gains generalize beyond a narrow set of conditions. Table 3 demonstrates that RAS show a greater QER decrease consistently across datasets and models. Across all three datasets — Healthcare, Fraud, and Crime — RAS reduces QER to 0.03–0.06 at n=5, despite differences are observed by graph topology, schema complexity, and domain vocabulary. Robustness also holds across model architectures. RAS achieves consistent gains over all five code-specialized models spanning. This uniformity suggests that the feedback mechanism is not dependent on any particular model's prior capacity for schema-aligned generation. Further, the standard deviation of QER (σ_{QER}) and consistency across input complexities reported in Table 4 provides further evidence of the robustness. Crucially, these values are consistent across model architectures and datasets with no outlying configurations, indicating that RAS produces reliably low error rates across independent runs rather than occasional successes masking high variance. The concrete failure signal incorporated at each reflection step guides generation toward a stable, executable region of the query space, constraining the stochastic variation inherent in autoregressive decoding.

6 Conclusion

This paper studies inference-time scaling for Text2Cypher generation by formalizing and comparing two strategies: Independent Scaling (IS) and Reflection-Augmented Scaling (RAS). We evaluate both approaches under a fixed compute budget across three Neo4j property graph datasets, five code-specialized language models, and multiple query complexity levels.

Our results reveal that IS and RAS exhibit fun-

damentally different scaling behaviors. IS reduces the Query Execution Error Rate (QER) in an approximately linear manner, where each additional sample provides a consistent marginal improvement through stochastic diversity. In contrast, RAS follows a concave trajectory, achieving its largest reduction at n=2, when execution feedback first becomes available, followed by diminishing returns. This two-phase behavior highlights a key asymmetry: the primary benefit of reflection is concentrated in the earliest feedback-driven correction step rather than in prolonged iterative refinement.

Empirically, RAS consistently outperforms IS under identical compute budgets, achieving substantially lower QER across all datasets and models. Notably, no IS configuration matches the performance of RAS at the same inference scale. These findings demonstrate that the structure of inference-time computation—specifically, whether it incorporates feedback—can be more critical than increasing model size or the number of independent samples.

Overall, our results suggest that effective inference-time scaling for structured generation requires moving beyond independent resampling toward feedback-driven strategies. Execution feedback provides a structured and informative signal that transforms the correction process from stochastic exploration into targeted refinement. Understanding how to optimally allocate compute across such feedback mechanisms has direct implications for deploying reliable and cost-efficient language model systems in graph query generation.

Although our experiments focus on Text2Cypher, the proposed framework is not limited to graph query languages. The same execution-grounded reflection mechanism can be extended to other structured semantic parsing tasks, such as Text2SQL, where outputs must satisfy strict syntactic and schema constraints. More broadly, our findings point toward a general paradigm in which verifiable feedback signals can improve inference-time scaling in program synthesis and agentic systems.

7 Limitations

This work has several limitations. First, our evaluation is restricted to Text2Cypher on Neo4j property graphs; inference scaling dynamics may differ for other semantic parsing tasks (e.g., Text2SQL) or database engines with distinct execution semantics.

Second, we cap the inference budget at n=5, reflecting a practical performance–latency trade-off; larger budgets may yield diminishing returns, higher latency, or over-correction from compounding feedback.

Third, we focus on open-weight, code-specialized models for reproducibility, leaving the scaling behavior of larger proprietary models as an open question.

Fourth, we evaluate performance primarily through query executability. Although executability does not guarantee semantic correctness, we treat it as a prerequisite reliability condition: in graph-based RAG and agentic systems, a non-executable query halts evidence retrieval entirely, causing the system to fail before any answer can be grounded in data. Reducing execution errors is therefore a necessary first step toward reliable downstream reasoning, and future work should extend this framework to jointly optimize executability and semantic fidelity.

Finally, our reliance on in-context learning to incorporate execution feedback introduces token and latency overhead as the reflection context grows. Execution traces are also used transiently rather than persisted; aggregating these logs could enable system-level learning across queries beyond single-session inference.

References

Jonathan Berant, Andrew Chou, Roy Frostig, and Percy Liang. 2013. Semantic parsing on Freebase from question-answer pairs. In *Proceedings of the 2013 Conference on Empirical Methods in Natural Language Processing*, pages 1533–1544.

Bradley Brown, Jordan Juravsky, Ryan Ehrlich, Ronald Clark, Quoc V. Le, Christopher Ré, and Azalia Mirhoseini. 2024. Large language monkeys: Scaling inference compute with repeated sampling. *arXiv preprint arXiv:2407.21787*.

Xinyun Chen, Maxwell Lin, Nathanael Schärli, and Denny Zhou. 2024. Teaching large language models to self-debug. In *The Twelfth International Conference on Learning Representations*.

