# NGDBench: Towards Neural Graph Data Management

Anonymous ACL submission


## Abstract

Data critical to real-world decision-making is increasingly found within organizations. Such data is heterogeneous, constantly evolving, and only imperfectly captured. However, current data management systems remain largely passive, retrieving what is explicitly stored while offering limited support for uncovering implicit structure or reasoning under noise, incompleteness, and continuous updates. We argue that next-generation data management requires neural capabilities, which can uncover complex latent relationships, distinguish reliable signals from noise, and remain consistent as the underlying data state evolves. To support this direction, we introduce NGDBENCH, a benchmark across five domains that unifies structured and unstructured sources. NGDBENCH adopts a graph view because graphs provide a flexible abstraction for modeling complex systems, capturing latent relationships, and subsuming structured formats such as relational tables. Each instance pairs a clean latent graph with a realistically perturbed observed graph. NGDBENCH supports full Cypher queries and dynamic data management operations. Evaluations of state-of-the-art Text-to-Cypher by LLMs and GraphRAG pipelines reveal that current neural query methods remain sensitive to noise and struggle with dynamic state tracking, highlighting the need for resilient, inference-capable data management. Our code is available at https://anonymous.4open.science/r/NGDBench-2E77.


## 1 Introduction

As high-quality internet data become increasingly scarce(Villalobos et al., 2024), critical decision-making data increasingly reside within organizations, such as enterprises and financial institutions(Kayali et al., 2025). These repositories, spanning relational, graph, and enterprise systems, are heterogeneous, continuously evolving, often noisy and with implicit information, like hidden entities and latent relation which are lost in the process of recording (Abaskohi et al., 2026). There are two dominant ways to access these data: Traditional databases support precise symbolic queries but rely on explicitly defined schemas and stored facts. Vector databases retrieve semantically similar items in embedding space, but they are not designed to perform deeper data analysis beyond similarity matching, such as complex logical queries and reasoning over implicit relations. AI agents access data through interfaces such as RAG, Text-to-SQL(Yu et al., 2018), or Text-to-Cypher(Ozsoy et al., 2024), which similarly expose observed facts rather than latent implications. As a result, the current data stack remains largely passive: it retrieves what is stored, but rarely infers what is implied.

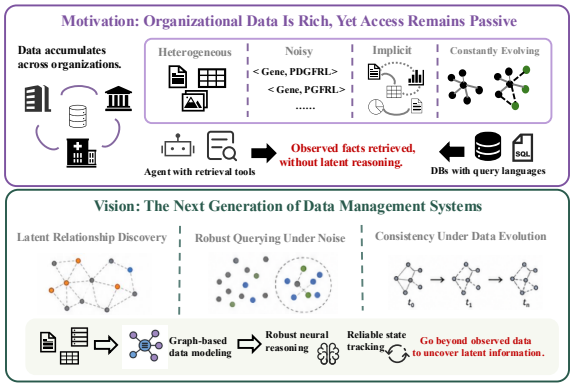


Figure 1: Motivation and Vision: Overcoming the limitations of passive data access through robust graph-based neural data reasoning.

Consider an AI agent performing financial risk analysis on a graph database. Querying only observed records via Text-to-Cypher over noisy or incomplete data fails to uncover latent fraud rings and hidden relations. This gap between observed data and the real-world state fundamentally hinders reliable AI understanding to the real environment.

Table 1: Comparison of Recent Neural Graph Reasoning Benchmarks. Existing datasets focus on static reasoning with limited logic (FOL/EPFO), whereas our NGDB-Dataset supports full database capabilities including Cypher queries, aggregations, and dynamic updates.

| Benchmark / Dataset | Data Format | Operators | Multi-variable Query | Aggregation Support | Dynamic Updates | General Graph Query (e.g., Cypher) |
|---|---|---|---|---|---|---|
| Q2B (Ren et al., 2020) | Id-based Triples | EPFO (AND, OR, $\exists$) | No | No | No | No |
| LitCQD (Demir et al., 2023) | Id-based Triples & Numbers | EPFO + Numeric Operators | No | Yes | No | No |
| NRN (Bai et al., 2023) | Id-based Triples & Numbers | FOL + Numeric Operators | No | No | No | No |
| $EFO_1$-CQA (Wang et al.) | Id-based Triples | FOL (AND, OR, NOT $\exists$) | No | No | No | No |
| $EFO_k$-CQA (Yin et al., 2025) | Id-based Triples | FOL (AND, OR, NOT $\exists$) | Yes | No | No | No |
| **NGDBENCH (Ours)** | **Ids, Numbers, and Strings** | **Cypher Operators** | **Yes** | **Yes** | **Yes** | **Yes** |

We therefore argue for a new form of data management system behind AI applications: a *neural database*. Unlike conventional databases that rely on symbolic matching or similarity search over stored records, a neural database should combine precise query execution ablity with neural inference—uncovering latent relationships, distinguishing reliable signals from noise and reliably tracking data state as data evolves.

Graph-structured data naturally supports this vision. Graphs flexibly capture entities, attributes, relations, and multi-hop dependencies (Reddit User, 2023; Neo4j, n.d.), expose latent structure from unstructured sources (Lai et al., 2025), and can subsume relational tables (Kim et al., 2024). As latent-world reasoning is fundamentally relational, graphs offer a suitable substrate for combining symbolic data management with neural inference.

Existing systems move only partway toward this goal. Neural Graph Database connects graph databases with GNN-based prediction (Besta et al., 2022), while neural complex query answering methods reason over incomplete graphs through logical query embeddings (Ren et al., 2023). Yet they are mostly restricted to Existential First-Order logic rather than full database queries (Wang et al.; Yin et al., 2025), assume the observed graph as the main source of truth despite invalid observed facts and missing valid facts (Huang et al., 2025; Drummond and Shearer, 2006), and provide limited support for state-changing operations such as insertion, deletion, and update. Meanwhile, existing mainstream graph datasets (Hu et al., 2020) are often homogeneous, modality-specific, or static. As shown in Table 1, they do not sufficiently test whether systems can execute full database-style queries over heterogeneous, noisy, incomplete, and dynamically evolving graph data.

To support further development in this direction, we introduce NGDBENCH, a benchmark for graph-based neural data management. NGDBENCH spans five domains, including financial, social, biomedical, AI-agent tool-use, and enterprise reporting scenarios, integrating structured records with unstructured sources under a unified Labeled Property Graph schema. Each instance pairs a clean *latent* graph with a realistically perturbed *observed* graph, covering structural and attribute-level imperfections such as missing edges, spurious connections, schema inconsistencies, and corrupted entity descriptions.

Unlike prior benchmarks focused on elementary logical operations, NGDBENCH supports the **full Cypher query language**, covering complex pattern matching, variable-length paths, numerical aggregations, and dynamic operations such as `CREATE`, `UPDATE`, and `DELETE`. We evaluate NGDBENCH on two widely used neural query paradigms, Text-to-Cypher and LLM-based GraphRAG, which perform graph reasoning by either translating language into symbolic queries or retrieving graph-grounded evidence. Although more promising than purely symbolic matching for handling noise and latent associations, they remain vulnerable to noisy observations and struggle with analytical reasoning and dynamic state tracking.

Our contributions are summarized as follows: (1) **Neural data management vision.** We argue for moving beyond passive record retrieval toward neural inference over latent graph structure, enabling robust querying under noise and consistency under updates. (2) **Neural graph data management benchmark.** We introduce NGDBENCH, a five-domain benchmark that unifies structured and unstructured sources, pairs clean latent graphs with perturbed observations, and supports full Cypher queries with dynamic operations. (3) **Evaluation of current paradigms.** We evaluate state-of-the-art Text-to-Cypher and GraphRAG methods, revealing limitations in noise robustness, analytical reasoning, and dynamic state tracking.

## 2 Related Work

**Neural Graph Database.** Existing definitions of Neural Graph Databases (NGDBs) generally follow two paradigms. One extends traditional graph databases by preserving symbolic storage, such as Labeled Property Graphs, while using encoders like LPG2vec to project graph topology into embeddings (Besta et al., 2022). The other treats NGDBs as latent-space data management systems, with neural storage and query engines operating over learned representations (Ren et al., 2023). Recent extensions further explore agentic (Bai et al., 2025b) and privacy-preserving (Hu et al., 2024) NGDB architectures. However, existing evaluations provide limited coverage of realistic noise, expressive queries and continuous updates.

**Complex Logical Query Answering**. Complex query answering is important for NGDBs, especially under incomplete data. Prior work studies this problem through geometric query embeddings (Hamilton et al., 2018; Ren et al., 2020; Zhang et al., 2021), logic-based reasoning (Arakelyan et al., 2021; Chen et al., 2022), inductive methods (Galkin et al., 2022; Zhu et al., 2022), and recent extensions to implicit constraints and abductive reasoning (Gao et al., 2025b,a). Yet most remain limited to Existential First-Order logic, falling short of realistic database workloads with variable-length paths, aggregation, and data management operations.

**Existing Datasets and Limitations**. Existing benchmarks mainly target static knowledge-graph reasoning or predictive learning over relational databases. KG reasoning benchmarks, including LS-CQA (Bai et al., 2024), LitCQD (Demir et al., 2023), and EFO$k$-CQA (Yin et al., 2025), support attributes or richer logical queries but remain largely static and homogeneous. Relational benchmarks, such as RelBench v2 (Gu et al., 2026) and 4DBInfer (Wang et al., 2024), focus on structured prediction over heterogeneous relational data. They therefore fall short in evaluating expressive graph query execution, noise robustness, and dynamic data management.

## 3 NGDBench

We introduce NGDBENCH, where each dataset therefore contains a clean graph representing latent ground truth, a corresponding perturbed graph representing observed data, and a collection of evaluation queries and management operations.

As illustrated in Fig. 2, our framework consists of three primary components. The **Data Collection & Unification** module gathers representative structured and unstructured datasets, converting them into a unified graph representation. The **Perturbation Generator** module introduces controlled perturbations into the original graphs according to predefined specifications, thereby generating the observed graphs. Then, the **Query Generator** module constructs a *Query Template Library* to sample both analytical and management-oriented queries. Lastly, comprehensive evaluation are implemented across robust analytical query to data management tasks.

### 3.1 Data Preparation

We construct NGDBENCH from five domains: social systems, finance, biomedicine, agent tool use, and enterprise reporting. These domains cover both structured repositories and unstructured documents, reflecting realistic settings where data is heterogeneous, noisy, and only partially explicit. We prefix each dataset with *NGD* to distinguish the processed benchmark graphs from their raw sources. Table 2 summarizes the dataset statistics.

#### 3.1.1 Structured Domains

We first select three structured sources with diverse schemas, relation types, and query patterns. For each source, we convert the original data into a Labeled Property Graph while preserving its core semantics.

**NGD-BI.** Built on the LDBC-BI benchmark (Szárnyas et al., 2022), NGD-BI captures a complex social-network setting similar to those appearing in realistic web-agent environments (Zhou et al., 2024). It contains rich many-to-many relations, dense connectivity, and attribute-structure correlations, making it suitable for evaluating multi-hop reasoning and noise-robust relationship discovery.

**NGD-Fin.** Adapted from LDBC-FIN (Qi et al., 2025), NGD-Fin models financial transaction ecosystems involving accounts, entities, events, and monetary flows (Yang et al., 2024). Its temporal and numerical properties, together with multi-edge transaction patterns, make it suitable for evaluating analytical queries, latent-risk discovery, and robustness to noisy or missing financial relations.

**NGD-Prime.** Derived from PrimeKG (Chandak et al., 2023), NGD-Prime integrates biomedical entities such as diseases, drugs, genes, and biological processes. Its dense cross-type relations

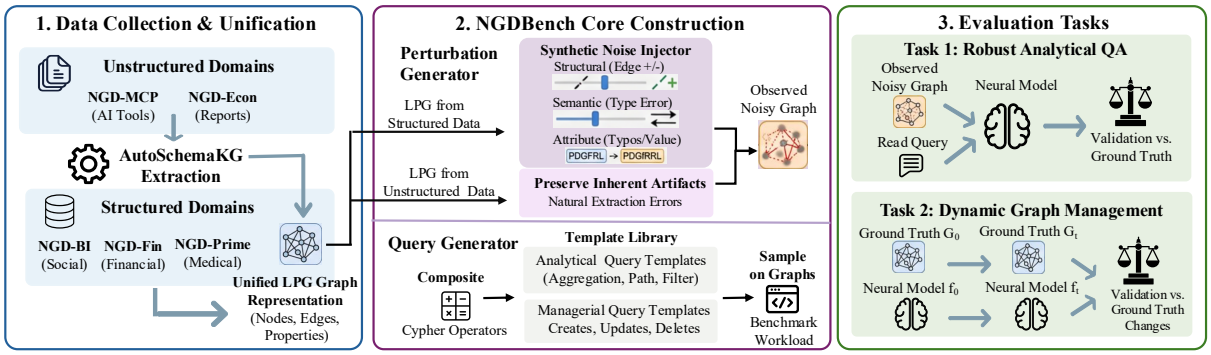


Figure 2: The NGDBench Framework Pipeline.(1) NGDBench unifies diverse data into LPG graphs, (2) constructs paired clean/noisy graphs via perturbation, and samples Cypher queries from template libraries. (3) Systems are evaluated on robust analytical QA and dynamic graph management under noisy observations.

make it suitable for evaluating latent relation recovery and evidence aggregation across heterogeneous biomedical entities.

#### 3.1.2 Unstructured Domains

We further include two unstructured sources to evaluate whether graph-based neural data management can expose latent structure from raw text. We use AutoSchemaKG (Bai et al., 2025a) to extract entities, relations, and schemas, and instantiate them as Labeled Property Graphs.

**NGD-MCP.** Built from MCP tool and trajectory data in Toucan-1.5M (Xu et al., 2025), NGD-MCP captures agent workflow dependencies, supporting graph reasoning over tools, actions, and intermediate states.

**NGD-Econ.** NGD-Econ is derived from company annual reports in LongBench (Bai et al., 2025c). Financial reports contain dispersed entities, metrics, events, and numerical evidence, which are important for long-document understanding and multi-step financial analysis (Chen et al., 2021; Reddy et al., 2024). By converting reports into graphs, NGD-Econ supports cross-document aggregation, relation discovery, and reasoning over implicit business and operational links.

### 3.2 Perturbation Generation

The perturbation component is designed to simulate noisy data conditions commonly found in real-world data management. To ground this design, we survey real-world datasets, prior synthetic-noise studies and data quality reports, with detailed evidence summarized in Appendix C, Table 7. The survey shows that noise levels vary substantially across domains and noise families: reported real-world noise ranges from 0.55% attribute errors to over 70% missing relations, while prior synthetic studies commonly inject noise at 10%–50%.

These observations suggest that no single fixed noise ratio can faithfully represent real-world data imperfections. We therefore design NGDBENCH as a configurable and traceable perturbation environment rather than a fixed-noise benchmark. It covers the major noise families identified in the survey and exposes parameterized controls over the injection ratio of each noise type, enabling both moderate default evaluation and customized stress testing. For each generated observed graph, NGDBENCH records the injected perturbations in noise log files, allowing users to inspect which nodes, edges, labels, or attributes were modified. Details on noise categories, real-world motivation, injection ratios, and perturbation logs are provided in Appendix C.

For the structured datasets (*NGD-BI*, *NGD-Fin*, and *NGD-Prime*), we perturb clean latent graphs to construct observed graphs. The Perturbation Generator injects topological, schema, and attribute noise. These perturbations capture the gap between clean latent structure and imperfect observed data.

For the unstructured datasets (*NGD-MCP* and *NGD-Econ*), we do not add extra perturbations. Their graphs are produced by automated extraction pipelines and already contain realistic imperfections, such as entity duplication, mention variation, relation extraction errors, and type inference errors.

### 3.3 Query Generation

To align NGDBENCH with practical graph data management systems, we adopt the **Labeled Property Graph (LPG)** model and the **Cypher** query language. LPGs store rich properties directly on nodes and edges, making them suitable for real-world data with attributes on both entities and relations, such as financial transactions with timestamps and amounts. Cypher is the de facto query

Table 2: Detailed statistics of graph structures (groundtruth and perturbed graphs) and query categories across different domains.

| Category | | NGD-BI | NGD-Fin | NGD-Prime | NGD-MCP | NGD-Econ | Summary |
|---|---|---|---|---|---|---|---|
| Graph Structures (Groundtruth) | #Nodes | 2,997,352 | 10,865 | 129,312 | 181,689 | 33,164 | – |
| | #Edges | 17,196,776 | 57,818 | 8,100,498 | 351,263 | 63,757 | – |
| Graph Structures (Perturbed) | #Nodes | 2,997,352 | 10,865 | 129,312 | 181,689 | 33,164 | – |
| | #Edges | 17,971,365 | 60,476 | 8,465,124 | 351,263 | 63,757 | – |
| Query Categories | #Analytical (No Agg) | 17,880 | 4,264 | 14,043 | 8,336 | 1,707 | 58,321 |
| | #Analytical (Agg.) | 9,580 | 4,412 | 6,494 | 4,167 | 1,050 | 25,703 |
| | #Management | 2,503 | 1,400 | 2,207 | – | – | 6,110 |

language for LPGs and is widely supported by modern graph engines such as Neo4j [1] and FalkorDB [2], improving the benchmark's practical relevance.

