
Grounding Large Language Models in Reaction Knowledge Graphs for Synthesis Retrieval

Olga Bunkova¹ Lorenzo Di Fruscia¹ Sophia Rupprecht² Artur M. Schweidtmann²

Marcel J.T. Reinders¹ Jana M. Weber^{1,*}

¹Department of Intelligent Systems, Delft University of Technology, The Netherlands

²Department of Chemical Engineering, Delft University of Technology, The Netherlands

{l.difruscia, s.rupprecht, a.schweidtmann, m.j.t.reinders, j.m.weber}@tudelft.nl

*Corresponding Author

Abstract

Large Language Models (LLMs) can aid synthesis planning in chemistry, but standard prompting methods often yield hallucinated or outdated suggestions. We study LLM interactions with a reaction knowledge graph by casting reaction path retrieval as a Text2Cypher (natural language to graph query) generation problem, and define single- and multi-step retrieval tasks. We compare zero-shot prompting to one-shot variants using static, random, and embedding-based exemplar selection, and assess a checklist-driven validator/corrector loop. To evaluate our framework, we consider query validity and retrieval accuracy. We find that one-shot prompting with aligned exemplars consistently performs best. Our checklist-style self-correction loop mainly improves executability in zero-shot settings and offers limited additional retrieval gains once a good exemplar is present. We provide a reproducible Text2Cypher evaluation setup to facilitate further work on KG-grounded LLMs for synthesis planning. Code is available at <https://github.com/Intelligent-molecular-systems/KG-LLM-Synthesis-Retrieval>.

1 Introduction

Large Language Models (LLMs) have transformed Natural Language Processing (NLP) by enabling tasks such as question answering and text summarization through large-scale pretraining on text corpora [17]. A key capability of LLMs is In-Context Learning (ICL), where models perform new tasks solely conditioned on instructions or demonstrations with no parameter updates [5]. Prompt engineering, the systematic design of inputs to improve outputs, has become essential to harness ICL effectively. Beyond language, LLMs are increasingly applied in various specialized fields, one of them being cheminformatics. The reason for this is their ability to process and interpret molecular data in string format [37].

Chemical synthesis or pathway planning seeks to design a sequence of reactions that produce a target molecule from available precursors or vice versa [30, 8]. With given data, the task requires the identification of optimal sequences of reactions, both forward towards products and backwards towards precursors [33, 32, 34]. In the context of synthesis in unknown chemical space, the task may involve forward prediction, where products are generated from reactants, or retrosynthetic predictions, where candidate precursors for a desired molecule are predicted [27, 13, 25][4]. In both cases, extending this to multi-step planning requires chaining single-step retrievals/predictions into coherent routes, typically through search algorithms [8, 15] [28].

LLMs offer promising reasoning abilities for reaction planning. Among others, they could function as flexible conversation agents to help assemble all possible synthesis routes, suggest alternative/optimized routes, or help evaluate suggested pathways. Yet they remain prone to hallucinations and

outdated knowledge in fast-moving chemical domains [31], [17]. Retrieval Augmented Generation (RAG) [14] partially mitigates these issues, but generally relies on isolated retrieval and not a single, connected view of molecules and reactions across sources. Knowledge Graphs (KGs) provide a more structured alternative by encoding molecules and reactions as entities and relations, preserving both short (direct precursors/products) and long-range links (multi-step synthesis routes) [24], [2]. This representation enables path-constrained retrieval and multi-hop reasoning [11, 10], offering a foundation for grounding LLM predictions in verified reaction data. A practical way to enable this interaction is to pose questions as declarative graph queries in *Cypher* language [6].

While Text2Cypher has been explored for general KGs [23, 9, 3]) there is, to our knowledge, no prior chemistry-specific, execution-grounded evaluation of Text2Cypher tasks (i.e., generating Cypher queries from natural language instruction) on reaction KGs. In this study we assess whether LLMs can generate executable Cypher queries that yield correct single- and multi-step retrosynthesis retrieval over a bipartite KG. Our findings translate into operating guidance: We contribute an evaluation protocol, metrics, and design recommendations for reliable Text2Cypher retrieval over reaction KGs. Furthermore, we analyze how demonstration choice and a lightweight correction loop affect the executed results across prompting strategies and tasks.

2 Methods

Reaction Knowledge Graph: Data and construction We build our dataset using USPTO reactions in SMILES format [36, 35], removing duplicates, canonicalizing SMILES strings, and filtering out rare entries. We retain reactions with at most four reactants, four products, four agents, and four solvents, which cover ~95% of the entire dataset.

