CypherLens: An Interactive Demo for Evaluating and Diagnosing NL-to-Cypher Systems

Zahra Mohammadi
Temple University
Philadelphia, Pennsylvania, USA
zahra.mohammadi@temple.edu

Ali Ansari
Temple University
Philadelphia, Pennsylvania, USA
ali.ansari@temple.edu

Longin Latecki
Temple University
Philadelphia, Pennsylvania, USA
latecki@temple.edu

Eduard Dragut
Temple University
Philadelphia, Pennsylvania, USA
edragut@temple.edu

ABSTRACT

Natural-language-to-Cypher (NL-to-Cypher) systems are typically evaluated using execution accuracy (EA), which treats queries as correct if they produce the same result as a reference. However, EA is brittle, penalizing semantically equivalent queries, rewarding coincidental matches, and providing no insight into failure modes.

We present **CypherLens**, an interactive platform for evaluating and diagnosing NL-to-Cypher methods that extends execution-based evaluation with detailed analysis and actionable insights. CypherLens automatically generates Cypher queries, executes both generated and reference queries, normalizes outputs, aligns the returned columns and aliases, and computes multiple evaluation metrics, including Exact Match and Soft Match, without requiring user intervention. The platform incorporates a query classifier to characterize query types and a diagnoser that assigns graph-native semantic and syntactic error labels based on a Cypher-specific taxonomy. CypherLens supports analysis at both query and dataset levels. At the query level, it provides detailed diagnostic signals that enable users to inspect individual failures and identify actionable fixes. At the dataset level, it aggregates metrics and error labels to reveal performance trends, such as query-class success rates and error distributions, enabling systematic understanding of model behavior. These insights are presented through an integrated visualization interface that supports interactive exploration and comparison. CypherLens enables users to move beyond binary evaluation toward a more comprehensive and interpretable analysis of NL-to-Cypher systems, supporting debugging, failure analysis, and model comparison.

PVLDB Reference Format:
Zahra Mohammadi, Ali Ansari, Longin Latecki, and Eduard Dragut. CypherLens: An Interactive Demo for Evaluating and Diagnosing NL-to-Cypher Systems. PVLDB, 19(12): 4558 - 4561, 2026.
doi:10.14778/3827998.3828065

PVLDB Artifact Availability:
The source code, data, and/or other artifacts have been made available at https://github.com/YasmeanMohammadi/CypherLens.

This work is licensed under the Creative Commons BY-NC-ND 4.0 International License. Visit https://creativecommons.org/licenses/by-nc-nd/4.0/ to view a copy of this license. For any use beyond those covered by this license, obtain permission by emailing info@vldb.org. Copyright is held by the owner/author(s). Publication rights licensed to the VLDB Endowment.
Proceedings of the VLDB Endowment, Vol. 19, No. 12 ISSN 2150-8097.
doi:10.14778/3827998.3828065

1 INTRODUCTION

Graph databases are increasingly used to represent and query knowledge graphs across scientific and domain-specific applications [5], where structured representations of entities, relations, datasets, methods, and tasks support semantic search, discovery, and downstream reasoning [1, 9–12]. As graph databases become more widely adopted, NL-to-Cypher (T2C) systems are becoming important for translating natural language questions into Cypher queries, thereby lowering the barrier to querying graph databases. Recent advances in large language models (LLMs) have significantly improved generation quality, with modern T2C systems achieving strong execution accuracy on standard benchmarks [8]. Despite this progress, evaluating and understanding T2C system behavior remains challenging. The dominant evaluation metric, execution accuracy (EA), determines correctness by comparing query results against a reference query. However, EA suffers from fundamental limitations that can obscure true model behavior and hinder systematic improvement. We highlight several key challenges, which arise at different stages of evaluation: result equality, Cypher result representation, diagnostic explanation, partial scoring, and semantic correctness under dataset-specific artifacts.

Challenge 1: Metric Brittleness. EA relies on strict result equality between generated and gold queries, leading to false negatives when semantically equivalent queries differ in superficial ways such as variable names, projection aliases, column ordering, or result ordering. Thus, correct queries may be penalized despite matching user intent, causing EA to underestimate true model performance [2].

Example: For the question *"What is the earliest founding year of teams with Spurs-drafted players?"* the gold query returns the result under the implicit key *min(n.inception_year)*, while the generated query returns the same value (1937) under the alias *earliest_founding_year*. Despite identical semantics and output, EA assigns a score of 0 due to mismatched column names.