We construct a Cypher query template library by organizing standard Cypher patterns into core operators, and generate benchmark queries using a perturbation-aware query sampler.

#### 3.3.1 Query Template Library Construction

To ensure broad Cypher coverage, we survey both the official Cypher operator set[3] and representative production-level Cypher queries. Although production queries can be complex, their logical structure is typically composed from recurring functional building blocks. Many official operators are runtime variants of these core functions, while others are not central to logical query semantics.

We therefore extract 29 core functional operators, including 23 analytical operators and 5 management operators, to guide query construction. These operators cover retrieval, traversal, filtering, projection, aggregation, and state-changing primitives. We then build analytical templates in stages, from atomic value-agnostic templates to no-aggregation (NoAGG) templates and aggregation (AGG) templates with `SUM`, `COUNT`, `MIN`, `MAX`, and `AVG`. This design captures the operator- and pattern-level foundations of many production analytical queries, making NGDBENCH a starting point for future production-like neural data management workloads.

We further construct management templates to evaluate dynamic data management. These templates focus on frequent fine-grained operations, organized into batches that represent logical units of work involving creation, update, or deletion. Details of the operators, query templates, and management workloads are provided in Appendices D.1, D.2.1, D.3, and E.

[1] https://neo4j.com/

[2] https://www.falkordb.com/

[3] https://neo4j.com/docs/cypher-manual/25/planning-and-tuning/operators/operators-detail

#### 3.3.2 Query Sampling and Reconstruction

We design a controllable query sampler that instantiates templates over both clean and perturbed graph regions. Users can specify the sampling ratio between clean regions, which support basic execution evaluation, and noisy regions, which test robustness under incomplete and corrupted observations.

We then reconstruct sampled Cypher queries into natural-language questions using an LLM, creating paired text-query instances for Text-to-Cypher and semantic understanding evaluation.

For non-aggregation queries with large answer sets, we reconstruct them as Boolean verification tasks rather than discarding them. Queries with fewer than 32 answers are kept unchanged; otherwise, we sample positive candidates from groundtruth answers and negatives from unrelated entities, and ask whether each candidate satisfies the query constraints. This setting better reflects practical scenarios where users verify specific entities rather than exhaustively enumerate all answers. Details are provided in Appendix D.2.5.

### 3.4 Task Formulation

We evaluate NGDBENCH along the following two dimensions.

#### 3.4.1 Task I: Robust Analytical Querying

This task evaluates whether a model can answer analytical queries over imperfect observed data while recovering the intended latent information.

**Problem Definition.** Let $\mathcal{G}^*$ denote the clean latent graph, and let $\tilde{\mathcal{G}} = \Phi(\mathcal{G}^*)$ be the observed graph generated by a perturbation function $\Phi$. Given a natural language query $q$ and the observed graph context $\tilde{\mathcal{G}}$, a model $f_\theta$ produces either a direct answer or a formal query such as Cypher. The target answer is obtained by executing the query logic on the clean graph:

$$a^* = \text{Exec}(q, \mathcal{G}^*). \tag{1}$$

The objective is to minimize the discrepancy between the model prediction and the clean-graph answer:

$$\min_{\theta} \mathcal{L}\left(f_\theta(q, \tilde{\mathcal{G}}), a^*\right), \tag{2}$$

where $\mathcal{L}$ denotes the task-specific evaluation metric, such as F1 score, accuracy, or relative error. This task tests whether models can reason beyond noisy observations rather than merely querying the perturbed graph.

#### 3.4.2 Task II: Sequential Editing for Dynamic Data Management

This task evaluates whether a model can track graph states under sequential `CREATE`, `UPDATE`, and `DELETE` operations.

**Problem Definition.** Let $\mathcal{G}_0$ be the initial graph state. At each step $t$ in a $T$-step session, the model receives a modification instruction $m_t$ and a validation query $q_t$. The graph update in Eq. (3), the gold-answer computation in Eq. (4), and the model prediction in Eq. (5) are defined as follows, where $\mathcal{H}_t = \langle \mathcal{G}_0, m_1, \ldots, m_t \rangle$ denotes the modification history:

$$\mathcal{G}_t = \mathrm{Apply}(\mathcal{G}_{t-1}, m_t), \tag{3}$$

$$a_t = \mathrm{Exec}(q_t, \mathcal{G}_t), \tag{4}$$

$$\hat{a}_t = \operatorname*{argmax}_{a \in \mathcal{A}} P_\theta(a \mid q_t, \mathcal{H}_t). \tag{5}$$

Then, we report step-wise accuracy over the full trajectory:

$$\mathcal{L}_{step} = \frac{1}{T} \sum_{t=1}^{T} I(\hat{a}_t = a_t), \tag{6}$$

where $I(\cdot)$ indicates whether the predicted answer matches the ground truth. This metric penalizes intermediate state-tracking errors rather than evaluating only the final state.

## 4 Experiment

### 4.1 Analytical Query on Structured Domains

**Experimental Settings and Metrics.** For each benchmark query, we execute the oracle Cypher query on a traditional graph database Neo4j over both the clean latent graph and the perturbed observed graph. The clean-graph result serves as the target answer, while the observed-graph result provides an oracle upper bound for methods restricted to noisy observations.

We use translated natural-language questions as model inputs and evaluate three query settings. (1) *No Aggregation (NoAgg)* queries return answer sets, so we report **Jaccard similarity** and **F1 score**. (2) *Boolean Queries* reformulate large-result queries as binary verification tasks, and we report **Accuracy**. (3) *Aggregation (Agg)* queries return scalar numerical values. To account for large variances in answer scales, we adopt relative-error-based metrics: **Median Relative Error (MdRE)** to capture typical prediction error while remaining robust to outliers, **Mean Squared Logarithmic Error (MSLE)** to penalize multiplicative deviations, and **Symmetric Mean Absolute Percentage Error (sMAPE)** for a scale-invariant and zero-robust measure of error. We have presented the calculation formula for metric in Appendix F.

**Baselines.** We evaluate two representative neural-based graph query paradigms: *Text-to-Cypher* and *GraphRAG*. These are currently the most common ways LLM-based systems access and reason over graph data. Text-to-Cypher uses an LLM to convert query intent into executable graph queries, thereby combining neural language understanding with symbolic database execution. GraphRAG instead retrieves relevant graph evidence, typically in textualized triple form, and asks an LLM to reason over the retrieved context. Compared with exact symbolic matching, both paradigms have greater potential to tolerate noisy observations and capture latent semantic associations, making them suitable test cases for neural graph data management.

The *Text-to-Cypher* baselines include the standard Neo4j-Text2Cypher implementation and strong code-generation LLMs, including DeepSeek V3.2 (DeepSeek-AI et al., 2025), GPT-5.1-codex[4], and Qwen3-Coder-480B (Yang et al., 2025). For *GraphRAG*, we use a dense triple retriever with Qwen3-0.6B (Team, 2025), retrieving the top-15 triples by semantic similarity.

Traditional CQA methods are excluded because they are largely limited to Existential First-Order logic and cannot execute full Cypher workloads such as aggregation or dynamic operations. Generic vector databases are also excluded because they mainly support similarity search over pre-indexed embeddings. Handling noise distributed across labels, attributes, relations, and paths would

[4]https://openai.com/zh-Hans-CN/index/gpt-5-1-codex-max/

Table 3: Performance comparison across three datasets under non-aggregation and aggregation settings. ("——" indicates values exceeding 10,000. Bold and underlined values denote the best and second-best results respectively.)

| Baseline | NGD-Fin | | | | | NGD-BI | | | | | NGD-Prime | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | NoAgg | | Agg | | | NoAgg | | Agg | | | NoAgg | | Agg | | |
| | Jaccard | F1 | MdRE | MSLE | sMAPE | Jaccard | F1 | MdRE | MSLE | sMAPE | Jaccard | F1 | MdRE | MSLE | sMAPE |
| Oracle Cypher | 0.448 | 0.471 | 0.039 | 0.649 | 0.119 | 0.887 | 0.911 | 0.355 | 8.388 | 0.783 | 0.736 | 0.790 | 0.203 | 7.990 | 0.562 |
| Neo4j-Text2Cypher | 0.319 | 0.319 | 0.660 | 31.877 | 0.975 | 0.212 | 0.213 | —— | 45.582 | 1.681 | 0.208 | 0.195 | —— | 40.760 | 1.481 |
| GPT5.1-Codex | 0.358 | 0.359 | 0.111 | 12.425 | 0.394 | **0.555** | **0.571** | **0.987** | **15.921** | **1.089** | **0.510** | **0.537** | **0.206** | **12.250** | **0.743** |
| DeepseekV3.2 | 0.345 | 0.345 | **0.102** | **5.636** | 0.334 | 0.394 | 0.404 | 0.992 | 20.862 | 1.182 | 0.426 | 0.446 | 0.409 | 12.547 | 0.835 |
| Qwen3-Coder | **0.441** | **0.456** | **0.102** | 6.054 | **0.332** | 0.306 | 0.315 | 0.992 | 22.655 | 1.198 | 0.389 | 0.410 | 0.430 | 13.197 | 0.852 |
| GraphRAG | 0.004 | 0.004 | 161.500 | 27.770 | 1.564 | 0.001 | 0.001 | —— | 52.527 | 1.994 | 0.002 | 0.002 | 705.650 | 40.659 | 1.725 |

Table 4: Performance comparison on management queries across different datasets.

| Baseline | NGD-Fin | | | NGD-BI | | | NGD-Prime | | |
|---|---|---|---|---|---|---|---|---|---|
| | MLRE | MSLE | sMAPE | MLRE | MSLE | sMAPE | MLRE | MSLE | sMAPE |
| Neo4j-Text2Cypher | 8.065 | 17.038 | 1.137 | 0.759 | 3.323 | 0.280 | 3.667 | 16.072 | 0.337 |
| GPT5.1-Codex | 3.036 | 11.727 | 0.771 | 0.516 | **2.327** | 0.233 | **3.342** | **15.138** | 0.360 |
| DeepseekV3.2 | 2.778 | 14.940 | 0.716 | **0.455** | 2.330 | **0.171** | 3.665 | 16.130 | 0.352 |
| Qwen3-Coder | **2.676** | **4.619** | **0.610** | 0.562 | 2.656 | 0.239 | 3.403 | 15.619 | **0.309** |
| GraphRAG | 5.129 | 17.510 | 1.518 | 9.043 | 81.106 | 1.979 | 6.337 | 41.776 | 1.958 |

require additional task-specific indexing and retrieval design, which falls outside their default query model. We instead include an *Oracle Cypher* control on the perturbed graph to isolate data-noise effects from model generation errors.

**Results on NoAgg and Agg Queries.** Table 3 reports results on NoAgg and Agg queries. Text-to-Cypher methods generally outperform GraphRAG, especially on aggregation queries, because structured query execution can retrieve complete matching results, while GraphRAG relies on dense top-$k$ retrieval and may miss relevant evidence when many triples are required.

However, Text-to-Cypher is still bounded by noise in the observed graph. Some aggregation-error metrics can become very large when realistic corruptions affect numerical attributes, since small data-entry or OCR may substantially distort downstream analytical results such as `SUM` or `AVG`. To separate model errors from data-noise effects, we split queries into consistent and inconsistent subsets, depending on whether their answers change after perturbation. Results in Appendix G show that models perform much better on consistent queries but degrade sharply on inconsistent ones, indicating that the gap between observed data and latent ground truth is a key bottleneck.

Overall, Text-to-Cypher works when observations are reliable but fails to infer beyond corrupted or missing records. GraphRAG uses textual evidence flexibly but lacks complete recall and precise reasoning. These limitations motivate neural graph data management methods that combine symbolic execution with inference over noisy observations.

**Results on Boolean Queries.** Figure 3 left reports results on Boolean queries. In this setting, GraphRAG improves substantially and becomes competitive with Text-to-Cypher. The candidate-based formulation narrows the retrieval space, making dense retrieval more reliable. In contrast, Boolean verification can require more complex Cypher logic, making it less convenient for Text-to-Cypher models. This suggests that RAG-style methods can be effective for local verification, but remain insufficient for complex analytical queries requiring broad and complete evidence retrieval.

### 4.2 Managerial Queries on Structured Data

We further extend our evaluation to the dynamic management phase. In this setting, each management batch consists of five sequential steps; after each step, we execute a validation query to verify the updated state. Given the varying scales of the target values, we adopt a suite of relative-error-based metrics: **Mean Log Relative Error (MLRE)** is used to capture typical prediction errors while remaining robust to outliers; **Mean Squared Logarithmic Error (MSLE)** is employed to penalize multiplicative deviations rather than additive ones; and **Symmetric Mean Absolute Percentage Error (sMAPE)** is selected for its scale invariance and stability, particularly when handling values close to zero.

The results are shown in Table 4 and Fig. 3 right. Text-to-Cypher methods perform well because it can generate explicit edit operations that directly update the graph. However, it is prone to error accumulation: an incorrect update at an earlier step can propagate and affect subsequent states. In contrast, GraphRAG reasons over the full edit history

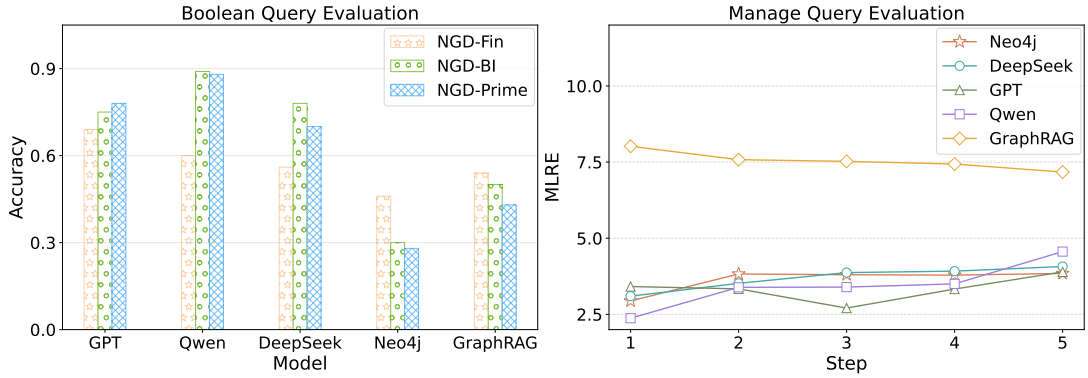


Figure 3: The left figure is the evaluation of boolean queries on three datasets. The right figure is the evaluation of dynamic steps on NGD-Prime, where lower MLRE indicates better performance.

in context, which helps mitigate cascading errors.

### 4.3 Analytical Query on Unstructured Data

We further evaluate graph-based RAG methods on the unstructured datasets, NGD-MCP and NGD-Econ. We compare three representative retrieval-based baselines: GraphRAG, which performs embedding-based retrieval over textualized triples extracted by our pipeline; HippoRAG2 (Gutiérrez et al., 2025), a recent text-first graph RAG method operating over raw documents; and ToG(**?**), a hybrid retrieval framework that iteratively searches over graph and textual contexts. Following prior graph-based RAG studies (Gutiérrez et al., 2025; Trivedi et al., 2022), we report token-level F1 for non-aggregation queries. We report **MSLE** for aggregation queries, and more metrics on this task are desplayed in Appendix G, Table 11.

As shown in Table 5, GraphRAG achieves the best results across both datasets, obtaining the lowest aggregation error and the highest non-aggregation F1. This suggests that directly retrieving textualized triples provides stronger evidence than reconstructing query-specific reasoning graphs from raw passages, especially when questions are grounded in local graph neighborhoods. However, the absolute performance remains limited: even the best NoAgg F1 is below 0.25, and aggregation errors remain non-negligible, indicating that current RAG pipelines still struggle with precise complex querying and numerical reasoning.

This reflects a key limitation of retrieval-based pipelines: retrieval and computation are separated. RAG methods first retrieve a bounded top-$k$ context and then ask the LLM to reason over truncated evidence. While this can help when answers appear in a small local neighborhood, it is weak for complex path queries and aggregation tasks, where accurate answers require complete evidence rather than a few retrieved snippets. As a result, RAG-based methods may miss relevant tuples, include distractors, or fail to compute exact counts and aggregate values. We provide concrete success and failure cases in Appendix H.2.