Individual chemical reactions can naturally be modeled using different graph-based representations. We rely on a Bipartite Graph (BG) for the KG construction. A BG contains two types of nodes connected only across types, as shown in Figure 1A. Each individual reaction is represented as a bipartite graph with two distinct node types: (**:Reaction**), identified by a unique id, and (**:Molecule**), uniquely identified by a canonicalized SMILES name. This representation preserves the multi-component context of a reaction and the required $\text{Mol} \rightarrow \text{Reac} \rightarrow \text{Mol}$ directionality for path reasoning [12, 18, 7]. The full reaction schema is shown in Figure 1B. For efficiency of this study, we randomly sample 50k reactions and load them into Neo4j [19].

Tasks: Single- and Multi-step retrieval We evaluate our framework on two distinct types of tasks. Single-step tasks target one-hop reaction context (reactants, products, agents/solvents). They focus on different aspects of retrosynthesis and aim to evaluate the ability of the LLM to generate valid Cypher queries of varying syntactic complexity in a chemically constrained scenario. Multi-step tasks target paths of length $\leq L$ linking precursors to a product ($L \in \{2, 3, 4\}$) under the bipartite directionality, representing a broader chemical scenario. The objective for single-step synthesis always requires retrieving the full reaction context (reactants, products, solvents etc.), whereas for multi-step synthesis, it requires retrieving ordered sequences of reaction nodes.

We randomly sample molecules from the KG, and instantiate queries by pairing natural language question templates with a reference Cypher template (sequential MATCH/OPTIONAL MATCH with RETURN/COLLECT) and the corresponding gold, i.e. correct, answers. The single-step setting comprises 1200 queries over six tasks (200 per type). The multi-step setting comprises 1200 queries over four tasks (300 per type), ensuring reachable paths of the specified length. We report our query categories in Tables 1C and 1D.

Text2Cypher prompting and exemplar selection We investigate how our prompting choices affect retrieval over the KG. We design five prompt versions per setting (single- P1- P5/multi-step P6- P10), increasingly adding instructions/context from Prompt one to five. Additionally, we compare zero-shot (ZS) prompting to various one-shot prompting strategies: Static (*IS*, one exemplar is selected and reused for all tasks), dynamic random (*IS-D-R*, for each query one exemplar is randomly selected from a predefined example bank), and dynamic semantic (*IS-D-S*, an exemplar is retrieved from a predefined example bank stored in a vector database, based on vector similarity) schemes. We illustrate the pipeline schema in Figure 2A. More details on the prompting strategies are reported in the Appendix section 5.1.

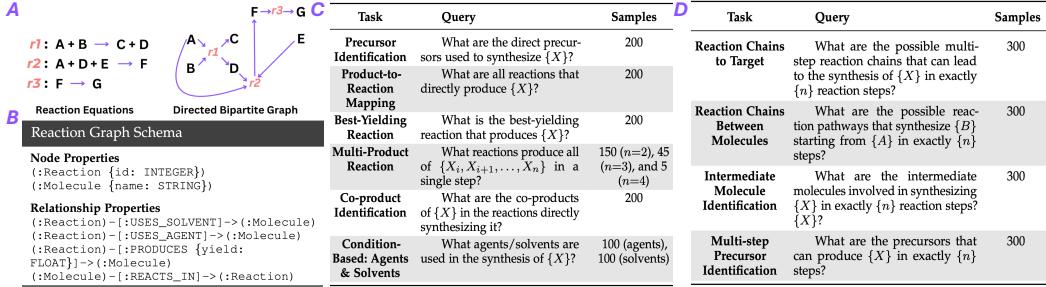


Figure 1: (A) Chemical reaction equation with reactants and products (left), and a possible directed bipartite graph representation, where nodes correspond to reactions and molecules (right). (B) Schema used for the reaction knowledge graph. (C, D) Retrosynthesis task types for single- and multi-step reaction retrieval, along with natural language queries and number of task samples. For multi-step reaction retrieval $n \in \{2, 3, 4\}$.

Checklist validator/corrector We implement a Chain-of-Verification (CoVe) style loop to address common generation errors. After query generation from natural language, a checklist, consisting of the most frequent error types from prior error analysis, validates the executability of the query in Neo4j. On failure, the LLM receives the error message and proposes a corrected query; we allow up to three attempts before termination. We illustrate the CoVe pipeline in Figure 2B.

Framework evaluation and implementation We evaluate our framework based on two concepts: Query evaluation and retrieval evaluation. For query evaluation, we compute text-to-text similarity with the metrics BLEU, METEOR, and ROUGE-L [22] to compare the generated and reference Cypher queries. These metrics vary in their strategy for comparing text sequences, and are further defined in the Appendix section 5.5.

For retrieval evaluation, retrieved results are collected as lists of dictionaries and compared to gold answers with micro-averaged precision, recall, and F1 scores across relevant keys (reactants, agents, solvents etc.). We show our scheme of the gold answer format for both in Figure 2C. For multi-step routes, we report exact-path precision, recall, and F1 scores (treating each path as an ordered set of reaction node ids) and Partial Path Recall (PPR), which gives partial credit when a predicted path matches a continuous terminal fragment of a gold path.