Challenge 2: Projection and Result Normalization. Unlike SQL, Cypher does not produce a fixed tabular schema [3]. Results may contain nested objects, maps, or node containers, and are returned as heterogeneous JSON structures. Aligning projections

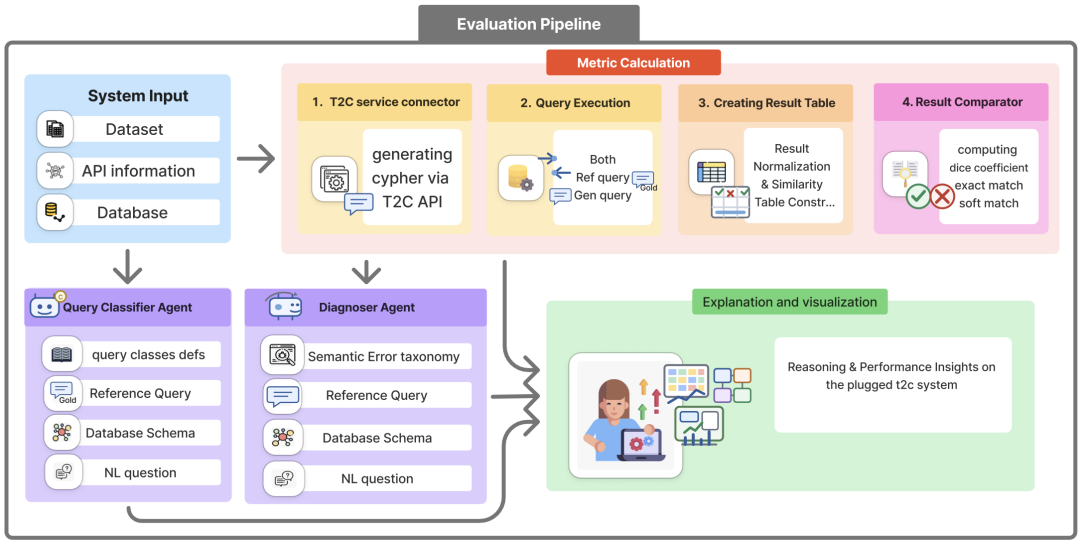

Figure 1: Overview of the evaluation system for diagnosing Text2Cypher models.

across queries requires normalization, alias resolution, and semantic matching, capabilities not supported by standard EA, which also fails to handle nulls, empty results, execution errors, or timeouts.

Example: A query returning `n.name, n.gender` and another returning `p.name AS founder, p.gender AS gender` produce identical data but differ in schema representation. Without normalization, they appear mismatched.

Challenge 3: Error Opacity and Query Blindness. Binary correctness signals indicate whether a query succeeds but provide little insight into why it fails. Existing evaluation approaches offer limited support for interactive, Cypher-specific diagnosis of semantic and syntactic failure modes, such as projection errors, relationship misuse, and missing conditions [7]. Recent systems such as SQLENS [4] and NL2SQL360 [6] study clause-level error detection and evaluation for SQL, but they are tied to SQL's tabular structure and do not directly extend to graph-native query languages such as Cypher. As a result, fine-grained diagnostic tools for NL-to-Cypher remain underexplored. In addition, aggregate metrics obscure performance differences across query types, making it difficult to determine whether failures arise from simple retrieval, aggregation, or multi-hop reasoning.

Example: A query failure may stem from an incorrect string literal, missing `OPTIONAL MATCH`, or an incorrect traversal direction, yet EA provides no signal to distinguish among these causes.

Challenge 4: Binary Scoring Limitation. EA treats near-miss queries and fundamentally incorrect ones identically. Queries that capture the correct core result but include extra projections or minor errors receive the same score as nonsensical outputs, limiting the usefulness of evaluation signals.

Example: A query asking *"Who are the CEOs, past or present, of companies headquartered in the Czech Republic?"* should return only CEO names. The generated query additionally returns company name, start year, and end year for each CEO. Although both queries identify the same executives, the extra columns cause EA to assign a score of 0—treating a near-miss with correct core content the same as a completely incorrect answer.

Challenge 5: Coincidental Correctness. Structurally incorrect queries may coincidentally return correct results due to dataset-specific artifacts. Since EA evaluates only result equality, it cannot detect such cases, leading to inflated performance estimates and masking underlying reasoning errors.

Example: A query asking for *players who received an award while playing for the Lakers* must constrain the award to the player's Lakers tenure. If a generated query instead returns all award-winning players who ever played for the Lakers, it may still match the gold result on the current dataset. EA would score it as correct, masking a traversal or condition error that could fail under data changes. In our evaluation on the Neo4j Twitter dataset, 7.75% of queries were flagged as correct-by-coincidence candidates.