Gustav Clemedtson et al. 2025. GraphRAFT: Retrieval augmented fine-tuning for knowledge graphs on graph databases. *arXiv preprint arXiv:2504.05478*.

Xuemei Dong, Chao Zhang, Yuhang Ge, Yuren Mao, Yunjun Gao, Jinshu Lin, Dongfang Lou, et al. 2023. C3: Zero-shot text-to-sql with chatgpt. *arXiv preprint arXiv:2307.07306*.

Dawei Gao, Haibin Wang, Yaliang Li, Xiuyu Sun, Yichen Qian, Bolin Ding, and Jingren Zhou. 2024.

Text-to-sql empowered by large language models: A benchmark evaluation. *Proceedings of the VLDB Endowment*, 17(5):1132–1145.

Daya Guo, Qihao Zhu, Dejian Yang, Zhenda Xie, Kai Dong, Wentao Zhang, Guanting Chen, Xiao Bi, Y. Wu, Y. K. Li, Fuli Luo, Yingfang Xiong, and Wen Liang. 2024. Deepseek-coder: When the large language model meets programming – the rise of code intelligence. *arXiv preprint arXiv:2401.14196*.

Gaétan J. D. R. Hains, Youry Khmelevsky, and Thibaut Tachon. 2023. From natural language to graph queries. In *2023 IEEE 19th International Conference on Software Engineering Research, Management and Applications (SERA)*, pages 1–6.

Binyuan Hui, Jian Yang, Zeyu Cui, Jiaxi Yang, Dayiheng Liu, Lei Zhang, Tianyu Liu, Jiajun Zhang, Bowen Yu, Kai Dang, et al. 2024. Qwen2.5-coder technical report. *arXiv preprint arXiv:2409.12186*.

M. Jung, A. Ricky, and M. R. Chatni. 2025. 3d optimization for ai inference scaling: Balancing accuracy, cost, and latency. *arXiv preprint arXiv:2510.18905*.

Yunshi Lan, Gaole He, Jinhao Jiang, Jing Jiang, Wayne Xin Zhao, and Ji-Rong Wen. 2021. A survey on complex knowledge base question answering: Methods, challenges and solutions. *arXiv preprint arXiv:2105.11644*.

Shiqi Liang, Kurt Stockinger, Tarcisio Mendes de Farias, Maria Anisimova, and Manuel Gil. 2021. Querying knowledge graphs in natural language. *Journal of Big Data*, 8(1):3.

Anton Lozhkov, Raymond Li, Loubna Ben Allal, Federico Cassano, Joel Lamy-Poirier, Nouamane Tazi, Ao Tang, Vsevolod Pyatkin, et al. 2024. Starcoder 2 and the stack v2: The next generation. In *The Twelfth International Conference on Learning Representations*.

Aman Madaan, Niket Tandon, Prakhar Gupta, Skyler Hallinan, Luyu Gao, Sarah Wiegreffe, Uri Alon, Nouha Dziri, Shrimai Prabhumoye, Yiming Yang, Shashank Gupta, Bodhisattwa Prasad Majumder, Katherine Hermann, Sean Welleck, Amir Yazdanbakhsh, and Peter Clark. 2023. Self-refine: Iterative refinement with self-feedback. In *Advances in Neural Information Processing Systems*, volume 36.

Mohammadreza Pourreza and Davood Rafiei. 2023. Din-sql: Decomposed in-context learning of text-to-sql with self-correction. In *Advances in Neural Information Processing Systems*, volume 36.

Baptiste Rozière, Jonas Gehring, Fabian Gloeckle, Sten Sootla, Itai Gat, Xiaoqing Ellen Tan, Yossi Adi, Jingyu Liu, Romain Sauvestre, Tal Remez, Jérémy Rapin, Artyom Kozhevnikov, Ivan Evtimov, Joanna Bitton, Manish Bhatt, Cristian Canton Ferrer, Aaron Grattafiori, Wenhan Xiong, Alexandre Défossez, Jade Copet, Faisal Azhar, Hugo Touvron, Louis Martin, Nicolas Usunier, Thomas Scialom, and Gabriel Synnaeve. 2023. Code Llama: Open foundation models for code. *arXiv preprint arXiv:2308.12950*.

Torsten Scholak, Nathan Schucher, and Dzmitry Bahdanau. 2021. PICARD: Parsing incrementally for constrained auto-regressive decoding from language models. In *Proceedings of the 2021 Conference on Empirical Methods in Natural Language Processing*, pages 9895–9901.

Noah Shinn, Federico Cassano, Ashwin Gopinath, Karthik R. Narasimhan, and Shunyu Yao. 2023. Reflexion: Language agents with verbal reinforcement learning. In *Advances in Neural Information Processing Systems*, volume 36.

Murugesan Sivasubramaniam et al. 2024. SM3-Text-to-Query: Supervising, scaling and synthesizing data for few-shot text-to-query. *arXiv preprint arXiv:2411.05521*.