Table 5: Performance of graph-based RAG methods on unstructured datasets. Agg reports MSLE (↓), and NoAgg reports F1 (↑).

| Baseline | NGD-MCP | | NGD-Econ | |
|---|---|---|---|---|
| | Agg ↓ | NoAgg ↑ | Agg ↓ | NoAgg ↑ |
| GraphRAG | **0.169** | **0.194** | **0.673** | **0.242** |
| HippoRAG2 | 1.050 | 0.022 | 2.138 | 0.040 |
| ToG-RAG | 0.359 | 0.013 | 3.809 | 0.212 |

## 5 Conclusion

We introduced NGDBENCH, a benchmark for advancing graph-based neural data management beyond passive querying over observed data. Unlike prior benchmarks limited to elementary logical operations, NGDBENCH spans five diverse domains and supports the full spectrum of Cypher queries, while rigorously testing robustness against realistic noise and dynamic updates. Our evaluation of state-of-the-art LLMs and RAG approaches reveals significant limitations in precise analytical reasoning and noise resilience. Consequently, NGDBENCH serves as a critical testbed for future research, paving the way for more reliable and capable neural database systems. We hope NGDBENCH will stimulate principled work on noise-aware querying, controlling error accumulation in sequential edits, and standardized evaluation protocols for next-generation neural data management systems.

## Limitations

One limitation of NGDBENCH is that the dynamic management workload is currently constructed only for structured graph datasets. For graphs extracted from unstructured sources, such as documents or reports, we do not yet provide management queries involving `CREATE`, `UPDATE`, or `DELETE`. This is because editing extracted knowledge graphs is not simply a syntactic operation. It often requires factual grounding and domain-aware validation, especially in knowledge-intensive domains such as biomedicine, where adding, revising, or deleting a relation should be supported by reliable evidence.

In addition, our evaluation primarily focuses on answer accuracy under noisy and evolving graph states. Other practical factors, such as execution efficiency, memory cost, uncertainty calibration, and evidence quality, are left for future work. Extending NGDBENCH along these dimensions would make it a more comprehensive testbed for neural graph data management.

Nevertheless, extending dynamic management workloads to unstructured-source graphs is an important direction. In real-world data management, unstructured data is also continuously evolving: new documents arrive, outdated information must be updated or archived, and extracted entities and relations may need to be corrected as evidence changes. Designing principled management operations for such graphs would enable NGDBENCH to better evaluate how future neural data management systems maintain consistency over evolving, partially observed, and evidence-grounded knowledge.

## References


Amirhossein Abaskohi, Tianyi Chen, Miguel Muñoz-Mármol, Curtis Fox, Amrutha Varshini Ramesh, Étienne Marcotte, Xing Han Lù, Nicolas Chapados, Spandana Gella, Peter West, Giuseppe Carenini, Christopher Pal, Alexandre Drouin, and Issam H. Laradji. 2026. Drbench: A realistic benchmark for enterprise deep research. *Preprint*, arXiv:2510.00172.

Alaa Althubaiti. 2016. Information bias in health research: Definition, pitfalls, and adjustment methods. *Journal of Multidisciplinary Healthcare*, 9:211–217.

Erik Arakelyan, Daniel Daza, Pasquale Minervini, and Michael Cochez. 2021. Complex query answering with neural link predictors. In *9th International Conference on Learning Representations, ICLR 2021, Virtual Event, Austria, May 3-7, 2021*. OpenReview.net.

Jiaxin Bai, Wei Fan, Qi Hu, Qing Zong, Chunyang Li, Hong Ting Tsang, Hongyu Luo, Yauwai Yim, Haoyu Huang, Xiao Zhou, Feng Qin, Tianshi Zheng, Xi Peng, Xin Yao, Huiwen Yang, Leijie Wu, Yi Ji, Gong Zhang, Renhai Chen, and Yangqiu Song. 2025a. Autoschemakg: Autonomous knowledge graph construction through dynamic schema induction from web-scale corpora. *Preprint*, arXiv:2505.23628.

Jiaxin Bai, Chen Luo, Zheng Li, Qingyu Yin, and Yangqiu Song. 2024. Understanding inter-session intentions via complex logical reasoning. In *Proceedings of the 30th ACM SIGKDD Conference on Knowledge Discovery and Data Mining, KDD 2024, Barcelona, Spain, August 25-29, 2024*, pages 71–82. ACM.

Jiaxin Bai, Chen Luo, Zheng Li, Qingyu Yin, Bing Yin, and Yangqiu Song. 2023. Knowledge graph reasoning over entities and numerical values. In *Proceedings of the 29th ACM SIGKDD Conference on Knowledge Discovery and Data Mining, KDD 2023, Long Beach, CA, USA, August 6-10, 2023*, pages 57–68. ACM.

Jiaxin Bai, Zihao Wang, Yukun Zhou, Hang Yin, Weizhi Fei, Qi Hu, Zheye Deng, Jiayang Cheng, Tianshi Zheng, Hong Ting Tsang, Yisen Gao, Zhongwei Xie, Yufei Li, Lixin Fan, Binhang Yuan, Wei Wang, Lei Chen, Xiaofang Zhou, and Yangqiu Song. 2025b. Top ten challenges towards agentic neural graph databases. *IEEE Data Eng. Bull.*, 49(1):104–123.

Yushi Bai, Shangqing Tu, Jiajie Zhang, Hao Peng, Xiaozhi Wang, Xin Lv, Shulin Cao, Jiazheng Xu, Lei Hou, Yuxiao Dong, Jie Tang, and Juanzi Li. 2025c. Longbench v2: Towards deeper understanding and reasoning on realistic long-context multitasks. *Preprint*, arXiv:2412.15204.

Maciej Besta, Patrick Iff, Florian Scheidl, Kazuki Osawa, Nikoli Dryden, Michal Podstawski, Tiancheng Chen, and Torsten Hoefler. 2022. Neural graph databases. In *Proceedings of the First Learning on Graphs Conference*, volume 198 of *Proceedings*

*of Machine Learning Research*, pages 31:1–31:38. PMLR.

Svetlana Bryzgalova, Sven Lerner, Martin Lettau, and Markus Pelger. 2025. Missing financial data. *The Review of Financial Studies*, 38(3):803–882.

Payal Chandak, Kexin Huang, and Marinka Zitnik. 2023. Building a knowledge graph to enable precision medicine. *Nature Scientific Data*.

Xuelu Chen, Ziniu Hu, and Yizhou Sun. 2022. Fuzzy logic based logical query answering on knowledge graphs. In *Thirty-Sixth AAAI Conference on Artificial Intelligence, AAAI 2022, Thirty-Fourth Conference on Innovative Applications of Artificial Intelligence, IAAI 2022, The Twelveth Symposium on Educational Advances in Artificial Intelligence, EAAI 2022 Virtual Event, February 22 - March 1, 2022*, pages 3939–3948. AAAI Press.

Zhiyu Chen, Wenhu Chen, Charese Smiley, Sameena Shah, Iana Borova, Dylan Langdon, Reema Moussa, Matt Beane, Ting-Hao Huang, Bryan Routledge, and William Yang Wang. 2021. FinQA: A dataset of numerical reasoning over financial data. In *Proceedings of the 2021 Conference on Empirical Methods in Natural Language Processing*, pages 3697–3711.

DeepSeek-AI, Aixin Liu, Aoxue Mei, Bangcai Lin, Bing Xue, Bingxuan Wang, Bingzheng Xu, Bochao Wu, Bowei Zhang, Chaofan Lin, Chen Dong, Chengda Lu, Chenggang Zhao, Chengqi Deng, Chenhao Xu, Chong Ruan, Damai Dai, Daya Guo, Dejian Yang, and 245 others. 2025. Deepseek-v3.2: Pushing the frontier of open large language models. *Preprint*, arXiv:2512.02556.

Caglar Demir, Michel Wiebesiek, Renzhong Lu, Axel-Cyrille Ngonga Ngomo, and Stefan Heindorf. 2023. Litcqd: Multi-hop reasoning in incomplete knowledge graphs with numeric literals. In *Machine Learning and Knowledge Discovery in Databases: Research Track - European Conference, ECML PKDD 2023, Turin, Italy, September 18-22, 2023, Proceedings, Part III*, volume 14171 of *Lecture Notes in Computer Science*, pages 617–633. Springer.

Nick Drummond and Rob Shearer. 2006. The open world assumption. In *eSI workshop: the closed world of databases meets the open world of the semantic web*, volume 15, page 1.

Mikhail Galkin, Etienne G. Denis, Jiapeng Wu, and William L. Hamilton. 2022. Nodepiece: Compositional and parameter-efficient representations of large knowledge graphs. In *The Tenth International Conference on Learning Representations, ICLR 2022, Virtual Event, April 25-29, 2022*. OpenReview.net.

Yisen Gao, Jiaxin Bai, Yi Huang, Xingcheng Fu, Qingyun Sun, and Yangqiu Song. 2025a. Unifying deductive and abductive reasoning in knowledge graphs with masked diffusion model. *Preprint*, arXiv:2510.11462.

Yisen Gao, Jiaxin Bai, Tianshi Zheng, Qingyun Sun, Ziwei Zhang, Jianxin Li, Yangqiu Song, and Xingcheng Fu. 2025b. Controllable logical hypothesis generation for abductive reasoning in knowledge graphs. *CoRR*, abs/2505.20948.

Justin Gu, Rishabh Ranjan, Charilaos Kanatsoulis, Haiming Tang, Martin Jurkovic, Valter Hudovernik, Mark Znidar, Pranshu Chaturvedi, Parth Shroff, Fengyu Li, and Jure Leskovec. 2026. Relbench v2: A large-scale benchmark and repository for relational data. *Preprint*, arXiv:2602.12606.

Manping Guo, Yiming Wang, Qiaoning Yang, Rui Li, Yang Zhao, Chenfei Li, Mingbo Zhu, Yao Cui, Xin Jiang, Song Sheng, Qingna Li, and Rui Gao. 2023. Normal workflow and key strategies for data cleaning toward real-world data: Viewpoint. *Interactive Journal of Medical Research*, 12:e44310.

Bernal Jiménez Gutiérrez, Yiheng Shu, Weijian Qi, Sizhe Zhou, and Yu Su. 2025. From rag to memory: Non-parametric continual learning for large language models. *arXiv preprint arXiv:2502.14802*.

William L. Hamilton, Payal Bajaj, Marinka Zitnik, Dan Jurafsky, and Jure Leskovec. 2018. Embedding logical queries on knowledge graphs. In *Advances in Neural Information Processing Systems 31: Annual Conference on Neural Information Processing Systems 2018, NeurIPS 2018, December 3-8, 2018, Montréal, Canada*, pages 2030–2041.

Qi Hu, Haoran Li, Jiaxin Bai, Zihao Wang, and Yangqiu Song. 2024. Privacy-preserved neural graph databases. In *Proceedings of the 30th ACM SIGKDD Conference on Knowledge Discovery and Data Mining*, KDD '24, page 1108–1118, New York, NY, USA. Association for Computing Machinery.

Weihua Hu, Matthias Fey, Marinka Zitnik, Yuxiao Dong, Hongyu Ren, Bowen Liu, Michele Catasta, and Jure Leskovec. 2020. Open graph benchmark: Datasets for machine learning on graphs. *Advances in neural information processing systems*, 33:22118–22133.

Yi Huang, Qingyun Sun, Yisen Gao, Haonan Yuan, Xingcheng Fu, and Jianxin Li. 2025. Is the information bottleneck robust enough? towards label-noise resistant information bottleneck learning. *arXiv preprint arXiv:2512.10573*.

Lucas Jarnac, Yoan Chabot, and Miguel Couceiro. 2025. Uncertainty management in the construction of knowledge graphs: A survey. *Transactions on Graph Data and Knowledge*, 3(1):3:1–3:48.

Moe Kayali, Fabian Wenz, Nesime Tatbul, and Cagatay Demiralp. 2025. Mind the data gap: Bridging llms to enterprise data integration. In *Proceedings of the 15th Conference on Innovative Data Systems Research (CIDR)*, Amsterdam, The Netherlands.

Myung Jun Kim, Léo Grinsztajn, and Gaël Varoquaux. 2024. Carte: Pretraining and transfer for tabular learning. *Preprint*, arXiv:2402.16785.

Longbin Lai, Changwei Luo, Yunkai Lou, Mingchen Ju, and Zhengyi Yang. 2025. Graphy'our data: Towards end-to-end modeling, exploring and generating report from raw data. *Preprint*, arXiv:2502.16868.

Neo4j. n.d. Why Graph Databases? https://neo4j.com/why-graph-databases/. Accessed: 2026-02-08.

Makbule Gulcin Ozsoy, Leila Messallem, Jon Besga, and Gianandrea Minneci. 2024. Text2cypher: Bridging natural language and graph databases. *Preprint*, arXiv:2412.10064.

Shipeng Qi, Bing Tong, Jiatao Hu, Heng Lin, Yue Pang, Wei Yuan, Songlin Lyu, Zhihui Guo, Ke Huang, Xujin Ba, Qiang Yin, Youren Shen, Yan Zhou, Tao Lv, Jia Li, Lei Zou, Yongwei Wu, Gábor Szárnyas, Xiaowei Zhu, and 2 others. 2025. The ldbc financial benchmark: Transaction workload. *Proc. VLDB Endow.*, 18(9):3007–3020.

Siyi Qian, Haochao Ying, Renjun Hu, Jingbo Zhou, Jintai Chen, Danny Z. Chen, and Jian Wu. 2023. Robust training of graph neural networks via noise governance. In *Proceedings of the Sixteenth ACM International Conference on Web Search and Data Mining*, pages 607–615.

Reddit User. 2023. Can someone explain a graph database as compared to a relational database? https://www.reddit.com/r/cscareerquestions/comments/17ru9l3/can_someone_explain_a_graph_database_as_compared/. Accessed: 2026-02-08.

Varshini Reddy, Rik Koncel-Kedziorski, Viet Dac Lai, Michael Krumdick, Charles Lovering, and Chris Tanner. 2024. DocFinQA: A long-context financial reasoning dataset. In *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 2: Short Papers)*, pages 445–458.

Hongyu Ren, Mikhail Galkin, Michael Cochez, Zhaocheng Zhu, and Jure Leskovec. 2023. Neural graph reasoning: Complex logical query answering meets graph databases. *Preprint*, arXiv:2303.14617.

Hongyu Ren, Weihua Hu, and Jure Leskovec. 2020. Query2box: Reasoning over knowledge graphs in vector space using box embeddings. In *8th International Conference on Learning Representations, ICLR 2020, Addis Ababa, Ethiopia, April 26-30, 2020*. OpenReview.net.

Christian Senger, Jens Kaltschmidt, Simon P. W. Schmitt, Markus G. Pruszydlo, and Walter E. Haefeli. 2010. Misspellings in drug information system queries: Characteristics of drug name spelling errors and strategies for their prevention. *International Journal of Medical Informatics*, 79(12):832–839.

Baoxu Shi and Tim Weninger. 2018. Open-world knowledge graph completion. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 32.

Gábor Szárnyas, Jack Waudby, Benjamin A. Steer, Dávid Szakállas, Altan Birler, Mingxi Wu, Yuchen Zhang, and Peter Boncz. 2022. The ldbc social network benchmark: Business intelligence workload. *Proc. VLDB Endow.*, 16(4):877–890.

Qwen Team. 2025. Qwen3 technical report. *Preprint*, arXiv:2505.09388.

Harsh Trivedi, Niranjan Balasubramanian, Tushar Khot, and Ashish Sabharwal. 2022. Musique: Multi-hop questions via single-hop question composition. *Transactions of the Association for Computational Linguistics*, 10:539–554.

Pablo Villalobos, Anson Ho, Jaime Sevilla, Tamay Besiroglu, Lennart Heim, and Marius Hobbhahn. 2024. Position: will we run out of data? limits of llm scaling based on human-generated data. In *Proceedings of the 41st International Conference on Machine Learning*, ICML'24. JMLR.org.

Minjie Wang, Quan Gan, David Wipf, Zhenkun Cai, Ning Li, Jianheng Tang, Yanlin Zhang, Zizhao Zhang, Zunyao Mao, Yakun Song, Yanbo Wang, Jiahang Li, Han Zhang, Guang Yang, Xiao Qin, Chuan Lei, Muhan Zhang, Weinan Zhang, Christos Faloutsos, and Zheng Zhang. 2024. 4dbinfer: a 4d benchmarking toolbox for graph-centric predictive modeling on rdbs. In *Proceedings of the 38th International Conference on Neural Information Processing Systems*, NIPS '24, Red Hook, NY, USA. Curran Associates Inc.

Zihao Wang, Hang Yin, and Yangqiu Song. Benchmarking the combinatorial generalizability of complex query answering on knowledge graphs. In *Thirty-fifth Conference on Neural Information Processing Systems Datasets and Benchmarks Track (Round 2)*.

Zhangchen Xu, Adriana Meza Soria, Shawn Tan, Anurag Roy, Ashish Sunil Agrawal, Radha Poovendran, and Rameswar Panda. 2025. Toucan: Synthesizing 1.5m tool-agentic data from real-world mcp environments. *Preprint*, arXiv:2510.01179.