Experiments use GPT-4.1-mini-2025-04-14 [20] via the OpenAI API with $T=0$ to ensure deterministic decoding. Additional implementation details are provided in Appendix Section 5.4.

3 Results

Our framework reliably retrieves single- and multi-step reaction information from the KG given natural language questions across most tasks. In the following, we examine which evaluation criteria are most informative and which design choices most affect retrieval quality.

Text-to-text similarity is a poor proxy for retrieval accuracy We observe that text-to-text similarity between queries measured by BLEU, METEOR, and ROUGE-L does not represent a good indicator of retrieval success. Two mechanisms explain the divergence. First, multiple Cypher formulations can be semantically equivalent and return the same answers despite different structures. Conversely, structurally similar queries with small syntactic shifts (e.g., losing edge directionality) keep the query overlap high while breaking or altering the retrieval. This mismatch can be observed in Figure 3, where retrieval F1 score can vary a lot (e.g. one-shot random vs one-shot semantic, Prompt one to three), with comparable text-to-text similarities for generated Cypher queries. Extended results for single-step tasks are illustrated in Appendix Figures A5, A6 and A4.

Largest performance gains from zero- to one-shot in multi-step tasks Moving from zero- to one-shot prompting removes most common retrieval errors. In multi-step task queries, the most occurring failures are *endpoint anchoring* (treating the target molecule as the start of the reaction

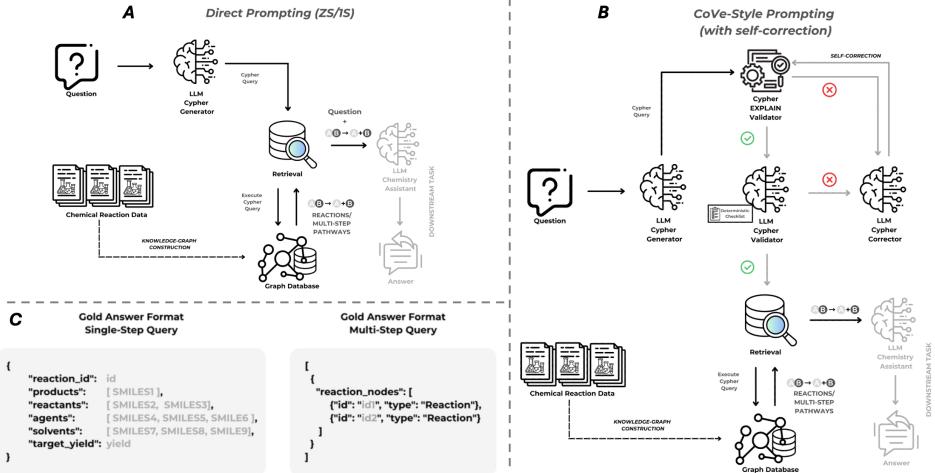


Figure 2: (A) Based on the user question, with (one-shot) or without (zero-shot) an exemplar, an LLM generates a Cypher query with the goal of retrieving relevant reactions/pathways. The query is then executed on the BG in Neo4j. (B) CoVe-style prompting. First, an LLM generates a candidate query. A validator LLM checks it against a fixed checklist, and outputs a list of specific errors if applicable. A corrector LLM then applies minimal edits only to resolve the flagged issues. The process repeats until either the query becomes valid or a maximum of three correction attempts are reached. (C) Format of gold answers for single-step and multi-step retrieval tasks.

pathway) and traversal-direction violation of the bipartite path. These errors are largely removed when moving from zero- to one-shot prompting, regardless of the tested strategy: Careful selection of a good exemplar instead of a random one, or choosing a different prompt version, offer only marginal gains. We show a representative task in Figure 4, where the error taxonomy table reports that *invalid query* and *retrieval error* rates decrease when an example is provided in the input Prompt. Extended results are illustrated in Appendix Figures A7 and A8.

Checklist self-correction (CoVe) has limited benefits, validator is the bottleneck The CoVe-loop mainly assists in the zero-shot setting: It mostly repairs generated queries by reducing completeness errors, i.e. missing reaction components (reactants, products, agents). For example, *Reactants missing* and *Products missing* retrieval errors go down by $\sim 80\%$ in absolute count for single-step tasks in Prompt 2 in a zero-shot setting when CoVe is used, while in this case we see no clear improvement using the CoVe in the one-shot semantic settings. We report a summary of the retrieval errors in the Appendix Table A2. The limiting factor is the validator, not the correction step: A generic checklist misses up to $\sim 86\text{-}95\%$ (*Non-detected error rate*) of task-specific failures such as duplicate or incomplete molecules. We recommend investing in task-specific/schema-aware validators for future CoVe-loops.