To address these challenges, we present CypherLens, an interactive platform for evaluating and diagnosing NL-to-Cypher systems. CypherLens executes and normalizes query results to handle metric brittleness and result heterogeneity (C1, C2), flags candidate correct-by-coincidence cases (C5), and reports exact and soft-match metrics for partial correctness (C4). It also classifies queries by structural type and assigns Cypher-specific semantic and syntactic error labels (C3). An interactive interface supports both query- and dataset-level analysis for failure diagnosis, model comparison, and systematic inspection of model behavior.

This demo focuses on the design and interactive workflow of CypherLens; full benchmarking of diagnostic accuracy, human-annotated error labels, and alternative partial-credit metrics is left to ongoing work.

2 SYSTEM OVERVIEW

Figure 1 shows the architecture of CypherLens, a modular evaluation platform for NL-to-Cypher systems. The system takes three inputs: a dataset of natural-language questions with gold Cypher queries, a database connection for query execution, and API access to the target T2C system. It does not require access to the model architecture or implementation. Given these inputs, CypherLens

generates candidate queries, executes and compares results, classifies query types, and diagnoses errors, producing both quantitative scores and qualitative diagnostic signals for each instance.

2.1 Execution and Schema-Aware Evaluation

The core of CypherLens is a schema-aware evaluation pipeline, illustrated in steps 1–4 of Figure 1. For each instance, the system submits the natural language question to the T2C API to generate a candidate Cypher query (step 1), then executes both generated and gold queries against the target Neo4j database (step 2). Because Cypher returns heterogeneous nested JSON rather than flat relational tables, direct comparison is brittle (C1, C2).

CypherLens first retrieves schema metadata from the Neo4j database, including node labels, relationship types, property keys, relationship directions, and constraints or indexes. It then builds a Similarity Table through staged matching: exact column matching, alias normalization, type-aware matching, and schema-aware matching. Embedding-based alignment is used only as a fallback when surface forms differ but expressions are semantically related, such as matching `sum(x)/size(collect(x)) AS average_x` with `avg(x) AS avg`. These alignments support inspection and partial scoring, not definitive proof of semantic equivalence.

The Result Comparator (step 4) uses the Similarity Table to compute *Exact Match* (EM), *Soft Match*, and *Dice coefficient*, capturing exact agreement, partial cell-level correctness, and unordered value overlap. These metrics reveal gradations of correctness hidden by binary execution accuracy (C4).

2.2 Query Type Classifier

Aggregate metrics treat all queries uniformly, obscuring how a T2C system performs across different levels of semantic complexity (C3). A model may achieve strong overall accuracy while failing on multi-hop or path queries, while high accuracy on simple retrieval tasks can inflate scores and mask such weaknesses. To address this, CypherLens incorporates a Query Classifier Agent that assigns each instance to a canonical query class (e.g., simple retrieval, aggregation, multi-hop traversal, path query) based on the natural language question, schema, and gold query. When dataset labels are available, they are reused; otherwise, the classifier is invoked automatically, ensuring compatibility with both annotated and unannotated benchmarks. This enables metric and error breakdowns by query type, allowing users to identify systematic weaknesses and potential targets for model improvement.

2.3 Diagnoser Agent

When a generated query fails, execution accuracy gives no explanation for the failure (C3). CypherLens therefore invokes a Diagnoser Agent whenever EM,=,0. The agent reasons over the question, graph schema, gold query, generated query, and sampled execution results, then proposes where and why the generated query diverges from the intended semantics.

Diagnoses use a hierarchical Cypher-specific error taxonomy from our broader study of NL-to-Cypher semantic errors. The taxonomy covers projection and value semantics, pattern topology, binding and scope, and clause-level errors, with subtypes such as *Wrong Relationship Type*, *Direction Mismatch*, *Wrong Hop Count*, *Implicit Condition Missing*, and *Variable Scope Error*. Each diagnosis includes a confidence score and an evidence-grounded rationale.