An Yang, Anfeng Li, Baosong Yang, Beichen Zhang, Binyuan Hui, Bo Zheng, Bowen Yu, Chang Gao, Chengen Huang, Chenxu Lv, Chujie Zheng, Dayiheng Liu, Fan Zhou, Fei Huang, Feng Hu, Hao Ge, Haoran Wei, Huan Lin, Jialong Tang, and 41 others. 2025. Qwen3 technical report. *Preprint*, arXiv:2505.09388.

Haotong Yang, Zhouchen Lin, and Muhan Zhang. 2022. Rethinking knowledge graph evaluation under the open-world assumption. In *Advances in Neural Information Processing Systems 35*.

Hongyang Yang, Boyu Zhang, Neng Wang, Cheng Guo, Xiaoli Zhang, Likun Lin, Junlin Wang, Tianyu Zhou, Mao Guan, Runjia Zhang, and Christina Dan Wang. 2024. Finrobot: An open-source ai agent platform for financial applications using large language models. *CoRR*, abs/2405.14767.

Hang Yin, Zihao Wang, Weizhi Fei, and Yangqiu Song. 2025. Efo k-cqa: Towards knowledge graph complex query answering beyond set operation. In *Proceedings of the 31st ACM SIGKDD Conference on Knowledge Discovery and Data Mining V. 2*, pages 5876–5887.

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, Brussels, Belgium. Association for Computational Linguistics.

Zhanqiu Zhang, Jie Wang, Jiajun Chen, Shuiwang Ji, and Feng Wu. 2021. Cone: Cone embeddings for multi-hop reasoning over knowledge graphs. In *Advances in Neural Information Processing Systems 34: Annual Conference on Neural Information Processing Systems 2021, NeurIPS 2021, December 6-14, 2021, virtual*, pages 19172–19183.

Shixiong Zhao and Hideaki Takeda. 2026. Diagnosing and mitigating semantic inconsistencies in wikidata's classification hierarchy. Preprint.

Shuyan Zhou, Frank F. Xu, Hao Zhu, Xuhui Zhou, Robert Lo, Abishek Sridhar, Xianyi Cheng, Tianyue Ou, Yonatan Bisk, Daniel Fried, Uri Alon, and Graham Neubig. 2024. Webarena: A realistic web environment for building autonomous agents. *CoRR*, abs/2307.13854.

Zhaocheng Zhu, Mikhail Galkin, Zuobai Zhang, and Jian Tang. 2022. Neural-symbolic models for logical queries on knowledge graphs. In *International Conference on Machine Learning, ICML 2022, 17-23 July 2022, Baltimore, Maryland, USA*, volume 162 of *Proceedings of Machine Learning Research*, pages 27454–27478. PMLR.

## A AI Use Statement

We used AI tools solely to assist with writing refinement (e.g., polishing and editing). In addition, during dataset construction, we used a large language model to automatically generate natural-language versions of cypher queries. All instances where large language models were used in the methodology or dataset pipeline are explicitly disclosed in the main text.

## B License and Data Usage Statement

The raw data sources used to construct NGDBENCH are released under permissive licenses, including the MIT License and the Apache License 2.0. NGDBENCH is a curated benchmark derived from these sources through schema normalization, graph construction, perturbation generation, and query annotation. These transformations are permitted under the corresponding licenses, and the resulting benchmark is released in compliance with their terms. We do not claim ownership of the original raw data sources, and users should follow the original licenses when redistributing, modifying, or extending NGDBENCH.

## C Details of Perturbation Generation

We design the perturbation module to approximate common data quality problems in the environments agents operate over. The goal is to evaluate whether a neural database can still answer queries and manage state when the observed graph differs from the latent graph because of missing records, extraction errors, schema ambiguity, or spurious links. Table 6 summarizes the perturbation families used in NGDBENCH and their default ratios. The default ratios are set according to Table 7.

Our perturbation design is motivated by four broad categories of real-world noise. First, incompleteness and missing facts are common in both knowledge graphs and database integration pipelines (Yang et al., 2022; Shi and Weninger, 2018; Guo et al., 2023). Missing links may arise during extraction, digitization, or data fusion, and are widely recognized as a core challenge under the open-world assumption (Yang et al., 2022; Shi and Weninger, 2018). This is captured in NGDBENCH by *incomplete_edges*. In NGD-Fin, for example, we remove selected edges between companies and loans to simulate omitted financial records or missing loan agreements in corporate filings and transaction logs (Bryzgalova et al., 2025).

Second, erroneous attribute values are pervasive in practice, arising from human data-entry mistakes, OCR failures, sensor malfunctions, transmission errors, and normalization inconsistencies (Guo et al., 2023; Althubaiti, 2016). These issues are reflected in our *attribute_noise* family. In NGD-Prime, for instance, we perturb the drug name *Ornipressin* to *Ornjpressin*, which mimics common spelling errors in medical records and drug information systems (Senger et al., 2010).

Third, schema and type inconsistency frequently appears in noisy graph construction, especially when data are crowdsourced, automatically extracted, or integrated across heterogeneous ontologies (Jarnac et al., 2025; Zhao and Takeda, 2026).

Table 6: Perturbation families used in NGDBENCH and their default injection ratios.

| Noise Family | What It Simulates | Typical Manifestation | Default Ratio |
|---|---|---|---|
| Incomplete edges | Missing facts or partial records | Removed relations in transactional, social, or biomedical graphs | 0.10 |
| False edges | Spurious joins or extraction mistakes | Incorrect links introduced by integration or inference pipelines | 0.15 |
| Relation-type noise | Ambiguous relation semantics | A valid edge assigned the wrong relation label | 0.20 |
| Node-type noise | Entity typing ambiguity | A node assigned an incorrect semantic type | 0.20 |
| Attribute noise | Local textual or numerical corruption | Typos, OCR artifacts, malformed values, or numeric deviations | 0.25 |

Typical manifestations include incorrect type labels, relation-type confusion, and semantic inconsistency between concept hierarchies. These issues are captured through *relation_type_noise* and *node_type_noise*. In NGD-Prime, for example, the entity *Recurrent fractures* can be relabeled from *effect/phenotype* to *exposure*, simulating domain-specific misclassification caused by conceptual ambiguity in collaborative biomedical knowledge sources (Zhao and Takeda, 2026; Althubaiti, 2016).

Fourth, erroneous links are common in information extraction and data fusion pipelines, where systems may incorrectly join entities or create spurious relationships from ambiguous context (Jarnac et al., 2025). This corresponds to our *false_edges* perturbation family. In NGD-BI, for example, we insert spurious *Comment_isLocatedIn_Country* edges between comments and unrelated countries to mimic realistic social-media analytics errors such as incorrect geolocation or ambiguous entity matching.

During the design of this module, we also surveyed reported noise characteristics in real-world datasets and synthetic noise settings used in prior graph denoising and refinement work (Yang et al., 2022; Shi and Weninger, 2018; Guo et al., 2023; Qian et al., 2023; Bryzgalova et al., 2025). Table 7 summarizes the main ranges we observed.

Two observations motivate our design. First, real-world noise is highly variable and often skewed across noise families, so a single fixed ratio cannot capture all realistic settings. Second, prior denoising work typically evaluates only a small set of synthetic noise levels. We therefore expose perturbation as a configurable environment generator rather than commit to one universal corruption pattern.

Our default configuration sets *incomplete_edges* to 0.10, *false_edges* to 0.15, *relation_type_noise* and *node_type_noise* to 0.20, and *attribute_noise* to 0.25. These values fall within the ranges summarized in Table 7 and define a moderate baseline setting that is neither trivial nor excessively destructive.

# D Details of Query Generation Module

## D.1 Query Operators Supported in NGDBENCH

The operators used in  is a minimized set covering all Cypher functionalities while ensuring runtime compatibility. For instance, the *HashJoin* operator acts as a unified abstraction for variants like *Node Hash Join*, *Value Hash Join*, and *Node Left-/Right Outer Hash Join*. Operators strictly for performance optimization, such as *Cache Properties*, were excluded as they do not influence query logic. The operators are shown in Table 8.

## D.2 Analytical Query Generation

The generation of our analytical query templates is structured into three progressive phases. The detailed workflow and corresponding example query templates for each phase are presented in Sections B2.2, B2.3, and B2.4, respectively.

### D.2.1 Query Template Design Principle

When designing our query generation module, we surveyed Cypher and practical graph-database workloads such as LDBC FinBench. We found that, although production queries are diverse, they are typically compositional: complex queries can be decomposed into a relatively small set of core operators and recurring query patterns. Neo4j's documentation likewise presents Cypher as compositions of clauses and execution steps.

We therefore extracted 29 core functional operators and constructed templates that cover the main basic and complex query families, including chain, variable-length traversal, join, pipeline, anti-pattern, union, triadic, and aggregation patterns,

Table 7: Survey summary of noise characteristics reported in real-world datasets and prior synthetic-noise studies, compiled from (Yang et al., 2022; Shi and Weninger, 2018; Guo et al., 2023; Qian et al., 2023; Bryzgalova et al., 2025). The ranges show that realistic noise levels vary substantially across domains and noise families.

| Source | Erroneous Attribute | Incompleteness | Type Inconsistency | Erroneous Links |
|---|---|---|---|---|
| Real-world datasets | 0.55%–26.9% in reported medical settings | Over 70% missing *born_in*; about 90% missing *ethnicity* in Freebase-style settings | 8.0%–38.5% across reported semantic domains | 10%–25% in reported financial settings |
| Prior synthetic injection | 10%–50% | 2.9% and 10.4% in representative studies | 10%–50% | 10%, 20%, and 30% |

Table 8: Classification and descriptions of the supported Cypher core operators serving as the benchmark's functional primitives.

| Category | Core Operators | Function Description |
|---|---|---|
| **Data Access** | `Scan`, `Seek` | Entry operators (e.g., node scan, index seek). |
| **Traversal & Matching** | `Expand`, `VarLengthExpand`, `ShortestPath`, `Repeat`, `Optional`, `Anti`, `Triadic` | Topology navigation, variable-length paths, and pattern matching (incl. anti/optional). |
| **Join & Set Operations** | `HashJoin`, `Apply`, `CartesianProduct`, `Argument`, `Union` | Combine intermediate results (hash / nested loop); includes unions. |
| **Filter & Projection** | `Filter`, `Project`, `Distinct` | Predicates, expressions, field selection, and deduplication. |
| **Aggregation & Control** | `Aggregate`, `Unwind`, `Sort`, `Limit`, `Skip`, `Top` | Grouping, list flattening, and output control (order/limit). |
| **Data Modification** | `Create`, `Merge`, `Delete`, `Set`, `Foreach` | Graph mutations: creation, deletion, property updates. |

rather than attempting to enumerate every production query verbatim. Our template library directly reflects this coverage.

For example, consider the following production-style query:

```
MATCH (person:Person {id:8796093022538})
    -[edge1:own]->(pAcc:Account)
      -[edge2:transfer]->(compAcc:
          Account)<-[edge3:own]-(com:
          Company)
WHERE edge2.timestamp > 1635724800000
    AND edge2.timestamp < 1638144000000
WITH compAcc.id AS compAccountId, sum(
    edge2.amount) AS sumEdge2Amount
RETURN compAccountId, round(
    sumEdge2Amount * 1000) / 1000 as
    sumEdge2Amount
ORDER BY sumEdge2Amount DESC;
```

Although this query appears complex as a whole, it can still be decomposed into the same building blocks already covered in NGDBench, such as *edge-property filtering*, a pipeline stage using `WITH`, *aggregation* via `sum`, *projection/post-processing* in `RETURN`, and *ordering* over the aggregated result.

As a result, queries seen in production can often be decomposed into the operator- and pattern-level building blocks already covered by NGDBench.

#### D.2.2 Phase 1: Basic Template Synthesis

**Description:** This phase utilizes core operators (`Scan`, `Seek`, `Expand`, `Projection`.) to construct the most fundamental query skeletons. These templates contain placeholders (e.g., `$LABEL`, `$PROP`) intended for subsequent parameter instantiation, corresponding to the *Abstract Prototypes* in the algorithm.

Listing 1: Phase 1 Examples: Basic Lookup and Traversal

```
// Example 1: Basic Node Lookup
// Operators: Scan + Filter + Project
MATCH (n:$LABEL)
WHERE n.$PROP = $VALUE
RETURN n

// Example 2: 1-Hop Traversal
// Operators: Expand + Match
MATCH (n:$LABEL)-[:$REL_TYPE]->(m)
RETURN m
```

#### D.2.3 Phase 2: Complexity Augmentation (NoAgg Set)

**Description:** In this phase, basic templates are extended to include more complex logic without aggregation. This includes Path Logic (`VarLengthExpand`, `Triadic`), Relational Logic

(UNION, Cartesian Product, HashJoin, Repeat), and complex filtering (Distinct, Unwind, Apply). The generated templates belong to the $\mathcal{T}_{NoAgg}$ set.

Listing 2: Phase 2 Examples: Path Logic and Set Operations

```
// Example 1: Variable-Length Path with
    Filtering
// Operators: VarLengthExpand + Filter
MATCH p =
    (n:$LABEL)-[:$REL_TYPE*1..3]->(m)
WHERE n.age > $AGE_LIMIT AND m.status =
    'Active'
RETURN p

// Example 2: Union Query
// Operators: Union
MATCH (n:$LABEL_A) WHERE n.type =
    'Gold' RETURN n
UNION
MATCH (n:$LABEL_B) WHERE n.credit >
    $CREDIT RETURN n
```

#### D.2.4 Phase 3: Aggregation Extension (Agg Set)

**Description:** This phase builds upon the complex templates from Phase 2 by adding aggregation operators (such as Sum, Min, Max, Avg, Count). This allows for testing the system's performance when handling grouping and statistical calculations.

Listing 3: Phase 3 Examples: Statistical Aggregation

```
// Example 1: Neighbor Count Statistics
// Operators: Count + Group By (Implicit)
MATCH (n:$LABEL)-[:$REL_TYPE]->(m)
RETURN n.id, count(m) AS degree

// Example 2: Average Property Calculation
// Operators: Avg + Filter
MATCH (n:$LABEL)
WHERE n.created_year > $YEAR
RETURN avg(n.salary) AS avg_salary
```

#### D.2.5 Boolean Query

We filter the generated templates to identify those utilizing the *Collect* or *Cartesian Product* operator, which typically yield multi-valued list results. For each identified template (Positive Template), we synthesize a corresponding Inverse Query Template (e.g., using negation or anti-pattern logic). These are coupled to form a Positive-Inverse pair, creating a dedicated set for validating query engine consistency. For instance, considering a scenario involving a Cartesian Product between Person nodes based on their IP locations, the generated pair is illustrated as follows:

- **Positive Template ($t_{pos}$):** Retrieves pairs of distinct persons sharing the *same* location IP.

Listing 4: Template Examples: Positive vs Inverse Conditions

```
// Positive Template (t_pos)
// Retrieves pairs of distinct persons
    sharing the same location IP
MATCH (a:Person), (b:Person)
WHERE a.locationIP = b.locationIP AND a <>
    b
WITH a, b
RETURN a, collect(b.birthday)[0..1] AS bs

// Inverse Template (t_inv)
// Retrieves pairs of distinct persons
    with different location IPs
MATCH (a:Person), (b:Person)
WHERE a.locationIP <> b.locationIP AND a
    <> b
WITH a, b
RETURN a, collect(b.birthday)[0..1] AS bs
```

.

```
MATCH (a:Person), (b:Person)
WHERE a.locationIP <> b.locationIP
     AND a <> b
WITH a, b
RETURN a, collect(b.birthday)
    [0..1] AS bs
```

### D.3 Management Queries

We developed the **Management Query Template Library** to evaluate the system's transactional capabilities and data integrity under mutation. The generation workflow proceeds from Template Construction to Query Sampling:

1. **Template Construction**: We first define parameterized query skeletons (e.g., CREATE (n:Person id: $id)) covering standard graph mutation logic.

2. **Parameter Instantiation**: To generate executable queries, we sample concrete values for these parameters. Data generators inject valid IDs and attributes to ensure the queries are semantically valid against the current graph schema.

3. **Batch Assembly**: Finally, these instantiated query instances are organized into Batches. Each batch represents a logical unit of work designed to simulate real-world transactional sequences.

A single batch is strictly structured as a sequence of $K$ steps (standard configuration $K = 5$), with each step constituting an Operation-Validation Pair. This pair consists of an Operation Template, which

executes a mutation using operators including Create , Delete, Set, and Remove, to modify the graph state, and a corresponding Validation Template, a read-only query designed to immediately verify the effect of the preceding operation (e.g., confirming a node deletion).

**Operation Template** ($top$): A mutation query template utilizing operators such as Create, Delete, Set, or Remove. This template modifies the graph state.

**Validation Template** ($tval$): A corresponding read-only query template designed to verify the immediate effect of the preceding operation.

**Uniform Operation Batches** These batches consist of repeated executions of the same operator type to test specific write throughput and stability.