4 Conclusions

We present a Text2Cypher pipeline over a reaction KG for chemical pathway planning, converting natural language questions into graph queries for both single- and multi-step reactions. We find that execution-grounded evaluation is essential: Text-to-text similarity for queries can be high while retrieval is incorrect. We further show that largest performance gains arise when moving from zero- to one-shot prompting, especially in the multi-step setting, with best results when the exemplar query is similar to the target query (task/intent). Lastly, a checklist-driven self-correction loop primarily improves executability on weak baselines, and yields only modest retrieval gains if an exemplar is already provided. Our results hold for the selected LLM and the designed knowledge graph, templates, and validator checks. Broader model comparisons, larger KGs, and task-specific validators are natural next steps. Our results provide practical guidelines for KG-grounded retrieval for reaction planning and indicate both challenges and exciting promises towards LLM-based reaction planning workflows.

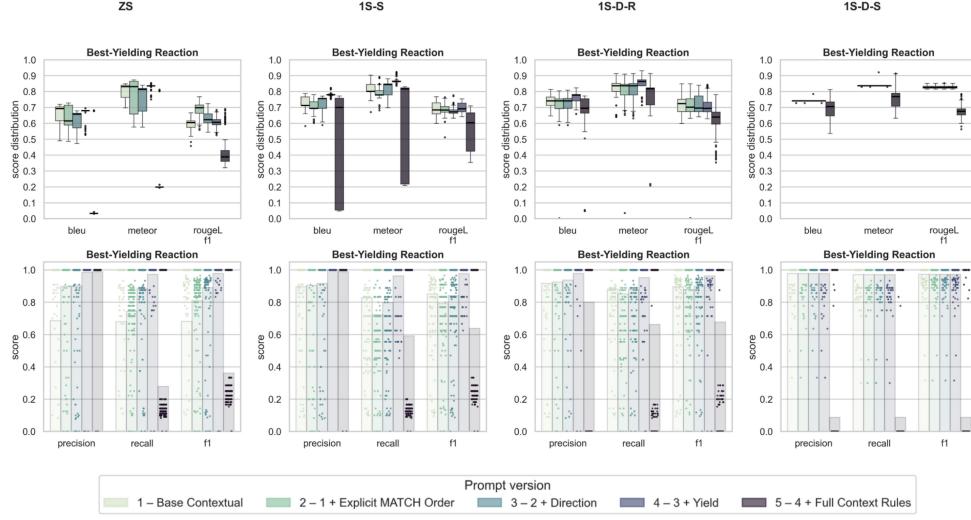


Figure 3: Text-to-text similarity (for Cypher queries, top row) and retrieval metrics (bottom row) for the single-step task Best-Yielding Reaction under different prompt settings: Zero-shot (ZS), one-shot static (1S-S), one-shot random (1S-D-R), and one-shot semantic (1S-D-S). Within each setting, five prompt versions (indicated in the bottom legend) add increasing contextual/structural guidance.