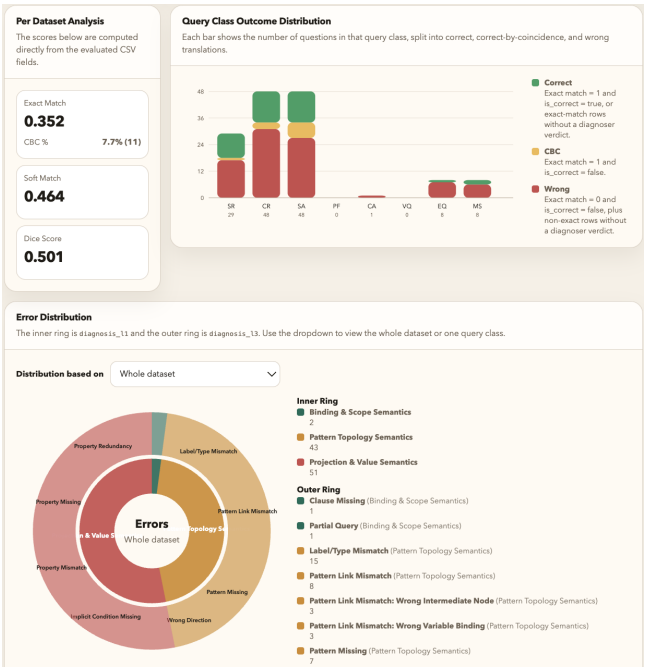

Figure 2: Results analysis interface showing overall metrics, per-query-class performance, and hierarchical error distributions across semantic failure categories.

The Diagnoser Agent can also inspect cases with EM,=,1 for coincidental correctness (C5). If a generated query matches the result but diverges structurally from the intended semantics, CypherLens flags it as a *correct-by-coincidence* candidate. Both the classifier and Diagnoser Agents use a provider-agnostic routing layer, allowing flexible integration with different LLM backends.

2.4 Interactive Visualization Interface

All evaluation signals are presented through an interactive visualization interface that supports exploration at two levels. At the dataset level, the interface provides aggregated metrics, per-query-class performance breakdowns, and error distributions at both the dataset and query-class levels, enabling users to identify global trends and pinpoint class-specific failure patterns. At the query level, users can examine individual instances in detail, including metric scores, query type, execution outputs, and diagnostic reports, facilitating targeted debugging and fine-grained failure analysis. Figure 2 illustrates the results analysis page, highlighting both overall performance and error breakdown.

3 DEMONSTRATION SCENARIOS

We demonstrate CypherLens through three scenarios that cover the full evaluation workflow: system setup and dataset-level analysis, per-query debugging, and coincidental-correctness inspection. All scenarios use a real Neo4j graph database built from the Twitter knowledge graph subset of the Text2Cypher benchmark [8]. The target T2C system follows the prompting-based method described in

that benchmark and is exposed to CypherLens through a webhook interface. In our demo configuration, the optional Query Classifier and Diagnoser Agents are instantiated with Gemini-2.5-Pro through the provider-agnostic routing layer.

Scenario 1: System Setup and Dataset-Level Evaluation. The user begins by configuring CypherLens through a lightweight setup interface (❶), providing a dataset of natural language questions with gold Cypher queries, database connection details, and API credentials for the target T2C system. Optional components, including the Diagnoser Agent, Query Classifier Agent, and correct-by-coincidence (CBC) inspection, can be enabled depending on desired evaluation depth and computational cost (❷), with users selecting their preferred LLM backend. With a single click, the system executes the full evaluation pipeline, including query generation, execution, metric computation, and analysis. Execution progress and intermediate results are streamed in real time, with logs available for post hoc inspection.

Upon completion, the interface presents a dataset-level results dashboard (❸). Users first examine aggregate scores across Exact Match and Soft Match, obtaining a graded view of performance beyond binary accuracy. As shown in Figure 2, an Exact Match of 0.352 alongside a Soft Match of 0.464 reveals that many failed queries are near-misses rather than fundamentally incorrect outputs. The dashboard also provides query-class breakdowns and error distributions, revealing whether failures concentrate in specific query types (e.g., multi-hop traversal or aggregation) and which error categories dominate. For example, a user may observe that while simple retrieval queries are largely solved, pattern topology errors dominate failures on multi-hop queries—an insight not visible under standard execution accuracy.

Scenario 2: Per-Query Debugging and Inspection. The user selects an individual query from the dataset view to open the per-query inspection panel (❶). The interface presents the natural language question, gold Cypher query, and generated query side by side, along with their execution results and the constructed similarity table (❷). Metric scores, like Exact Match, Soft Match, and Dice coefficient, are displayed at the top, allowing the user to assess both correctness and degree of similarity.

The Diagnoser Agent output (❸) provides a structured error label, confidence score, and textual rationale grounded in the query, schema, and execution evidence. For example, the system may flag that a generated query uses a one-hop relationship where a two-hop traversal is required—a *Wrong Hop Count* error that may appear correct under the current dataset but would fail under different graph instances. Soft Match of 0.5 correctly reflects that the core result is partially correct, a distinction EA obscures by assigning a score of 0. The Query Classifier label is also shown, placing the query within its semantic category. Together, these signals help users inspect failure modes, distinguish near-miss errors from incorrect outputs, and identify likely directions for prompt refinement or model improvement.