Charlie Snell, Jaehoon Lee, Kelvin Xu, and Aviral Kumar. 2024. Scaling llm test-time compute optimally can be more effective than scaling model parameters. *arXiv preprint arXiv:2408.03314*.

Bailin Wang, Richard Shin, Xiaodong Liu, Oleksandr Polozov, and Matthew Richardson. 2020. RAT-SQL: Relation-aware schema encoding and linking for text-to-sql parsers. In *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics*, pages 7567–7578.

Xuezhi Wang, Jason Wei, Dale Schuurmans, Quoc Le, Ed Chi, Sharan Narang, Aakanksha Chowdhery, and Denny Zhou. 2023. Self-consistency improves chain of thought reasoning in language models. In *The Eleventh International Conference on Learning Representations*.

Yixuan Wu et al. 2024. Inference scaling laws: An empirical analysis of compute-optimal inference for problem-solving with language models. *arXiv preprint arXiv:2408.00724*.

Wen-tau Yih, Matthew Richardson, Christopher Meek, Ming-Wei Chang, and Jina Suh. 2016. The value of semantic parse labeling for knowledge base question answering. In *Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics*, pages 201–206.

Tao Yu, Rui Zhang, Kai Yang, Michihiro Yasunaga, Dongxu Wang, Zifan Li, James Ma, Irene Li, Qingning Yao, Shanelle Roman, Zilin Zhang, and Dragomir Radev. 2018. Spider: A large-scale human-labeled dataset for complex and cross-domain semantic parsing and text-to-sql task. In *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing*, pages 3911–3921.

A Appendix

A.1 Experimental Results

Table 4 provides a granular breakdown of the Query Execution Error Rate (QER) across all evaluated

datasets, models, and query complexities. As expected, single-pass generation (QER@1) degrades significantly as query difficulty increases, with execution failures climbing as high as 0.97 on Hard queries. While scaling to n=5 via Independent Scaling (IS@5) yields moderate improvements, Reflection-Augmented Scaling (RAS@5) consistently suppresses the error rate to below 0.10, even on the most topologically complex queries. Furthermore, the reported standard deviations (σ_{IS} and σ_{RAS}) reveal that RAS not only reduces the mean error but also tightens the variance across independent runs—particularly at the Hard complexity level. This indicates that execution-grounded reflection effectively constrains the generation space, yielding vastly more stable and predictable behavior than simple independent resampling.

A.2 Input text complexities

Table 5 provides a representative sample of the natural language queries used in our evaluation, categorized by dataset and complexity tier. To provide a concrete understanding of the structural challenges introduced at each level, we detail the specific input prompts alongside their corresponding evaluation identifiers. As illustrated in the table, *Easy* queries (e.g., `crime_easy_1`) focus on straightforward entity retrieval and direct attribute filtering. *Medium* queries (e.g., `health_medium_1`) introduce relational constraints, requiring the model to successfully join multiple entity types. Finally, *Hard* queries (e.g., `fraud_hard_1`) demand advanced structural reasoning, requiring the generated Cypher code to handle cyclic multi-hop paths, mutual relationships, and conditional aggregations simultaneously. These examples highlight the escalating topological complexity that the inference strategies must navigate.

B Broader Applications: Reflection as a Generalized Reasoning Engine

While this study focuses on Text2Cypher as a rigorous testbed for execution-grounded scaling, the underlying reflection-augmented in-context learning (ICL) mechanism is fundamentally domain-agnostic. Beyond structured query generation, this iterative feedback loop can serve as the core component of a generalized advanced reasoning engine.

In standard language model deployments, complex reasoning is often bottlenecked by a single-forward pass. By treating environmental feedback—such as database execution errors, compiler outputs, or external API responses—as structured diagnostic signals, the Reflection-Augmented Scaling (RAS) methodology effectively transforms a static generation task into an agentic, self-correcting workflow.

As a generalized reasoning engine, this architecture can be extended to multi-turn analytical tasks where a model must autonomously navigate knowledge graphs, verify its own intermediate logical steps, and recover from hallucinations before returning a final answer to the user. Consequently, understanding the scaling trajectory of reflection (e.g., the sharp performance gains at n=2) informs the compute-optimal design of these broader agentic systems, ensuring that inference-time compute is allocated efficiently during complex, multi-step reasoning tasks.

The full production implementation is not publicly released because it is part of the corporation's internal reasoning-engine. To support reproducibility, we describe the inference procedure, prompt construction, execution-validation protocol, datasets, model choices, decoding settings, and evaluation metrics in detail. We also provide pseudocode for both Independent Scaling and Reflection-Augmented Scaling, enabling independent reimplementation of the experimental logic without exposing proprietary system components.