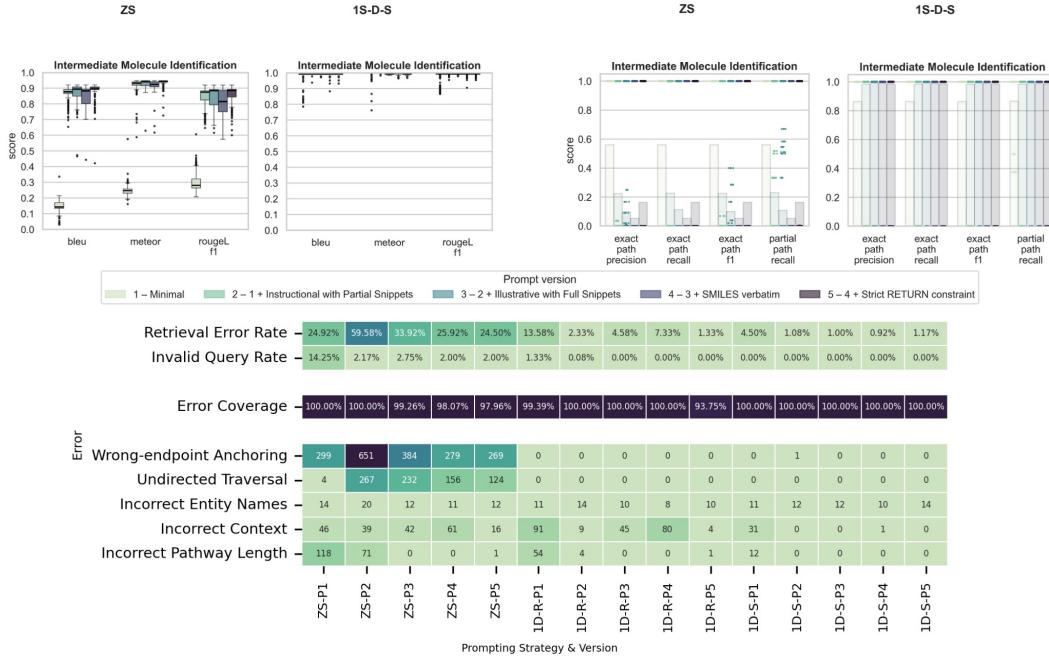


Figure 4: Text-to-text similarity (for Cypher queries, top, left) and retrieval metrics (top, right) for the multi-step task Intermediate Molecule Identification under zero-shot (ZS) and one-shot semantic (1S-D-S). One-shot largely removes zero-shot failures (anchoring, traversal direction, pathway length). Within each setting, five prompt versions add increasing contextual/structural guidance. The error taxonomy table (bottom) reports the rate of non-executable queries, incorrect retrievals, and their absolute counts across five error categories for all multi-step tasks, which cover almost all observed cases.

References

- [1] OpenAI Prompt Engineering Guide. URL <https://platform.openai.com/docs/guides/prompt-engineering/strategy-write-clear-instructions>.
- [2] Darren Edge, Ha Trinh, Newman Cheng, Joshua Bradley, Alex Chao, Apurva Mody, Steven Truitt, Dasha Metropolitansky, Robert Osazuwa Ness, and Jonathan Larson. From Local to Global: A Graph RAG Approach to Query-Focused Summarization. 4 2024. URL <http://arxiv.org/abs/2404.16130>.
- [3] G. Agrawal et al. Can knowledge graphs reduce hallucinations in llms?: A survey. *Proceedings of the 2024 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, 2024.
- [4] P. Seidl et al. Improving few- and zero-shot reaction template prediction using modern hopfield networks. *J. Chem. Inf. Model.*, vol. 62, no. 9, pp. 2111–2120, 2022.
- [5] T. B. Brown et al. Language models are few-shot learners. *arXiv*, doi: [10.48550/arXiv.2005.14165](https://doi.org/10.48550/arXiv.2005.14165), 2020.
- [6] N. Francis et al. Cypher: An evolving query language for property graphs. *Proceedings of the 2018 International Conference on Management of Data*, 2018.
- [7] Angel Garcia-Chung, Marisol Bermúdez-Montaña, Peter F. Stadler, Jürgen Jost, and Guillermo Restrepo. Chemically inspired Erdős–Rényi hypergraphs. *Journal of Mathematical Chemistry*, 62(6), 7 2024.
- [8] Guillaume Gricourt, Philippe Meyer, Thomas Duigou, and Jean Loup Faulon. Artificial Intelligence Methods and Models for Retro-Biosynthesis: A Scoping Review. *ACS Synthetic Biology*, 13(8), 2024.
- [9] M. Hornsteiner et al. Real-time text-to-cypher query generation with large language models for graph databases. doi: [10.3390/fi16120438](https://doi.org/10.3390/fi16120438), 2024.
- [10] Nourhan Ibrahim, Samar Aboulela, Ahmed Ibrahim, and Rasha Kashef. A survey on augmenting knowledge graphs (KGs) with large language models (LLMs): models, evaluation metrics, benchmarks, and challenges. *Discover Artificial Intelligence*, 4(1), 12 2024.
- [11] Amanda Kau, Xuzeng He, Aishwarya Nambissan, Aland Astudillo, Hui Yin, and Amir Aryani. Combining Knowledge Graphs and Large Language Models. 7 2024. URL <https://arxiv.org/abs/2407.06564v1>.
- [12] Steffen Klamt, Utz Uwe Haus, and Fabian Theis. Hypergraphs and Cellular Networks. *PLoS Computational Biology*, 5(5), 2009.
- [13] D. Kreutter, P. Schwaller, and J.-L. Reymond. Predicting enzymatic reactions with a molecular transformer. *Chem. Sci.*, vol. 12, no. 25, pp. 8648–8659, 2021.
- [14] Patrick Lewis, Ethan Perez, Aleksandra Piktus, Fabio Petroni, Vladimir Karpukhin, Naman Goyal, Heinrich Küttler, Mike Lewis, Wen Tau Yih, Tim Rocktäschel, Sebastian Riedel, and Douwe Kiela. Retrieval-Augmented Generation for Knowledge-Intensive NLP Tasks. *Advances in Neural Information Processing Systems*, 5 2020. URL <https://arxiv.org/abs/2005.11401v4>.
- [15] Songtao Liu, Dandan Zhang, Zhengkai Tu, Hanjun Dai, and Peng Liu. Evaluating Molecule Synthesizability via Retrosynthetic Planning and Reaction Prediction. 11 2024. URL <https://arxiv.org/pdf/2411.08306.pdf>.
- [16] Microsoft Research. Biomednlp-biomedbert-base-uncased-abstract. <https://huggingface.co/microsoft/BiomedNLP-BiomedBERT-base-uncased-abstract>, 2025. Hugging Face model card; accessed 2025-09-02.
- [17] Shervin Minaee, Tomas Mikolov, Narjes Nikzad, Meysam Chenaghlu, Richard Socher, Xavier Amatriain, and Jianfeng Gao. Large Language Models: A Survey. 2 2024. URL <https://arxiv.org/abs/2402.06196v2>.