Scenario 3: Candidate Correct-by-Coincidence Inspection. This scenario highlights a blind spot in execution-based evaluation: queries that return the correct result but are semantically incorrect (C5). The user filters the dataset view to cases where Exact Match equals one and CBC is flagged (❶), isolating queries that standard evaluation would mark as correct.

The user then opens a flagged instance in the per-query panel (❷). In this example, the generated query skips an intermediate project node and links employees directly to cities, while the gold query follows the required two-hop path. Because the dataset happens to make both traversals return the same result, the query receives EM,=,1. The Diagnoser Agent identifies the structural mismatch (❸) and flags it as a *correct-by-coincidence* candidate with a *Pattern Topology* error. As shown in Figure 2, approximately 7% of EM,=,1 queries are flagged as CBC candidates, illustrating how CypherLens reveals semantic errors hidden by execution accuracy.

4 CONCLUSION

We presented CypherLens, an interactive platform for evaluating and diagnosing NL-to-Cypher systems beyond execution accuracy. CypherLens addresses key limitations of execution-based evaluation, including metric brittleness, limited partial-correctness signals, lack of diagnostic insight, and correct-by-coincidence cases. By combining schema-aware result normalization, multi-metric scoring, query classification, and Cypher-specific diagnostic explanations, the platform supports both dataset-level and per-query analysis. Users can compare systems, identify systematic weaknesses, inspect likely failure modes, and better understand when execution-based correctness may mask semantic errors. Future work will connect these diagnostic signals to automatic refinement loops and broader benchmark-level validation.

Acknowledgements This work was supported in part by the U.S. NSF awards III-2107213, and ITE-2333789.

REFERENCES

[1] Mustapha Adamu, Qi Zhang, Huitong Pan, Longin Jan Latecki, and Eduard C. Dragut. 2025. Querying Climate Knowledge: Semantic Retrieval for Scientific Discovery. In *Workshop on Information Retrieval for Climate Impact*.

[2] Vashu Chauhan, Shobhit Raj, Shashank Mujumdar, Avirup Saha, and Anannay Jain. 2025. Mind the Query: A Benchmark Dataset towards Text2Cypher Task. In *EMNLP*. 1890–1905.

[3] Nadime Francis et al. 2018. Cypher: An evolving query language for property graphs. In *CIKM*. 1433–1445.

[4] Yue Gong, Chuan Lei, Xiao Qin, Kapil Vaidya, Balakrishnan Narayanaswamy, and Tim Kraska. 2025. SQLENS: An end-to-end framework for error detection and correction in text-to-SQL. *arXiv preprint arXiv:2506.04494* (2025).

[5] A. Kruger, Ramon Lawrence, and E.C. Dragut. 2006. Building a terabyte NEXRAD radar database for hydrometeorology research. *Computers Geosciences* 32, 2 (2006), 247–258.

[6] Boyan Li, Yuyu Luo, Chengliang Chai, Guoliang Li, and Nan Tang. 2024. The dawn of natural language to sql: Are we fully ready? *arXiv preprint arXiv:2406.01265* (2024).

[7] Xinyu Liu, Shuyu Shen, Boyan Li, Nan Tang, and Yuyu Luo. 2025. Nl2sql-bugs: A benchmark for detecting semantic errors in nl2sql translation. In *SIGKDD*. 5662–5673.

[8] Makbule Gulcin Ozsoy, Leila Messallem, Jon Besga, and Gianandrea Minneci. 2025. Text2cypher: Bridging natural language and graph databases. In *The GenAIK Workshop*. 100–108.

[9] Huitong Pan, Mustapha Adamu, Qi Zhang, Eduard Dragut, and Longin Jan Latecki. 2025. ClimateIE: A Dataset for Climate Science Information Extraction. In *ClimateNLP 2025*. 76–98.

[10] Huitong Pan, Qi Zhang, Mustapha Adamu, Eduard Dragut, and Longin Jan Latecki. 2025. Taxonomy-Driven Knowledge Graph Construction for Domain-Specific Scientific Applications. In *Findings of ACL*. 4295–4320.

[11] Huitong Pan, Qi Zhang, Cornelia Caragea, Eduard Dragut, and Longin Jan Latecki. 2024. SciDMT: A Large-Scale Corpus for Detecting Scientific Mentions. In *LREC-COLING 2024*. 14407–14417.

[12] Qi Zhang, Zhijia Chen, Huitong Pan, Cornelia Caragea, Longin Jan Latecki, and Eduard Dragut. 2024. SciER: An Entity and Relation Extraction Dataset for Datasets, Methods, and Tasks in Scientific Documents. In *EMNLP*.