Model	Cmplx.	QER@1	IS@5	RAS@5	σ_{IS}	σ_{RAS}
		CRIME				
CodeLlama-13B	Easy	0.17	0.04	0.02	0.02	0.02
	Medium	0.34	0.17	0.05	0.04	0.04
	Hard	0.70	0.22	0.07	0.06	0.06
DeepSeek-Coder-6.7B	Easy	0.21	0.06	0.02	0.03	0.02
	Medium	0.35	0.24	0.05	0.06	0.04
	Hard	0.83	0.23	0.06	0.07	0.05
CodeLlama-7B	Easy	0.21	0.08	0.04	0.03	0.02
	Medium	0.63	0.16	0.06	0.06	0.04
	Hard	0.91	0.32	0.06	0.08	0.04
Qwen2.5-Coder-7B	Easy	0.30	0.09	0.03	0.02	0.02
	Medium	0.70	0.19	0.07	0.06	0.04
	Hard	0.96	0.35	0.07	0.07	0.06
StarCoder2-7B	Easy	0.34	0.13	0.04	0.03	0.02
	Medium	0.76	0.22	0.07	0.07	0.04
	Hard	0.95	0.25	0.07	0.08	0.07
		FRAUD				
CodeLlama-13B	Easy	0.12	0.05	0.01	0.02	0.02
	Medium	0.33	0.14	0.03	0.04	0.04
	Hard	0.69	0.18	0.04	0.06	0.07
DeepSeek-Coder-6.7B	Easy	0.14	0.05	0.02	0.03	0.02
	Medium	0.44	0.15	0.06	0.05	0.04
	Hard	0.75	0.18	0.08	0.06	0.06
CodeLlama-7B	Easy	0.18	0.09	0.03	0.02	0.02
	Medium	0.63	0.19	0.06	0.06	0.04
	Hard	0.87	0.24	0.09	0.07	0.06
Qwen2.5-Coder-7B	Easy	0.28	0.07	0.03	0.02	0.02
	Medium	0.68	0.18	0.04	0.07	0.04
	Hard	0.92	0.32	0.06	0.06	0.06
StarCoder2-7B	Easy	0.36	0.11	0.02	0.03	0.02
	Medium	0.73	0.21	0.06	0.06	0.04
	Hard	0.97	0.28	0.08	0.09	0.08
		HEALTH				
CodeLlama-13B	Easy	0.08	0.01	0.01	0.02	0.02
	Medium	0.24	0.13	0.04	0.04	0.04
	Hard	0.63	0.11	0.05	0.06	0.05
DeepSeek-Coder-6.7B	Easy	0.08	0.03	0.02	0.02	0.02
	Medium	0.32	0.15	0.06	0.04	0.03
	Hard	0.71	0.22	0.03	0.07	0.06
CodeLlama-7B	Easy	0.13	0.03	0.02	0.02	0.02
	Medium	0.38	0.17	0.06	0.05	0.04
	Hard	0.79	0.19	0.06	0.07	0.04
Qwen2.5-Coder-7B	Easy	0.20	0.07	0.03	0.03	0.02
	Medium	0.47	0.19	0.05	0.05	0.04
	Hard	0.86	0.16	0.03	0.06	0.06
StarCoder2-7B	Easy	0.25	0.08	0.04	0.03	0.02
	Medium	0.69	0.23	0.06	0.04	0.04
	Hard	0.95	0.21	0.05	0.07	0.06

Table 4: QER per dataset at n=1 (QER@1, averaged over IS and RAS baselines) and n=5 (IS@5, RAS@5). σ_{IS} and σ_{RAS} denote the standard deviation of QER at n=5 under IS and RAS, respectively.

Query ID	Complexity	Natural Language Question
Crime Dataset		
`crime_easy`	Easy	Find all crime incidents and return their incident IDs and crime types.
`crime_medium`	Medium	Find people involved in crime incidents at a specific location and return the person name and incident ID.
`crime_hard`	Hard	Identify repeat offenders who are connected to more than one crime incident and return the person name and the number of incidents.
Fraud Dataset		
`fraud_easy`	Easy	List all transactions that are marked as fraudulent and return their transaction IDs.
`fraud_medium`	Medium	Find customers who have more than two fraudulent transactions and return the customer name and the number of fraudulent transactions.
`fraud_hard`	Hard	Find pairs of accounts that have transferred money between each other and are both involved in at least one fraudulent transaction, and return the account IDs and the number of shared fraudulent transactions.
Healthcare Dataset		
`health_easy`	Easy	Find all patients in the database and return their names.
`health_medium`	Medium	List all medical records for a specific patient and the physicians associated with those records.
`health_hard`	Hard	Find patients who have been treated by more than one physician for the same medical condition and return the condition and physician count.

Table 5: Detailed natural language questions across the Crime, Fraud, and Healthcare datasets, including their assigned evaluation identifiers and complexity tiers.