- [18] Stefan Müller, Christoph Flamm, and Peter F. Stadler. What makes a reaction network “chemical”? *Journal of Cheminformatics*, 14(1), 12 2022.
- [19] Neo4j. URL <https://neo4j.com/>.
- [20] OpenAI. Gpt-4.1-mini. <https://platform.openai.com/docs/models#gpt-4-1-mini>, 2025. Model release, April 2025.
- [21] ORDerly. URL <https://github.com/sustainable-processes/ORDerly>.
- [22] Makbule Gulcin Ozsoy. Enhancing Text2Cypher with Schema Filtering. 5 2025. URL <https://arxiv.org/pdf/2505.05118v1.pdf>.
- [23] Makbule Gulcin Ozsoy, Leila Messallem, Jon Besga, and Gianandrea Minneci. Text2Cypher: Bridging Natural Language and Graph Databases. 12 2024. URL <https://arxiv.org/pdf/2412.10064.pdf>.
- [24] Bryan Perozzi, Bahare Fatemi, Dustin Zelle, Anton Tsitsulin, Mehran Kazemi, Rami Al-Rfou, and Jonathan Halcrow. Let Your Graph Do the Talking: Encoding Structured Data for LLMs. 2 2024. URL <https://arxiv.org/abs/2402.05862v1>.
- [25] Daniel Probst, Matteo Manica, Yves Gaetan Nana Teukam, Alessandro Castrogiovanni, Federico Paratore, and Teodoro Laino. Biocatalysed synthesis planning using data-driven learning. *Nature Communications* 13, no. 1, 2022.
- [26] Sander Schulhoff, Michael Ilie, Nishant Balepur, Konstantine Kahadze, Amanda Liu, Chenglei Si, Yinheng Li, Aayush Gupta, HyoJung Han, Sevien Schulhoff, Pranav Sandeep Dulepet, Saurav Vidyadhara, Dayeon Ki, Sweta Agrawal, Chau Pham, Gerson Kroiz, Feileen Li, Hudson Tao, Ashay Srivastava, Hevander Da Costa, Saloni Gupta, Megan L. Rogers, Inna Gonçarencio, Giuseppe Sarli, Igor Galynker, Denis Peskoff, Marine Carpuat, Jules White, Shyamal Anadkat, Alexander Hoyle, and Philip Resnik. The Prompt Report: A Systematic Survey of Prompt Engineering Techniques. 6 2024. URL <https://arxiv.org/pdf/2406.06608.pdf>.
- [27] P. Schwaller, T. Laino, T. Gaudin, P. Bolgar, C. Bekas, and A. A. Lee. Molecular transformer - a model for uncertainty-calibrated chemical reaction prediction. *ACS Cent. Sci.*, vol. 5, no. 9, pp. 1572–1583, 2019.
- [28] M. H. S. Segler, M. Preuss, and M. P. Waller. Planning chemical syntheses with deep neural networks and symbolic ai. *Nature*, vol. 555, no. 7698, pp. 604–610, 2018.
- [29] Sentence-Transformers. all-mpnet-base-v2. <https://huggingface.co/sentence-transformers/all-mpnet-base-v2>, 2021. Hugging Face model card; accessed 2025-09-02.
- [30] Felix Strieth-Kalthoff, Sara Szymkuć, Karol Molga, Alán Aspuru-Guzik, Frank Glorius, and Bartosz A. Grzybowski. Artificial Intelligence for Retrosynthetic Planning Needs Both Data and Expert Knowledge. *Journal of the American Chemical Society*, 2024.
- [31] S. M Towhidul Islam Tonmoy, S M Mehedi Zaman, Vinija Jain, Anku Rani, Vipula Rawte, Aman Chadha, and Amitava Das. A Comprehensive Survey of Hallucination Mitigation Techniques in Large Language Models. 1 2024. URL <https://arxiv.org/abs/2401.01313v3>.
- [32] A. Voll and W. Marquardt. Reaction network flux analysis: Optimization-based evaluation of reaction pathways for biorenewables processing. *AICHe Journal*, 2012. doi: 10.1002/aic.12704.
- [33] Jana M. Weber, Zhen Guo, Zhang Chonghuan, Artur M. Schweidtmann, and Alexei Lapkin. Chemical data intelligence for sustainable chemistry. *Chemical Society Reviews*, 2021. doi: 10.1039/D1CS00477H.
- [34] Jana M. Weber, Zhen Guo, and Alexei Lapkin. Discovering circular process solutions through automated reaction network optimization. *ACS Engineering Au*, 8 2022. doi: 10.1021/acsengineeringau.2c00002.
- [35] D. Weininger. Smiles, a chemical language and information system. 1. introduction to methodology and encoding rules. *J. Chem. Inf. Comput. Sci.*, vol. 28, no. 1, pp. 31–36, 1988.