Listing 5: Uniform Batch Example

```
// Management Queries
CREATE (n:Person {city: 'Kaohsiung'})
     -[:Guarantee]->
     (g:Person {id: 21..})

CREATE (n:Person {city: 'Hanover'})
     -[:Guarantee]->
     (g:Person {id: 50})

CREATE (n:Person {city: 'Hamamatsu'})
     -[:Guarantee]->
     (g:Person {id: 21..})

CREATE (n:Person {city: 'Shanghai'})
     -[:Guarantee]->
     (g:Person {id: 911})

CREATE (n:Person {city: 'Manipal'})
     -[:Guarantee]->
     (g:Person {id: 21..})

// Validation Query
MATCH (g:Person)<-[:Guarantee]-(n:Person)
RETURN count(n)
```

*Explanation:* The management queries sequentially insert five new Guarantee relationships between Person nodes from different cities. The validation query counts the total number of such relationships to verify that all five insertions were successfully persisted.

**Mixed Operation Batches** These batches interleave different types of operators (e.g., insertion followed by deletion) to evaluate the system's handling of complex transactional logic and isolation.

Listing 6: Mixed Batch Example

```
// Management Queries
CREATE (n:TagClass {name: 'SportsMember'})
CREATE (n:TagClass {name: 'Company'})
MATCH (n:TagClass {id: 'SportsMember'})
DETACH DELETE n
CREATE (n:TagClass {name: 'Country'})
CREATE (n:TagClass {name: 'MemberOfParliament'})
MATCH (n:TagClass {id: 'Country'})
DETACH DELETE n

// Validation Query
MATCH (n:Post)
RETURN count(n) AS cnt
```

*Explanation:* This sequence performs a "create-then-delete" stress test. It creates four new TagClass nodes but immediately deletes two of them ('SportsMember' and 'Country') within the same batch. The validation query checks the count of Post nodes, serving as a background integrity check to ensure these schema modifications do not corrupt unrelated data entities.

# E Full Query Templates

## E.1 No-aggregation Template

This section lists the full non-aggregation analytical query templates used to instantiate the $\mathcal{T}_{\text{NoAgg}}$ library. We group templates by operator pattern to make the construction process easier to inspect.

### E.1.1 Basic Templates

```
T001:
MATCH (n:$LABEL {$PROP: $VALUE})-[:
    $REL_TYPE]->(m)
RETURN m

T002:
MATCH (a:$LABEL1)-[:$REL1]->(b:$LABEL2
    )
MATCH (a)-[:$REL2]->(b)
RETURN a, b

T003:
MATCH (a:$LABEL1)-[:$REL]-(b:$LABEL2)
WHERE b.$PROP $OP $VALUE
RETURN a
```

### E.1.2 Anti Templates

```
T001:
MATCH (a:$LABEL1 {$PROP1: $VALUE})-[:
    $REL_TYPE]-()-[:$REL_TYPE]-(b:
    $LABEL2)
WHERE NOT (a)-[:$REL_TYPE]-(b)
RETURN b.$RETURN_PROP

T002:
MATCH (a:$LABEL)-[:$REL2]->(:$LABEL3)
WHERE NOT (a)-[:$REL1]->(:$LABEL2)
RETURN a.$RET_PROP
```

### E.1.3 Union Templates

Listing 7: Union Templates

```
T001:
MATCH (a:$LABEL1 {$PROP1: $VALUE})
RETURN a.$PROP1
UNION
MATCH (b:$LABEL2 {$PROP2: $VALUE2})
RETURN b.$PROP2
```

### E.1.4 Chain Templates

Listing 8: Chain and Path Templates

```
T001:
MATCH (a:$LABEL1)-[:$REL1]->(b:$LABEL2
    )-[:$REL2]->(c:$LABEL3)-[:$REL3
    ]->(d:$LABEL4)
RETURN a, b, d

T002:
MATCH (a:$LABEL1)-[:$REL_TYPE*
    $MIN_HOPS..$MAX_HOPS]-(b:$LABEL2)
RETURN a, b

T003:
MATCH (start:$START_LABEL {$START_PROP
    : $START_VALUE})
MATCH p = (start)
  ((n1:$L1)<-$D1[:$R1]-$D2(r:
    $REL_NODE_LABEL)-$D3[:$R2]->(n2:
    $L2)
  WHERE r.$NODE_PROP = $NODE_VALUE){
    $MIN_HOPS..$MAX_HOPS}
  (end:$END_LABEL)
RETURN p
```

### E.1.5 Pipelined Templates

Listing 9: Pipelined Templates with `WITH`

```
T001:
MATCH (a:$LABEL1 {$PROP1: $VALUE})-[:
    $REL1]->(b:$LABEL2)
WITH b
MATCH (b)-[:$REL2]->(c:$LABEL3)
RETURN b.$RET_PROP1, c.$RET_PROP2

T002:
MATCH (a:$LABEL1 {$PROP1: $VALUE})-[:
    $REL1]->(b:$LABEL2)
WITH b
MATCH (b)-[:$REL2]->(c:$LABEL3)
WITH b, c
MATCH (c)-[:$REL3]->(d:$LABEL4)
RETURN b.$RET_PROP1, c.$RET_PROP2, d.
    $RET_PROP3

T003:
MATCH (a:$LABEL1 {$PROP1: $VALUE})-[:
    $REL1]->(b:$LABEL2)
WITH b
MATCH (b)-[:$REL2]->(c:$LABEL3)
WHERE c.$FILTER_PROP = $FILTER_VAL
WITH b, c
MATCH (c)-[:$REL3]->(d:$LABEL4)
RETURN b.$RET_PROP1, c.$RET_PROP2, d.
    $RET_PROP3
```

### E.1.6 Nested-Loop Templates

Listing 10: Nested-Loop and Predicate Templates

```
T001:
MATCH (a:$LABEL1 {$PROP1: $VALUE})
MATCH (b:$LABEL2 {$PROP2: a.$REF_PROP
    })
WHERE a <> b
RETURN a, b

T002:
MATCH (a:$LABEL1)
WHERE (a)-[:$REL_TYPE]->(:$LABEL2)
RETURN a.$RETURN_PROP

T004:
MATCH (n:$LABEL)
WHERE (n)-[:$REL1]->(:$L1) OR (n)-[:
    $REL2]->(:$L2)
RETURN n.$PROP

T005:
MATCH (n:$LABEL)
WHERE NOT (n)-[:$REL1]->(:$L1) OR (n)
    -[:$REL2]->(:$L2)
RETURN n.$PROP

T006:
MATCH (n:$LABEL)
WHERE n.$PROP $OP $VAL OR (n)-[:$REL
    ]->(:$L2)
RETURN n.$RET
```

## E.2 Aggregation Template

### E.2.1 Basic

```
T001:
MATCH (n:$LABEL) RETURN sum(n)
```

### E.2.2 Basic Aggregation

```
T001:
MATCH (a:$LABEL) RETURN avg(a.$PROP)
    AS avg_value

T003:
MATCH (a:$LABEL) RETURN sum(a.$PROP)
    AS total_value

T004:
MATCH (a:$LABEL) RETURN min(a.$PROP)
    AS min_value

T005:
```

```
MATCH (a:$LABEL) RETURN max(a.$PROP)
    AS max_value
```

### E.2.3 Basic Expand Aggregation

```
T001:
MATCH (n:$LABEL {$PROP: $VALUE})-[:
    $REL_TYPE]->(m) RETURN count(m) AS
     cnt

T002:
MATCH (n:$LABEL {$PROP: $VALUE})-[:
    $REL_TYPE]->(m:$LABEL2) RETURN avg
    (m.$RET_PROP2) AS avg_value

T003:
MATCH (n:$LABEL {$PROP: $VALUE})-[:
    $REL_TYPE]->(m:$LABEL2) RETURN sum
    (m.$RET_PROP2) AS total
```

### E.2.4 Basic Expand Into Aggregation

```
T001:
MATCH (a:$LABEL1)-[:$REL1]->(b:$LABEL2
    ) MATCH (a)-[:$REL2]->(b) RETURN
    count(*) AS cnt

T002:
MATCH (a:$LABEL1)-[:$REL1]->(b:$LABEL2
    ) MATCH (a)-[:$REL2]->(b) RETURN a
    , count(b) AS cnt
```

### E.2.5 Basic Filter Aggregation

```
T001:
MATCH (a:$LABEL1)-[r:$REL_TYPE]-(b:
    $LABEL2) WHERE r.$REL_PROP $OP
    $VALUE RETURN count(a) AS cnt

T002:
MATCH (a:$LABEL1)-[r:$REL_TYPE]-(b:
    $LABEL2) WHERE r.$REL_PROP $OP
    $VALUE RETURN a, count(b) AS cnt

T003:
MATCH (a:$LABEL1)-[r:$REL_TYPE]-(b:
    $LABEL2) WHERE r.$REL_PROP $OP
    $VALUE RETURN avg(r.$REL_PROP),
    max(r.$REL_PROP)
```

### E.2.6 Nested Loop

```
T007:
MATCH (a:$LABEL1 {$PROP1: $VALUE})
    MATCH (b:$LABEL2 {$PROP2: a.
    $REF_PROP}) RETURN count(b) AS cnt

T009:
MATCH (a:$LABEL1) WHERE (a)-[:
    $REL_TYPE]->(:$LABEL2) RETURN
    count(a) AS cnt

T011:
MATCH (a:$LABEL1 {$PROP1: $VALUE}), (b
    :$LABEL2) WHERE NOT (a)-[:
    $REL_TYPE]->(b) RETURN count(b) AS
     cnt

T012:
MATCH (n:$LABEL) WHERE (n)-[:$REL1
    ]->(:$LABEL2) OR (n)-[:$REL2]->(:
    $LABEL3) RETURN count(n) AS cnt

T013:
MATCH (n:$LABEL) WHERE NOT (n)-[:$REL1
    ]->(:$LABEL2) OR (n)-[:$REL2]->(:
    $LABEL3) RETURN count(n)

T014:
MATCH (n:$LABEL) WHERE n.$PROP $OP
    $VAL OR (n)-[:$REL]->(:$LABEL2)
    RETURN avg(n.$RET)
```

### E.2.7 Chain

```
T004:
MATCH (a:$LABEL1)-[:$REL1]->(b:$LABEL2
    )-[:$REL2]->(c:$LABEL3)-[:$REL3
    ]->(d:$LABEL4) RETURN count(d) AS
    cnt

T005:
MATCH (a:$LABEL1)-[:$REL1]->(b:$LABEL2
    )-[:$REL2]->(c:$LABEL3)-[:$REL3
    ]->(d:$LABEL4) RETURN a, count(d)
    AS cnt

T006:
MATCH (a:$LABEL1)-[:$REL1]->(b:$LABEL2
    )-[:$REL2]->(c:$LABEL3)-[:$REL3
    ]->(d:$LABEL4) RETURN avg(d.$RET)

T008:
MATCH (a:$LABEL1)-[:$REL_TYPE *
    $MIN_HOPS..$MAX_HOPS]-(b:$LABEL2)
    RETURN count(b) AS cnt

T009:
MATCH (a:$LABEL1)-[:$REL_TYPE *
    $MIN_HOPS..$MAX_HOPS]-(b:$LABEL2)
    RETURN a, count(b) AS cnt

T010:
MATCH p = (a:$LABEL1)-[:$REL_TYPE *
    $MIN_HOPS..$MAX_HOPS]-(b:$LABEL2)
    RETURN avg(length(p)) AS avg_len

T011:
MATCH (start:$START_LABEL {$START_PROP
    : $START_VALUE}) MATCH p = (start)
     ((n1:$LABEL2)<-$D1[:$R1]-$D2(r:
    $REL_NODE_LABEL)-$D3[:$R2]->(n2:
    $LABEL3) WHERE r.$NODE_PROP =
    $NODE_VALUE){$MIN_HOPS..$MAX_HOPS}
     (end:$END_LABEL) RETURN count(p)
    AS path_count
```

```
T012:
MATCH (start:$START_LABEL {$START_PROP
    : $START_VALUE}) MATCH p = (start)
     (...){$MIN_HOPS..$MAX_HOPS} (end:
    $END_LABEL) RETURN avg(length(p))
    AS avg_len
```

### E.2.8 Join

```
T004:
MATCH (a:$LABEL1), (b:$LABEL2) WHERE a
    .$PROP1 = b.$PROP2 AND a <> b
    RETURN count(*) AS cnt

T006:
MATCH (a:$LABEL1), (b:$LABEL2) WHERE a
    .$PROP1 = b.$PROP2 AND a <> b WITH
     a, count(b) AS cnt WHERE cnt >
    $THRESHOLD RETURN a
```

### E.2.9 Pipelined

```
T001:
MATCH (a:$LABEL1 {$PROP1: $VALUE})-[:
    $REL1]->(b:$LABEL2) WITH b, count
    (*) AS cnt WHERE cnt > 0 RETURN b.
    $RET_PROP1

T002:
MATCH (a:$LABEL1 {$PROP1: $VALUE})-[:
    $REL1]->(b:$LABEL2) WITH b MATCH (
    b)-[:$REL2]->(c:$LABEL3) WHERE c.
    $FILTER_PROP = $FILTER_VAL RETURN
    count(*) AS total_cnt

T003:
MATCH (a:$LABEL1 {$PROP1: $VALUE})-[:
    $REL1]->(b:$LABEL2) WITH b MATCH (
    b)-[:$REL2]->(c:$LABEL3) WHERE c.
    $FILTER_PROP = $FILTER_VAL RETURN
    b, count(c) AS cnt

T004:
MATCH (a:$LABEL1 {$PROP1: $VALUE})-[:
    $REL1]->(b:$LABEL2) WITH b MATCH (
    b)-[:$REL2]->(c:$LABEL3) WHERE c.
    $FILTER_PROP = $FILTER_VAL WITH b,
     count(c) AS cnt WHERE cnt >
    $THRESHOLD RETURN b, cnt
```

### E.2.10 Anti

```
T004:
MATCH (a:$LABEL) WHERE NOT a.$PROP $OP
     $VALUE RETURN count(a) AS cnt

T006:
MATCH (a:$LABEL1 {$PROP1: $VALUE}), (b
    :$LABEL2) WHERE NOT (a)-[:
    $REL_TYPE]->(b) RETURN count(b) AS
     cnt

T007:
MATCH (a:$LABEL1 {$PROP1: $VALUE}), (b
    :$LABEL2) WHERE NOT (a)-[:
    $REL_TYPE]->(b) RETURN a, count(b)
     AS cnt

T008:
MATCH (a:$LABEL) WHERE NOT (a)-[:$REL1
    ]->(:$LABEL2) OR (a)-[:$REL2]->(:
    $LABEL3) RETURN count(a) AS cnt
```

### E.2.11 Union

```
T002:
MATCH (a:$LABEL1)-[:$REL1]->(b:$LABEL2
    ) WITH a, $AGG_FUNC1(b.$RET_PROP1)
     AS metric1 WHERE metric1 $OP1
    $THRESHOLD1 RETURN a.$PROP1 AS
    EntityName, '$CATEGORY1' AS
    Category, metric1 AS ValueScore
    UNION ALL MATCH (c:$LABEL3)<-[:
    $REL2]-(d:$LABEL4) WITH c,
    $AGG_FUNC2(d.$RET_PROP3) AS
    metric2 WHERE metric2 $OP2
    $THRESHOLD2 RETURN c.$PROP2 AS
    EntityName, '$CATEGORY2' AS
    Category, metric2 AS ValueScore

T003:
MATCH (a:$LABEL1) RETURN count(a.$PROP
    ) AS cnt UNION MATCH (b:$LABEL2)
    RETURN count(b.$PROP) AS cnt
```

### E.2.12 Triadic

```
T002:
MATCH (a:$LABEL1)-[:$REL]-()-[:$REL]-(
    b:$LABEL2) WHERE NOT (a)-[:$REL]-(
    b) WITH a, count(DISTINCT b) AS
    cnt RETURN a.$PROP, cnt
```

## E.3 Judge Templates

This section lists the query templates used for judge-oriented query sampling. Each template is paired with an anti-template when available, so that the instantiated query can be used to compare positive and contrastive graph conditions.