- [36] Daniel S. Wigh, Joe Arrowsmith, Alexander Pomberger, Kobi C. Felton, and Alexei A. Lapkin. ORDerly: Data Sets and Benchmarks for Chemical Reaction Data. *Journal of Chemical Information and Modeling*, 64(9):3790–3798, 5 2024.
- [37] Qiang Zhang, Keyang Ding, et al. Scientific Large Language Models: A Survey on Biological & Chemical Domains. 1 2024. URL <https://arxiv.org/abs/2401.14656v2>.

5 Appendix

5.1 Prompt strategies

In our setting, prompt engineering starts with a baseline, minimal prompt. Based on subsequent error analysis, additional instructions and constraints are incrementally introduced to further improve the model’s context: It includes relevant schema and formatting rules, dataset-specific nuances, and guidance on the retrieval of full reaction contexts. The user prompt contains the input question, whereas the system prompt defines the model’s role as a helpful assistant for generating Cypher queries from natural language. In total, 5 Prompts were tested during the prompt engineering experiments (see Prompts P1- P5) for the single-step tasks, and 5 Prompts for the multi-step task (see Prompts P6- P10), where each prompt was designed following OpenAI’s guidelines on prompt engineering [1].

Static one-shot prompting In this setting, a single representative example is selected for all tasks. In particular we select an example corresponding to the most common query pattern. For the single-step example bank, it is Product Identification (shared across three out of six query types). For the multi-step example bank, these are Multi-Step Product Discovery and Forward Synthesis Intermediate Identification. The selected examples are kept the same across all tasks.

Dynamic one-shot: Random selection In this setting, we randomly select an example from the corresponding example bank. Since the chosen example may be dissimilar from the given input question, it serves as a baseline for evaluating exemplar quality for Cypher generation.

Dynamic one-shot: Embedding-based selection In this setting, the example bank is stored in a vector database. We use the sequence model listed in 5.4 to compute the embeddings for each example. The input question is embedded in the same way, and we select the one-shot exemplar by top-1 cosine similarity. To prevent chemistry specific influence, SMILES strings are masked in all queries, so similarity reflects task intent rather than chemical content. Because the example bank contains “opposite” tasks (forward vs retrosynthesis), logical intent descriptions are also added to all demonstrations, to prioritize *structural alignment* on the underlying task over surface-level semantic similarity. Here structural alignment denotes exemplar/query pairs sharing the same Cypher pattern type.

5.2 Exemplar selection for dynamic one-shot

In one-shot prompting strategies, the system prompt remains unchanged, containing the same guidelines as in a zero-shot setting, with an example pair of natural language question and the corresponding Cypher query now included in the user prompt. Exemplars for the one-shot setting are always extracted from an example bank. It contains questions similar in intent to those in the test dataset, but formulated in the reverse direction as forward synthesis tasks. This prevents the LLM from “copying” query patterns directly and instead evaluates how well it generalizes on the Text2Cypher task. The example banks are different from single- and multi-step tasks. We summarize them in Tables A1. Note that SMILES strings are intentionally excluded from the examples to prevent biasing Cypher generation with chemical context.

5.3 Chain-of-Verification (CoVe)

The Chain-of-Verification method is a self-criticism framework designed to verify LLM-generated responses. Given a generated answer, the model also creates a checklist to verify it. After each point on the checklist is addressed and the context is aggregated, a refined response is produced [26]. After the LLM generates the candidate query, its executability is checked with EXPLAIN in Neo4j. If the query is executable, a deterministic directionality corrector is applied. If the query is not executable, an LLM corrector is called to fix it based on the error message. For that, all SMILES strings are masked with placeholders to avoid special-character interference. The next step is the LLM validation against a fixed, task-specific checklist derived from prior error analysis. If the query is classified as valid, it is executed; otherwise, it is passed again to the LLM corrector to apply minimal edits based on the identified issues. The process repeats until either the validator labels the