### E.3.1 Basic Aggregation

```
T002:
MATCH (g:$GROUP_LABEL)<-[:$REL]-(n:
    $LABEL) WITH g, n RETURN g.
    $GROUP_PROP AS $GROUP_ALIAS,
    collect(n.$NODE_PROP)[0..1] AS
    $COLLECT_ALIAS
Anti:
MATCH (g:$GROUP_LABEL), (n:$LABEL)
    WHERE NOT (g)<-[:$REL]-(n) WITH g,
     n RETURN g.$GROUP_PROP AS
```

```
    $GROUP_ALIAS, collect(n.$NODE_PROP
    )[0..1] AS $COLLECT_ALIAS

T006:
MATCH (a:$LABEL1 {$PROP1: $VALUE}), (b
    :$LABEL2) WHERE NOT (a)-[:$REL]->(
    b) WITH b ORDER BY b.$PROP_ID
    LIMIT 32 RETURN b.$PROP_ID
Anti:
MATCH (a:$LABEL1 {$PROP1: $VALUE}), (b
    :$LABEL2) WHERE (a)-[:$REL]->(b)
    WITH b ORDER BY b.$PROP_ID LIMIT
    32 RETURN b.$PROP_ID
```

### E.3.2 Nested-Loop

```
T008:
MATCH (a:$LABEL1 {$PROP1: $VALUE})
    MATCH (b:$LABEL2 {$PROP2: a.
    $REF_PROP}) WITH a, b ORDER BY b.
    $PROP_ID RETURN a, collect(b.
    $PROP_ID)[0..1] AS bs
Anti:
MATCH (a:$LABEL1 {$PROP1: $VALUE})
    MATCH (b:$LABEL2) WHERE b.$PROP2
    <> a.$REF_PROP WITH a, b ORDER BY
    b.$PROP_ID RETURN a, collect(b.
    $PROP_ID)[0..1] AS bs

T010:
MATCH (a:$LABEL1) WHERE (a)-[:
    $REL_TYPE]->(:$LABEL2) WITH a
    RETURN collect(a.$RETURN_PROP)
    [0..1] AS values
Anti:
MATCH (a:$LABEL1) WHERE NOT (a)-[:
    $REL_TYPE]->(:$LABEL2) WITH a
    RETURN collect(a.$RETURN_PROP)
    [0..1] AS values
```

### E.3.3 Chain

```
T007:
MATCH (a:$LABEL1)-[:$REL1]->(b:$LABEL2
    )-[:$REL2]->(c:$LABEL3)-[:$REL3
    ]->(d:$LABEL4) WITH a, d RETURN a,
     collect(d.$PROP)[0..1] AS ds
Anti:
MATCH (a:$LABEL1), (d:$LABEL4) WHERE
    NOT (a)-[:$REL1]->(b:$LABEL2)-[:
    $REL2]->(c:$LABEL3)-[:$REL3]->(d)
    WITH a, d RETURN a, collect(d.
    $PROP)[0..1] AS ds
```

### E.3.4 Join

```
T002:
MATCH (a:$LABEL1), (b:$LABEL2) WHERE a
    .$PROP1 = b.$PROP2 AND a <> b WITH
     a, b RETURN a, b
Anti:
MATCH (a:$LABEL1), (b:$LABEL2) WHERE a
    .$PROP1 <> b.$PROP2 AND a <> b
    WITH a, b RETURN a, b

T005:
MATCH (a:$LABEL1), (b:$LABEL2) WHERE a
    .$PROP1 = b.$PROP2 AND a <> b WITH
     a, b RETURN a, collect(b.$PROP)
    [0..1] AS bs
Anti:
MATCH (a:$LABEL1), (b:$LABEL2) WHERE a
    .$PROP1 <> b.$PROP2 AND a <> b
    WITH a, b RETURN a, collect(b.
    $PROP)[0..1] AS bs
```

### E.3.5 Anti

```
T001:
MATCH (a:$LABEL) WHERE NOT a.$PROP $OP
     $VALUE WITH a ORDER BY a.$PROP_ID
     LIMIT 10 RETURN a.$PROP_ID
Anti:
MATCH (a:$LABEL) WHERE a.$PROP $OP
    $VALUE WITH a ORDER BY a.$PROP_ID
    LIMIT 10 RETURN a.$PROP_ID

T005:
MATCH (a:$LABEL) WHERE NOT a.$PROP $OP
     $VALUE WITH a RETURN collect(a.
    $RET_PROP)[0..1] AS vals
Anti:
MATCH (a:$LABEL) WHERE a.$PROP $OP
    $VALUE WITH a RETURN collect(a.
    $RET_PROP)[0..1] AS vals

T009:
MATCH (a:$LABEL) WHERE NOT (a)-[:$REL1
    ]->(:$LABEL2) OR (a)-[:$REL2]->(:
    $LABEL3) WITH a RETURN collect(a.
    $RET_PROP)[0..1] AS vals
```

### E.3.6 Triadic

```
T001:
MATCH (a:$LABEL1)-[:$REL]-()-[:$REL]-(
    b:$LABEL2) WHERE NOT (a)-[:$REL]-(
    b) WITH b ORDER BY b.$PROP_ID
    LIMIT 10 RETURN b.$PROP_ID
Anti:
MATCH (a:$LABEL1)-[:$REL]-()-[:$REL]-(
    b:$LABEL2) WHERE (a)-[:$REL]-(b)
    WITH b ORDER BY b.$PROP_ID LIMIT
    10 RETURN b.$PROP_ID
```

## E.4 Management Templates

This section lists the management-oriented templates used to instantiate graph update queries. Each template contains a write operation together with pre-validation and post-validation analytical queries, enabling us to test whether graph updates

produce the expected observable changes in downstream query results.

### E.4.1 Create Templates

```
T001:
Pre-validation:
MATCH (a:$LABEL) RETURN avg(a.$PROP)
    AS avg_value
Update:
CREATE (n:$LABEL {$PROP: $VALUE})
Post-validation:
MATCH (a:$LABEL) RETURN avg(a.$PROP)
    AS avg_value

T002:
Pre-validation:
MATCH (g:$GROUP_LABEL)<-[:$REL]-(n:
    $LABEL) RETURN g.$GROUP_PROP,
    count(n) AS cnt
Update:
CREATE (n:$LABEL {$PROP: $NAME})-[:
    $REL]->(g:$GROUP_LABEL {$PROP_ID:
    $GID})
Post-validation:
MATCH (g:$GROUP_LABEL)<-[:$REL]-(n:
    $LABEL) RETURN g.$GROUP_PROP,
    count(n) AS cnt

T003:
Pre-validation:
MATCH (a:$L1)-[:$R1]->(b:$L2)-[:$R2
    ]->(c:$L3) RETURN count(*) AS cnt
Update:
MATCH (a:$L1 {$PROP_ID1: $ID1}), (b:
    $L2 {$PROP_ID2: $ID2}) CREATE (a)
    -[:$R1]->(b)
Post-validation:
MATCH (a:$L1)-[:$R1]->(b:$L2)-[:$R2
    ]->(c:$L3) RETURN count(*) AS cnt

T004:
Pre-validation:
MATCH (a:$LABEL1)-[r:$REL_TYPE]-(b:
    $LABEL2) WHERE r.$REL_PROP > $VAL
    RETURN count(a) AS cnt
Update:
CREATE (a:$LABEL1)-[r:$REL_TYPE {
    $REL_PROP: $NEW_VAL}]->(b:$LABEL2
    {$PROP_ID: $BID})
Post-validation:
MATCH (a:$LABEL1)-[r:$REL_TYPE]-(b:
    $LABEL2) WHERE r.$REL_PROP > $VAL
    RETURN count(a) AS cnt

T005:
Pre-validation:
MATCH (a:$L1) MATCH (b:$L2) WHERE a.
    $P1 = b.$P2 RETURN count(b) AS cnt
Update:
CREATE (n:$L1 {$P1: $VAL})
Post-validation:
MATCH (a:$L1) MATCH (b:$L2) WHERE a.
    $P1 = b.$P2 RETURN count(b) AS cnt

T006:
Pre-validation:
MATCH p = (a:$L1)-[:$R *$MIN_HOPS..
    $MAX_HOPS]-(b:$L2) RETURN avg(
    length(p)) AS avg_len
Update:
MATCH (a:$L1 {$PROP_ID1: $ID1}), (b:
    $L2 {$PROP_ID2: $ID2}) CREATE (a)
    -[:$R]->(b)
Post-validation:
MATCH p = (a:$L1)-[:$R *$MIN_HOPS..
    $MAX_HOPS]-(b:$L2) RETURN avg(
    length(p)) AS avg_len

T007:
Pre-validation:
MATCH (g:$GL {$PROP_ID: $GID})<-[:$R
    ]-(n:$L {$P: $V}) RETURN count(n)
    AS exists
Update:
CREATE (n:$L {$P: $V})-[:$R]->(g:$GL {
    $PROP_ID: $GID})
Post-validation:
MATCH (g:$GL {$PROP_ID: $GID})<-[:$R
    ]-(n:$L {$P: $V}) RETURN count(n)
    AS exists

T008:
Pre-validation:
MATCH (a:$L1), (b:$L2) WHERE a.$P1 = b
    .$P2 AND a <> b RETURN count(*) AS
     cnt
Update:
CREATE (n:$L1 {$P1: $VAL})
Post-validation:
MATCH (a:$L1), (b:$L2) WHERE a.$P1 = b
    .$P2 AND a <> b RETURN count(*) AS
     cnt

T009:
Pre-validation:
MATCH (a:$L1 {$P1: $V}), (b:$L2) WHERE
     NOT (a)-[:$R]->(b) RETURN count(b
    ) AS cnt
Update:
MATCH (a:$L1 {$PROP_ID1: $AID}), (b:
    $L2 {$PROP_ID2: $BID}) CREATE (a)
    -[:$R]->(b)
Post-validation:
MATCH (a:$L1 {$P1: $V}), (b:$L2) WHERE
     NOT (a)-[:$R]->(b) RETURN count(b
    ) AS cnt

T010:
Pre-validation:
MATCH (a:$L1)-[:$R1]->(b:$L2)-[:$R2
    ]->(c:$L3)-[:$R3]->(d:$L4) RETURN
    avg(d.$NP) AS final_avg
Update:
CREATE (n:$L4 {$NP: $VAL})<-[:$R3]-(c:
    $L3 {$PROP_ID: $CID})
Post-validation:
```

```
MATCH (a:$L1)-[:$R1]->(b:$L2)-[:$R2
    ]->(c:$L3)-[:$R3]->(d:$L4) RETURN
    avg(d.$NP) AS final_avg

T011:
Pre-validation:
MATCH (a:$L2) RETURN sum(a.$P2) AS
    total
Update:
CREATE (n:$L1 {$PROP: $VALUE})
Post-validation:
MATCH (a:$L2) RETURN sum(a.$P2) AS
    total

T012:
Pre-validation:
MATCH (a:$L3)-[:$R2]->(b:$L4) RETURN
    count(*) AS cnt
Update:
MATCH (a:$L1 {$PROP_ID1: $ID1}), (b:
    $L2 {$PROP_ID2: $ID2}) CREATE (a)
    -[:$R1]->(b)
Post-validation:
MATCH (a:$L3)-[:$R2]->(b:$L4) RETURN
    count(*) AS cnt

T013:
Pre-validation:
MATCH (a:$L2) RETURN avg(a.$P2) AS
    avg_val
Update:
CREATE (n:$L1 {$P1: $VAL})
Post-validation:
MATCH (a:$L2) RETURN avg(a.$P2) AS
    avg_val
```

### E.4.2 Merge Templates

```
T001:
Pre-validation:
MATCH (a:$LABEL) RETURN sum(a.$PROP)
    AS total
Update:
MERGE (n:$LABEL {$PROP_ID: $VALUE})
    SET n.$PROP = $VAL
Post-validation:
MATCH (a:$LABEL) RETURN sum(a.$PROP)
    AS total

T002:
Pre-validation:
MATCH (n:$L1 {$P: $V})-[:$R]->(m)
    RETURN count(m) AS cnt
Update:
MATCH (a:$L1 {$PROP_ID1: $ID1}), (b:
    $L2 {$PROP_ID2: $ID2}) MERGE (a)
    -[:$R]->(b)
Post-validation:
MATCH (n:$L1 {$P: $V})-[:$R]->(m)
    RETURN count(m) AS cnt

T003:
Pre-validation:
MATCH (a:$L1)-[:$R1]->(b:$L2) MATCH (a
    )-[:$R2]->(b) RETURN count(*) AS
    cnt
Update:
MATCH (a:$L1 {$PROP_ID1: $ID1}), (b:
    $L2 {$PROP_ID2: $ID2}) MERGE (a)
    -[:$R2]->(b)
Post-validation:
MATCH (a:$L1)-[:$R1]->(b:$L2) MATCH (a
    )-[:$R2]->(b) RETURN count(*) AS
    cnt

T004:
Pre-validation:
MATCH (a:$L1 {$P1: $V}) MATCH (b:$L2 {
    $P2: a.$P1}) RETURN count(b) AS
    cnt
Update:
MERGE (n:$L1 {$PROP_ID: $VALUE}) SET n
    .$P1 = $NEW_REF
Post-validation:
MATCH (a:$L1 {$P1: $V}) MATCH (b:$L2 {
    $P2: a.$P1}) RETURN count(b) AS
    cnt

T005:
Pre-validation:
MATCH (n:$L) WHERE (n)-[:$R1]->(:$L1)
    OR (n)-[:$R2]->(:$L2) RETURN count
    (n) AS cnt
Update:
MATCH (n:$L {$PROP_ID: $VALUE}), (m:
    $L2 {$PROP_ID: $MID}) MERGE (n)-[:
    $R2]->(m)
Post-validation:
MATCH (n:$L) WHERE (n)-[:$R1]->(:$L1)
    OR (n)-[:$R2]->(:$L2) RETURN count
    (n) AS cnt

T006:
Pre-validation:
MATCH (a:$L1)-[:$R1]->(b:$L2)-[:$R2
    ]->(c:$L3) RETURN avg(c.$P) AS
    avg_val
Update:
MERGE (n:$L3 {$PROP_ID: $VALUE}) SET n
    .$P = $VAL
Post-validation:
MATCH (a:$L1)-[:$R1]->(b:$L2)-[:$R2
    ]->(c:$L3) RETURN avg(c.$P) AS
    avg_val

T007:
Pre-validation:
MATCH (a:$L1) CALL {WITH a MATCH (b:
    $L2) WHERE a.$P1 = b.$P2 RETURN b.
    $BP AS val ORDER BY b.$BP DESC
    LIMIT 10} RETURN a, val
Update:
MERGE (n:$L2 {$PROP_ID: $VALUE}) SET n
    .$BP = $NEW_VAL
Post-validation:
MATCH (a:$L1) CALL {WITH a MATCH (b:
    $L2) WHERE a.$P1 = b.$P2 RETURN b.
```

```
    $BP AS val ORDER BY b.$BP DESC
    LIMIT 10} RETURN a, val

T008:
Pre-validation:
MATCH (a:$L1)-[:$R *$MIN_HOPS..
    $MAX_HOPS]-(b:$L2) RETURN count(b)
     AS cnt
Update:
MATCH (a:$L1 {$PROP_ID1: $ID1}), (b:
    $L2 {$PROP_ID2: $ID2}) MERGE (a)
    -[:$R]->(b)
Post-validation:
MATCH (a:$L1)-[:$R *$MIN_HOPS..
    $MAX_HOPS]-(b:$L2) RETURN count(b)
     AS cnt

T009:
Pre-validation:
MATCH (a:$L1), (b:$L2) WHERE a.$P1 = b
    .$P2 WITH a, count(b) AS cnt WHERE
     cnt > $THRESHOLD RETURN a
Update:
MERGE (n:$L2 {$PROP_ID: $VALUE, $P2:
    $REF_VAL})
Post-validation:
MATCH (a:$L1), (b:$L2) WHERE a.$P1 = b
    .$P2 WITH a, count(b) AS cnt WHERE
     cnt > $THRESHOLD RETURN a

T010:
Pre-validation:
MATCH (s:$SL {$SP: $SV}) MATCH p = (s)
     ((n1:$L1)<-[:$R1]-(r:$RL)-[:$R2
    ]->(n2:$L2) WHERE r.$NP = $NV){
    $MIN_HOPS..$MAX_HOPS} (e:$EL)
    RETURN count(p) AS path_cnt
Update:
MERGE (r:$RL {$PROP_ID: $VALUE}) SET r
    .$NP = $NV
Post-validation:
MATCH (s:$SL {$SP: $SV}) MATCH p = (s)
     ((n1:$L1)<-[:$R1]-(r:$RL)-[:$R2
    ]->(n2:$L2) WHERE r.$NP = $NV){
    $MIN_HOPS..$MAX_HOPS} (e:$EL)
    RETURN count(p) AS path_cnt

T011:
Pre-validation:
MATCH (a:$L2) RETURN max(a.$P2) AS
    max_val
Update:
MERGE (n:$L1 {$PROP_ID: $VALUE}) SET n
    .$P1 = $VAL
Post-validation:
MATCH (a:$L2) RETURN max(a.$P2) AS
    max_val

T012:
Pre-validation:
MATCH p = (a:$L3)-[:$R2 *$MIN_HOPS..
    $MAX_HOPS]-(b:$L4) RETURN avg(
    length(p)) AS avg_len
Update:
MATCH (a:$L1 {$PROP_ID1: $ID1}), (b:
    $L2 {$PROP_ID2: $ID2}) MERGE (a)
    -[:$R1]->(b)
Post-validation:
MATCH p = (a:$L3)-[:$R2 *$MIN_HOPS..
    $MAX_HOPS]-(b:$L4) RETURN avg(
    length(p)) AS avg_len

T013:
Pre-validation:
MATCH (a:$L3), (b:$L4) WHERE a.$P3 = b
    .$P4 RETURN count(*) AS cnt
Update:
MERGE (n:$L1 {$PROP_ID: $VALUE}) SET n
    .$P1 = $VAL
Post-validation:
MATCH (a:$L3), (b:$L4) WHERE a.$P3 = b
    .$P4 RETURN count(*) AS cnt
```

### E.4.3 Delete Templates

```
T001:
Pre-validation:
MATCH (a:$LABEL) RETURN max(a.$PROP)
    AS max_val
Update:
MATCH (n:$LABEL {$PROP_ID: $VALUE})
    DELETE n
Post-validation:
MATCH (a:$LABEL) RETURN max(a.$PROP)
    AS max_val

T002:
Pre-validation:
MATCH (n:$L1 {$P: $V})<-[:$R]-(m)
    RETURN count(m) AS cnt
Update:
MATCH (a)-[r:$R]->(b:$L1 {$PROP_ID:
    $VALUE}) DELETE r
Post-validation:
MATCH (n:$L1 {$P: $V})<-[:$R]-(m)
    RETURN count(m) AS cnt

T003:
Pre-validation:
MATCH (a:$L1), (b:$L2) WHERE a.$P1 = b
    .$P2 RETURN count(*) AS cnt
Update:
MATCH (n:$L2 {$PROP_ID: $VALUE})
    DETACH DELETE n
Post-validation:
MATCH (a:$L1), (b:$L2) WHERE a.$P1 = b
    .$P2 RETURN count(*) AS cnt

T004:
Pre-validation:
MATCH (a:$L1 {$P1: $V}), (b:$L2) WHERE
     NOT (a)-[:$R]->(b) RETURN count(b
    ) AS cnt
Update:
MATCH (a:$L1 {$PROP_ID1: $AID})-[r:$R
    ]->(b:$L2 {$PROP_ID2: $BID})
    DELETE r
Post-validation:
```

```
MATCH (a:$L1 {$P1: $V}), (b:$L2) WHERE
     NOT (a)-[:$R]->(b) RETURN count(b
    ) AS cnt

T005:
Pre-validation:
MATCH (a:$L1)-[:$R1]->(b:$L2)-[:$R2
    ]->(c:$L3) RETURN count(c) AS cnt
Update:
MATCH (b:$L2 {$PROP_ID: $VALUE})
    DETACH DELETE b
Post-validation:
MATCH (a:$L1)-[:$R1]->(b:$L2)-[:$R2
    ]->(c:$L3) RETURN count(c) AS cnt

T006:
Pre-validation:
MATCH (n:$L) WHERE (n)-[:$R1]->(:$L1)
    OR (n)-[:$R2]->(:$L2) RETURN avg(n
    .$NP)
Update:
MATCH (n:$L {$PROP_ID: $VALUE}) DETACH
     DELETE n
Post-validation:
MATCH (n:$L) WHERE (n)-[:$R1]->(:$L1)
    OR (n)-[:$R2]->(:$L2) RETURN avg(n
    .$NP)

T007:
Pre-validation:
MATCH p = (a:$L1)-[:$R *$MIN_HOPS..
    $MAX_HOPS]-(b:$L2) RETURN avg(
    length(p)) AS avg_len
Update:
MATCH (n {$PROP_ID: $VALUE}) DETACH
    DELETE n
Post-validation:
MATCH p = (a:$L1)-[:$R *$MIN_HOPS..
    $MAX_HOPS]-(b:$L2) RETURN avg(
    length(p)) AS avg_len

T008:
Pre-validation:
MATCH (target:$L {$PROP_ID: $VALUE})
    -[:$R]->(g:$GL) WITH g MATCH (g)
    <-[:$R]-(n:$L) WITH g, n ORDER BY
    n.$P DESC LIMIT 3 RETURN g.$GP, n.
    $P AS top_val
Update:
MATCH (n:$L {$PROP_ID: $VALUE}) DETACH
     DELETE n
Post-validation:
MATCH (g:$GL {$GP: $PREVIOUS_GP_VALUE
    }) MATCH (g)<-[:$R]-(n:$L) WITH g,
     n ORDER BY n.$P DESC LIMIT 3
    RETURN g.$GP, n.$P AS top_val

T009:
Pre-validation:
MATCH (a:$L1), (b:$L2) WHERE a.$P1 = b
    .$P2 WITH a, count(b) AS cnt WHERE
     cnt > $THRESHOLD RETURN a
Update:
MATCH (n:$L2 {$PROP_ID: $VALUE})
    DETACH DELETE n
Post-validation:
MATCH (a:$L1), (b:$L2) WHERE a.$P1 = b
    .$P2 WITH a, count(b) AS cnt WHERE
     cnt > $THRESHOLD RETURN a

T010:
Pre-validation:
MATCH (start:$SL {$SP: $SV}) MATCH p =
     (start) (...){$MIN_HOPS..
    $MAX_HOPS} (end:$EL) RETURN count(
    p) AS cnt
Update:
MATCH (n {$PROP_ID: $VALUE}) DETACH
    DELETE n
Post-validation:
MATCH (start:$SL {$SP: $SV}) MATCH p =
     (start) (...){$MIN_HOPS..
    $MAX_HOPS} (end:$EL) RETURN count(
    p) AS cnt

T011:
Pre-validation:
MATCH (a:$L2) RETURN min(a.$P2) AS
    min_val
Update:
MATCH (n:$L1 {$PROP_ID: $VALUE})
    DETACH DELETE n
Post-validation:
MATCH (a:$L2) RETURN min(a.$P2) AS
    min_val

T012:
Pre-validation:
MATCH (a:$L3)-[:$R2]->(b:$L4) RETURN
    count(*) AS cnt
Update:
MATCH (a)-[r:$R1]->(b:$L1 {$PROP_ID:
    $VALUE}) DELETE r
Post-validation:
MATCH (a:$L3)-[:$R2]->(b:$L4) RETURN
    count(*) AS cnt

T013:
Pre-validation:
MATCH (a:$L2) RETURN avg(a.$P2) AS
    avg_val, count(a) AS cnt
Update:
MATCH (n:$L1 {$PROP_ID: $VALUE})
    DETACH DELETE n
Post-validation:
MATCH (a:$L2) RETURN avg(a.$P2) AS
    avg_val, count(a) AS cnt
```

### E.4.4 Set Templates

```
T001:
Pre-validation:
MATCH (a:$LABEL) RETURN avg(a.$PROP)
    AS avg_val
Update:
MATCH (n:$LABEL {$PROP_ID: $VALUE})
    SET n.$PROP = $VAL
```

```
Post-validation:
MATCH (a:$LABEL) RETURN avg(a.$PROP)
    AS avg_val

T002:
Pre-validation:
MATCH (a:$L1)-[r:$R]-(b:$L2) WHERE r.
    $RP $OP $V RETURN count(*) AS cnt
Update:
MATCH (a {$PROP_ID1: $AID})-[r:$R]->(b
     {$PROP_ID2: $BID}) SET r.$RP =
    $VALUE
Post-validation:
MATCH (a:$L1)-[r:$R]-(b:$L2) WHERE r.
    $RP $OP $V RETURN count(*) AS cnt

T003:
Pre-validation:
MATCH (a:$L1), (b:$L2) WHERE a.$P1 = b
    .$P2 RETURN count(*) AS cnt
Update:
MATCH (n:$L1 {$PROP_ID: $VALUE}) SET n
    .$P1 = $NEW_VAL
Post-validation:
MATCH (a:$L1), (b:$L2) WHERE a.$P1 = b
    .$P2 RETURN count(*) AS cnt

T004:
Pre-validation:
MATCH (a:$L1)-[:$R1]->(b:$L2)-[:$R2
    ]->(c:$L3) RETURN sum(c.$P) AS
    total
Update:
MATCH (n:$L3 {$PROP_ID: $VALUE}) SET n
    .$P = $VAL
Post-validation:
MATCH (a:$L1)-[:$R1]->(b:$L2)-[:$R2
    ]->(c:$L3) RETURN sum(c.$P) AS
    total

T005:
Pre-validation:
MATCH (a:$L1) MATCH (b:$L2 {$P2: a.$P1
    }) RETURN count(b) AS cnt
Update:
MATCH (n:$L1 {$PROP_ID: $VALUE}) SET n
    .$P1 = $NEW_REF
Post-validation:
MATCH (a:$L1) MATCH (b:$L2 {$P2: a.$P1
    }) RETURN count(b) AS cnt

T006:
Pre-validation:
MATCH (n:$L) WHERE n.$P $OP $V OR (n)
    -[:$R]->(:$L2) RETURN count(n) AS
    cnt
Update:
MATCH (n:$L {$PROP_ID: $VALUE}) SET n.
    $P = $NEW_V
Post-validation:
MATCH (n:$L) WHERE n.$P $OP $V OR (n)
    -[:$R]->(:$L2) RETURN count(n) AS
    cnt

T007:
Pre-validation:
MATCH (n:$L {$PROP_ID: $VALUE}) RETURN
     n.$P AS old_value
Update:
MATCH (n:$L {$PROP_ID: $VALUE}) SET n.
    $P = $VAL
Post-validation:
MATCH (n:$L {$PROP_ID: $VALUE}) RETURN
     n.$P AS new_value

T008:
Pre-validation:
MATCH (a:$L1)-[:$R *$MIN_HOPS..
    $MAX_HOPS]-(b:$L2) RETURN count(b)
     AS cnt
Update:
MATCH (n {$PROP_ID: $VALUE}) SET n:$L2
Post-validation:
MATCH (a:$L1)-[:$R *$MIN_HOPS..
    $MAX_HOPS]-(b:$L2) RETURN count(b)
     AS cnt

T009:
Pre-validation:
MATCH (a:$L1), (b:$L2) WHERE a.$P1 = b
    .$P2 WITH a, avg(b.$BP) AS avg_b
    WHERE avg_b > $THRESHOLD RETURN a
Update:
MATCH (n:$L2 {$PROP_ID: $VALUE}) SET n
    .$BP = $VAL
Post-validation:
MATCH (a:$L1), (b:$L2) WHERE a.$P1 = b
    .$P2 WITH a, avg(b.$BP) AS avg_b
    WHERE avg_b > $THRESHOLD RETURN a

T010:
Pre-validation:
MATCH p = (a:$L1)-[:$R *$MIN_HOPS..
    $MAX_HOPS]-(b:$L2) RETURN avg(
    length(p)) AS avg_len
Update:
MATCH (a {$PROP_ID1: $AID}), (b {
    $PROP_ID2: $BID}) CREATE (a)-[:$R
    ]->(b)
Post-validation:
MATCH p = (a:$L1)-[:$R *$MIN_HOPS..
    $MAX_HOPS]-(b:$L2) RETURN avg(
    length(p)) AS avg_len

T011:
Pre-validation:
MATCH (a:$L2) RETURN sum(a.$P2) AS
    total
Update:
MATCH (n:$L1 {$PROP_ID: $VALUE}) SET n
    .$P1 = $VAL
Post-validation:
MATCH (a:$L2) RETURN sum(a.$P2) AS
    total

T012:
Pre-validation:
```

```
MATCH (a:$L3)-[r:$R2]-(b:$L4) WHERE r.
    $RP $OP $V RETURN count(*) AS cnt
Update:
MATCH (a {$PROP_ID1: $AID})-[r:$R1]->(
    b {$PROP_ID2: $BID}) SET r.$RP =
    $VALUE
Post-validation:
MATCH (a:$L3)-[r:$R2]-(b:$L4) WHERE r.
    $RP $OP $V RETURN count(*) AS cnt

T013:
Pre-validation:
MATCH (a:$L3)-[:$R2]->(b:$L4)-[:$R3
    ]->(c:$L5) RETURN count(*) AS cnt
Update:
MATCH (n {$PROP_ID: $VALUE}) SET n:$L2
Post-validation:
MATCH (a:$L3)-[:$R2]->(b:$L4)-[:$R3
    ]->(c:$L5) RETURN count(*) AS cnt
```

### E.4.5 Foreach Templates

```
T001:
Pre-validation:
MATCH (n:$LABEL) RETURN count(n) AS
    cnt
Update:
FOREACH (x IN $LIST | CREATE (:$LABEL
    {val: x}))
Post-validation:
MATCH (n:$LABEL) RETURN count(n) AS
    cnt

T002:
Pre-validation:
MATCH (n:$L1 {$P: $V})-[:$R]->(m)
    RETURN count(m) AS cnt
Update:
MATCH (a:$L1 {$PROP_ID1: $AID}), (
    others:$L2) WHERE others.$PROP_ID2
     IN $IDS FOREACH (o IN [others] |
    CREATE (a)-[:$R]->(o))
Post-validation:
MATCH (n:$L1 {$P: $V})-[:$R]->(m)
    RETURN count(m) AS cnt

T003:
Pre-validation:
MATCH (a:$L1), (b:$L2) WHERE a.$P1 = b
    .$P2 RETURN count(*) AS cnt
Update:
MATCH (n:$L1) WHERE n.$PROP_ID IN $IDS
     FOREACH (i IN [n] | SET i.$P1 =
    $NEW_VAL)
Post-validation:
MATCH (a:$L1), (b:$L2) WHERE a.$P1 = b
    .$P2 RETURN count(*) AS cnt

T004:
Pre-validation:
MATCH (a:$L1)-[:$R1]->(b:$L2)-[:$R2
    ]->(c:$L3) RETURN avg(c.$P) AS
    avg_val
Update:
MATCH (n:$L3) WHERE n.$PROP_ID IN $IDS
     FOREACH (i IN [n] | SET i.$P =
    $VAL)
Post-validation:
MATCH (a:$L1)-[:$R1]->(b:$L2)-[:$R2
    ]->(c:$L3) RETURN avg(c.$P) AS
    avg_val

T005:
Pre-validation:
MATCH (a:$L1) MATCH (b:$L2 {$P2: a.$P1
    }) RETURN count(b) AS cnt
Update:
MATCH (n:$L1) WHERE n.$PROP_ID IN $IDS
     FOREACH (i IN [n] | SET i.$P1 =
    $NEW_REF)
Post-validation:
MATCH (a:$L1) MATCH (b:$L2 {$P2: a.$P1
    }) RETURN count(b) AS cnt

T006:
Pre-validation:
MATCH (n:$L) WHERE (n)-[:$R1]->(:$L1)
    OR (n)-[:$R2]->(:$L2) RETURN sum(n
    .$NP)
Update:
MATCH (n:$L) WHERE n.$PROP_ID IN $IDS
    FOREACH (i IN [n] | CREATE (i)-[:
    $R1]->(:$L1))
Post-validation:
MATCH (n:$L) WHERE (n)-[:$R1]->(:$L1)
    OR (n)-[:$R2]->(:$L2) RETURN sum(n
    .$NP)

T007:
Pre-validation:
MATCH (n:$L) WHERE n.$PROP_ID IN $IDS
    RETURN n.$PROP_ID AS id, n.$P AS
    val ORDER BY id
Update:
MATCH (n:$L) WHERE n.$PROP_ID IN $IDS
    FOREACH (i IN [n] | SET i.$P = i.
    $P + $INC)
Post-validation:
MATCH (n:$L) WHERE n.$PROP_ID IN $IDS
    RETURN n.$PROP_ID AS id, n.$P AS
    val ORDER BY id

T008:
Pre-validation:
MATCH p = (a:$L1)-[:$R *$MIN_HOPS..
    $MAX_HOPS]-(b:$L2) RETURN avg(
    length(p)) AS avg_len
Update:
MATCH (a:$L1), (b:$L2) WHERE a.
    $PROP_ID IN $A_IDS AND b.$PROP_ID
    IN $B_IDS FOREACH (x IN [1] |
    CREATE (a)-[:$R]->(b))
Post-validation:
MATCH p = (a:$L1)-[:$R *$MIN_HOPS..
    $MAX_HOPS]-(b:$L2) RETURN avg(
    length(p)) AS avg_len

T009:
```

```
Pre-validation:
MATCH (a:$L1), (b:$L2) WHERE a.$P1 = b
    .$P2 WITH a, count(b) AS cnt WHERE
     cnt > $THRESHOLD RETURN a
Update:
MATCH (n:$L2) WHERE n.$PROP_ID IN $IDS
     FOREACH (i IN [n] | SET i.$P2 =
    $NEW_REF)
Post-validation:
MATCH (a:$L1), (b:$L2) WHERE a.$P1 = b
    .$P2 WITH a, count(b) AS cnt WHERE
     cnt > $THRESHOLD RETURN a

T010:
Pre-validation:
MATCH (s:$SL {$SP: $SV}) MATCH p = (s)
     ((n1:$L1)<-[:$R1]-(r:$RL)-[:$R2
    ]->(n2:$L2) WHERE r.$NP = $NV){
    $MIN_HOPS..$MAX_HOPS} (e:$EL)
    RETURN count(p) AS cnt
Update:
MATCH (r:$RL) WHERE r.$PROP_ID IN $IDS
     FOREACH (i IN [r] | SET i.$NP =
    $NV)
Post-validation:
MATCH (s:$SL {$SP: $SV}) MATCH p = (s)
     ((n1:$L1)<-[:$R1]-(r:$RL)-[:$R2
    ]->(n2:$L2) WHERE r.$NP = $NV){
    $MIN_HOPS..$MAX_HOPS} (e:$EL)
    RETURN count(p) AS cnt

T011:
Pre-validation:
MATCH (a:$L2) RETURN avg(a.$P2) AS
    avg_val, count(a) AS cnt
Update:
MATCH (n:$L1) WHERE n.$PROP_ID IN $IDS
     FOREACH (i IN [n] | SET i.$P1 =
    $VAL)
Post-validation:
MATCH (a:$L2) RETURN avg(a.$P2) AS
    avg_val, count(a) AS cnt

T012:
Pre-validation:
MATCH (a:$L3)-[:$R2]->(b:$L4) RETURN
    count(*) AS cnt
Update:
MATCH (a:$L1), (b:$L2) WHERE a.
    $PROP_ID IN $A_IDS AND b.$PROP_ID
    IN $B_IDS FOREACH (x IN [1] |
    CREATE (a)-[:$R1]->(b))
Post-validation:
MATCH (a:$L3)-[:$R2]->(b:$L4) RETURN
    count(*) AS cnt

T013:
Pre-validation:
MATCH (a:$L3), (b:$L4) WHERE a.$P3 = b
    .$P4 RETURN count(*) AS cnt
Update:
MATCH (n:$L1) WHERE n.$PROP_ID IN $IDS
     FOREACH (i IN [n] | SET i.$P1 =
    $NEW_VAL)
Post-validation:
MATCH (a:$L3), (b:$L4) WHERE a.$P3 = b
    .$P4 RETURN count(*) AS cnt
```

# F Detailed Evaluation Metrics

In this section, we provide detailed definitions and mathematical formulations for the evaluation metrics used in our experiments. The metrics are categorized based on the output characteristics of the three distinct experimental settings: *No Aggregation (NoAgg)*, *Boolean Queries*, and *Aggregation (Agg)*.

## F.1 Set-based Metrics

In the *NoAgg* setting, the model output and the ground truth are treated as sets of entities. Let $y$ denote the ground-truth set of entities and $\hat{y}$ denote the predicted set of entities for a given query.

**Jaccard Similarity.** To measure the overall overlap between the predicted set and the ground-truth set, we utilize the Jaccard Similarity coefficient. It is defined as the size of the intersection divided by the size of the union of the two sets:

$$J(y, \hat{y}) = \frac{|y \cap \hat{y}|}{|y \cup \hat{y}|}. \tag{7}$$

A value of 1 indicates a perfect match, while 0 indicates no overlap.

**F1 Score.** To jointly assess the precision and recall of the retrieved entities, we employ the F1 score. First, Precision ($P$) and Recall ($R$) are calculated as follows:

$$P = \frac{|y \cap \hat{y}|}{|\hat{y}|}, \quad R = \frac{|y \cap \hat{y}|}{|y|}. \tag{8}$$

The F1 score is the harmonic mean of Precision and Recall:

$$F_1 = 2 \cdot \frac{P \cdot R}{P + R}. \tag{9}$$

## F.2 Classification Metrics

**Accuracy.** We treat this as a binary classification problem. Let $N$ be the total number of candidate pairs evaluated. Let $y_i \in \{0, 1\}$ be the ground truth label for the $i$-th pair (where 1 represents a valid result pair and 0 otherwise), and let $\hat{y}_i \in \{0, 1\}$ be the model's prediction. Accuracy is defined as:

$$\text{Accuracy} = \frac{1}{N} \sum_{i=1}^{N} I(\hat{y}_i = y_i), \tag{10}$$

where $I(\cdot)$ is the indicator function which equals 1 if the condition is true and 0 otherwise.

Table 9: Performance comparison on aggregation queries, where execution on the noisy graph is equivalent to the clean graph.

| Baseline | NGD-Fin | | | NGD-BI | | | NGD-Prime | | |
|---|---|---|---|---|---|---|---|---|---|
| | MdRE | MSLE | sMAPE | MdRE | MSLE | sMAPE | MdRE | MSLE | sMAPE |
| Neo4j-Text2Cypher | 0.000 | 0.055 | 0.021 | 0.000 | 0.000 | 0.001 | 0.965 | 9.495 | 1.155 |
| GPT5.1-Codex | 0.000 | 0.001 | 0.001 | 0.000 | 3.060 | 0.615 | 0.000 | 11.476 | 0.541 |
| DeepseekV3.2 | 0.000 | 0.001 | 0.003 | 0.000 | 5.148 | 0.763 | 0.000 | 8.115 | 0.614 |
| Qwen3-Coder | 0.000 | 0.048 | 0.018 | 0.000 | 2.801 | 0.756 | 0.000 | 7.623 | 0.639 |
| GraphRAG | 0.000 | 0.588 | 0.904 | —— | 0.480 | 1.998 | 0.875 | 2.400 | 1.201 |

Table 10: Performance comparison on aggregation queries, where execution on the noisy graph is inequivalent to the clean graph.

| Baseline | NGD-Fin | | | NGD-BI | | | NGD-Prime | | |
|---|---|---|---|---|---|---|---|---|---|
| | MdRE | MSLE | sMAPE | MdRE | MSLE | sMAPE | MdRE | MSLE | sMAPE |
| Neo4j-Text2Cypher | 0.179 | 27.846 | 0.810 | —— | 16.082 | 16.509 | —— | 46.272 | 1.541 |
| GPT5.1-Codex | 0.131 | 14.940 | 0.771 | 0.992 | 19.820 | 1.233 | 0.237 | 12.341 | 0.766 |
| DeepseekV3.2 | 0.113 | 7.629 | 0.451 | 0.995 | 25.755 | 1.312 | 0.582 | 13.345 | 0.875 |
| Qwen3-Coder | 0.116 | 8.231 | 0.446 | 1.000 | 28.878 | 1.337 | 0.578 | 14.185 | 0.891 |
| GraphRAG | 254.333 | 36.515 | 1.777 | —— | 61.688 | 1.979 | 993.563 | 47.204 | 1.814 |

### F.3 Numerical Error Metrics

In the *Agg* setting, the output is a scalar numerical value (e.g., from `COUNT`, `MAX`, `MIN`). Let $y_i$ be the ground truth numerical value and $\hat{y}_i$ be the predicted value for the $i$-th query, with a total of $N$ queries.

**Median Relative Error (MdRE).** To capture the typical prediction error while remaining robust to outliers, we calculate the median of the absolute relative errors:

$$\text{MdRE} = \text{median}\left(\left\{\frac{|\hat{y}_i - y_i|}{y_i}\right\}_{i=1}^{N}\right). \quad (11)$$

Note that for stability, a small $\epsilon$ is typically added to the denominator if $y_i$ can be zero.

**Mean Squared Logarithmic Error (MSLE).** To penalize multiplicative deviations rather than additive ones (which is crucial for data with large variances), we use MSLE:

$$\text{MSLE} = \frac{1}{N}\sum_{i=1}^{N}\left(\log(1+\hat{y}_i) - \log(1+y_i)\right)^2. \quad (12)$$

**Symmetric Mean Absolute Percentage Error (sMAPE).** We adopt sMAPE to provide a scale-invariant measure of error that is robust to zero values. It is calculated as the average of the absolute difference divided by the sum of absolute values:

$$\text{sMAPE} = \frac{100\%}{N}\sum_{i=1}^{N}\frac{|\hat{y}_i - y_i|}{(|\hat{y}_i| + |y_i|)/2}. \quad (13)$$

This metric ranges from 0% to 200%, providing a symmetric view of overestimation and underestimation.

**Mean Log Relative Error (MLRE).** We employ MLRE to assess the relative deviation between predicted and actual values, which is particularly useful for capturing order-of-magnitude errors. It is calculated as the mean absolute logarithm of the ratio between the prediction and the ground truth, stabilized by a small constant $\epsilon$:

$$\text{MLRE} = \frac{1}{N}\sum_{i=1}^{N}\left|\ln\left(\frac{\hat{y}_i + \epsilon}{y_i + \epsilon}\right)\right|, \quad (14)$$

where $\epsilon$ is a small term (e.g., $10^{-6}$) added to prevent numerical instability. This metric treats the error as a ratio, meaning that a prediction of twice the ground truth yields the same error as a prediction of half the ground truth.

## G More Experiments

We show more results about aggregation queries on structured domains in Table 9 and Table 10.

And we show full metrics about aggregation queries on graphs constructed from structured domains in Tabel 11.

Table 11: Performance comparison of RAG methods on aggregation queries. Exact measures exact-match accuracy, $\pm 1$ measures the proportion of predictions within one unit of the gold answer, and Null denotes invalid or empty outputs; RMSE, MdRE, MSLE, and sMAPE measure numerical errors. Higher is better for Exact and $\pm 1$, while lower is better for Null and all error metrics.

| Baseline | NGD-MCP | | | | | | | NGD-Econ | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | Ex. | $\pm 1$ | Null | RMSE | MdRE | MSLE | sMAPE | Ex. | $\pm 1$ | Null | RMSE | MdRE | MSLE | sMAPE |
| HippoRAG2 | 0.088 | 0.250 | 0.619 | 53.012 | 1.000 | 1.051 | 1.196 | 0.012 | 0.038 | 0.097 | 158.308 | 1.000 | 2.138 | 0.926 |
| GraphRAG | 0.493 | 0.624 | 0.047 | 1.795 | 0.000 | 0.170 | 0.396 | 0.241 | 0.458 | 0.154 | 363623.864 | 0.667 | 0.673 | 0.637 |
| TOG | 0.266 | 0.342 | 0.044 | 16.169 | 0.000 | 0.359 | 0.380 | 0.260 | 0.323 | 0.125 | 18730.315 | 0.000 | 3.809 | 0.371 |

# H Experiment Details

## H.1 General Failure Case Study

We summarize representative failure cases of different baselines. These cases illustrate that existing methods fail for different reasons: Text-to-Cypher methods are limited by exact symbolic execution over noisy graphs and may also misunderstand schema semantics, while RAG-style methods suffer from top-$k$ retrieval truncation and fragmented structural evidence. Together, these examples explain why current approaches struggle with robust query answering under noisy, incomplete, and structurally complex data.

**Neo4j Text2Cypher.** We observe two typical failure modes.

**Case 1: correct query generation but wrong answer due to graph noise.** For the query "Find the count of drugs that do not have an `exposure_molfunc` relationship with Fluocinonide", the generated Cypher query is valid. However, executing it on the noisy graph returns **8807**, while the clean answer is **7957**. This happens because the method relies on traditional symbolic execution and therefore cannot recover from incomplete-edge noise.

**Case 2: incorrect query generation due to schema misunderstanding.** For the query "Find the count of diseases not associated with Cotinine through a `molfunc_molfunc` relationship", the system incorrectly treats Cotinine as a `molfunc` entity and produces an invalid Cypher query, leading to the wrong answer **0**. This shows that Text-to-Cypher baselines may fail both because of graph noise and because of schema or type misunderstanding.

**Advanced LLM-based Text2Cypher.** Stronger LLMs, such as GPT-5.1-Coder, usually avoid obvious query-generation errors, but they still fail when the graph itself is noisy. For example, for the query "Find account owned by person with create time '2020-01-12 20:52:02.831' and return its node ID", GPT-5.1-Coder generates the correct Cypher query. However, it returns only one account instead of all three correct accounts, because some ownership edges are missing in the noisy graph. This indicates that even advanced LLM-based Text-to-Cypher methods remain limited by exact execution over incomplete graphs.

**RAG-style Methods.** We observe two main failure modes.

**Case 1: poor aggregation due to top-$k$ retrieval truncation.** For the query "Find the average number of likes across posts", the correct answer is **23.7056**, but GraphRAG outputs **Zero**. The reason is that aggregation requires broad or even complete graph coverage, while retrieval only provides a top-$k$ subset of context, which is insufficient for exact computation.

**Case 2: failure on multi-hop reasoning because retrieval breaks structural dependencies.** For the query asking for entities that promote active transportation within 1 to 5 steps, the correct answers include *policies* and *government*. However, GraphRAG returns semantically related but incorrect entities such as *Cities*, *Kenya*, and *Many cities in Europe*. This suggests that retrieval provides fragmented local evidence and does not preserve the full path structure required for accurate multi-hop reasoning.

## H.2 Case Study of RAG-based Methods

We summarize representative success and failure cases of RAG-based methods. These cases show that retrieval-based methods can succeed when the answer is contained in a small set of directly retrieved triples, but often fail when the query requires complete structural evidence, multi-hop path preservation, or exact aggregation. For readability, we abbreviate the retrieved contexts and only show representative snippets.

**GraphRAG failure on multi-hop chain queries.**

- **Query:** For the windows performance report gathering task, find the entity representing `complete_report` and return its ID along with the IDs of all entities reachable from it within 1 to 5 `followed_by` hops.
- **Ground truth:** `additional metrics`, `system uptime`, `network config`, and `system specs`.
- **Retrieved context:** GraphRAG retrieves several report-related triples, including `windows_performance_report_gathering_additional_metrics` `followed_by` `windows_performance_report_gathering_complete_report`, together with other partially related task and report triples. However, the retrieved context does not preserve the complete `followed_by` chain.
- **Model output:** The model identifies`windows_performance_report_gathering_complete_report`, but incorrectly concludes that there are no outgoing `followed_by` edges and returns only this entity.
- **Analysis:** This is a typical multi-hop failure. Retrieval provides fragmented local evidence, while the query requires complete path-level structure.

**GraphRAG success on single-hop queries.**

- **Query:** For the contingency plan development task, retrieve the outputs produced by the resource allocation strategy implementation.
- **Ground truth:** `tiered resource pools`.
- **Retrieved context:** The retrieved context contains the key triple `contingency_plan_development_resource_allocation_strategy_implementation` `produces_output` `contingency_plan_development_tiered_resource_pools`,along with other resource-allocation-related triples.
- **Model output:** The model correctly returns `contingency_plan_development_tiered_resource_pools`.
- **Analysis:** The required evidence appears as a directly retrieved one-hop triple. This suggests that GraphRAG can handle simple local triple matching, but may struggle with multi-hop traversal or global graph structure.

**ToG-RAG success with direct relational evidence.**

- **Query:** How many entities, and what are their IDs, are such that IRS Clearing Members shall only reduce a call for performance bond?
- **Ground truth:** Count: `1`; entity: `through the receipt of performance bond deposits`.
- **Retrieved context:** The retrieved context contains a relevant document passage about performance bond deposits and the key triple `(IRS Clearing Members, shall only reduce a call for performance bond, through the receipt of performance bond deposits)`.
- **Model output:** The model identifies the retrieved relation and correctly answers `1`.
- **Analysis:** This case succeeds because the exact relational evidence is retrieved. It does not require exhaustive traversal or aggregation over a large result set.

**ToG-RAG failure on aggregation under incomplete retrieval.**

- **Query:** How many tools does the Hawaii last minute getaway task use?
- **Ground truth:** `12`.
- **Retrieved context:** ToG-RAG retrieves several relevant triples, such as `hawaii_last_minute_getaway_task` `uses_tool` `hawaii_last_minute_getaway_check_amenities` and `hawaii_last_minute_getaway_task` `uses_tool` `hawaii_last_minute_getaway_search_hotels`, together with a long dialogue context about hotel amenities and travel planning.
- **Model output:** The model counts only the retrieved tools and returns `3`.

- **Analysis:** This is a typical aggregation failure caused by retrieval truncation. Exact counting requires the complete set of `uses_tool` edges, but top-$k$ retrieval only exposes partial evidence.

**HippoRAG2 lucky hit without relevant retrieval.**

- **Query:** For the visual cube representation task, how many outcomes does the draw front face step achieve?
- **Ground truth:** `1`.
- **Retrieved context:** The retrieved contexts are unrelated to the target task, including CrewAI documentation and shadcn/ui code snippets. No relevant knowledge graph evidence about the visual cube representation task is retrieved.
- **Model output:** The model answers that the draw front face step produces exactly one outcome, namely a square serving as the front face of the cube, and returns `1`.
- **Analysis:** Although the answer matches the ground truth, it is not grounded in the retrieved evidence. The model likely relies on commonsense reasoning about drawing a cube, so this should be interpreted as a lucky hit rather than successful graph reasoning.

**HippoRAG2 failure on exact two-hop structural queries.**

- **Query:** Find the entity with ID `mineral_information_server` and the entities connected to it by exactly two `uses_server` hops, and return their IDs.
- **Ground truth:** `geological information server` and `time scale server`.
- **Retrieved context:** HippoRAG2 retrieves unrelated conversational logs and server troubleshooting content, including WildFly connection errors and other MCP-related dialogue. The retrieved context does not mention `mineral_information_server` or the required `uses_server` paths.
- **Model output:** The model states that no relevant entity or relation can be found and returns `None`.
- **Analysis:** This case highlights the weakness of text-first retrieval pipelines on precise graph queries that require exact entity matching and path preservation.

This is an appendix.