Task	Query	Intent	Task	Query	Intent
Product Identification	What products are formed when molecule X reacts?	Identify other molecules with a specific role in the same reaction as a given molecule.	Reaction Pathway Discovery	List all possible multi-step reaction pathways of 5 steps, starting from molecule X ?	Find all reaction pathways of n steps where a given molecule is an endpoint.
Reactant-to-Reaction Mapping	Which one-step reactions involve molecule X as a reactant?	Find all reactions where the given molecule participates.	Reaction Chains Between Molecules	How can molecule B be synthesized from A in exactly 5 steps?	Find all reaction pathways of n steps from a given starting molecule to a given target molecule.
Best-Yielding Reaction Given Reactant	Which reaction involving molecule X as a reactant has the highest yield?	Rank reactions involving a given molecule based on their yield.	Multi-Step Product Discovery	What products are reachable from molecule X via exactly 5 reaction steps?	Find all molecules that are endpoints (precursors/products) of an n -step reaction pathway involving the specified molecule.
Multi-Precursor Reaction	Which single-step reactions use both molecule X_1 and X_2 in a single step?	Identify reactions that simultaneously involve all molecules in a given set.	Forward-Synthesis Intermediate Discovery	Which molecules appear as intermediates in forward reaction pathways starting from X with exactly 5 reaction steps?	Find all molecules that are intermediates of an n -step reaction pathway involving the specified molecule.
Co-reactant Identification	In reactions where molecule X is used as a reactant, what are the co-reactants?	Identify co-participant molecules in the same role as a given molecule within a reaction.			

Figure A1: Example banks for single-step (left) and multi-step (right) retrieval. Alongside each task we report their natural language queries and corresponding logical intent.

query as correct or a limit of three correction attempts is reached.

We report the distribution of retrieval errors for two Prompt version, P2 and P4, with and without self-correction, in Figure A2. *Wrong Reactant Directionality* errors are corrected by the deterministic directionality corrector, not by the LLM corrector, which explains most of the improvement for the zero-shot setting in the retrieval rate.

5.4 Implementation details

The USPTO dataset was preprocessed using the ORDERly [21] library, to reduce inconsistencies and improve data quality. For demonstration selection we use the Sentence Transformer model `all-mpnet-base-v2` [29]. The sentence encoder is BiomedNLP-BiomedBERT[16] (also available via Hugging Face), and the similarity threshold for cosine similarity is 0.93. Orchestration uses LangChain for prompting and LangGraph for workflow control; the KG backend is Neo4j.

5.5 Evaluation

Surface metrics for generated Cypher The generated Cypher queries are compared to the reference queries using BLEU, METEOR and ROUGE-L scores. Since Text2Cypher is a text-to-text generation task, these metrics are used for evaluating their similarity:

- **BLEU**: It measures how many n-grams (contiguous word sequences) in the generated text also appear in the reference text. It focuses on exact word overlap and is precision-oriented;
- **METEOR**: Extends BLEU by accounting for stemming (reducing words to their root form), synonyms, and word order, combining both precision and recall into a single score. It better captures meaning similarity than exact wording;
- **ROUGE-L**: Evaluates the longest common subsequence between the generated and reference texts, emphasizing how well the overall sequence and structure of words are preserved.

They all range from 0 to 1, where higher values indicate greater fidelity to the reference text.

Retrieval metrics The retrieved results are represented as lists of dictionaries with specific keys. Generated queries can present different dictionary structures and different key names compared to the reference query. For this reason, a key-matching procedure is required. We first normalize each key by lowercasing, trimming whitespace, converting camelCase to snake_case, and removing trailing `name/names`. The matching process involves three stages, illustrated in Figure A3:

- **Exact matching:** Normalized predicted keys are matched to ground truth keys based on string equality;
- **Rule-based and lexical matching:** Stemming is performed to reduce keys to their root form (e.g., agents → agent);
- **Embedding-based matching (semantic):** Remaining unmatched predicted keys are compared to gold keys using embedding similarity. Each prediction is assigned to the highest scoring key if the score $\geq \text{threshold}$. Notably, multiple predicted keys may map to a single gold key, so we aggregate their values during evaluation.

In the single-step setting, for each sample, True Positives, False Positives, and False Negatives are computed. They are then used to compute Precision, Recall and F1 metrics. These metrics are computed separately for each key (e.g. *products*, *reactants*, *solvents*), before sample-aggregation, so we refer to them as micro-averaged.

In the multi-step setting, the gold answer is a list of distinct reaction paths, represented by an ordered sequence of reaction nodes. Two paths are considered identical if their ordered id sequences are the same, eliminating the need for key matching. The two main metrics here are Exact Path Retrieval, that assigns a score of 0 to any incorrect path and 1 to exact path overlaps, and Partial Path Recall, which measures how much of the retrieved route overlaps with a continuous subpath of the reference (gold) pathway.

5.6 Single- and multi-step task results

Here we report results across all tasks and settings, for query generation and retrieval for single- (Figures A4, A5 and A6) and multi-step tasks (Figures A7 A8) respectively.

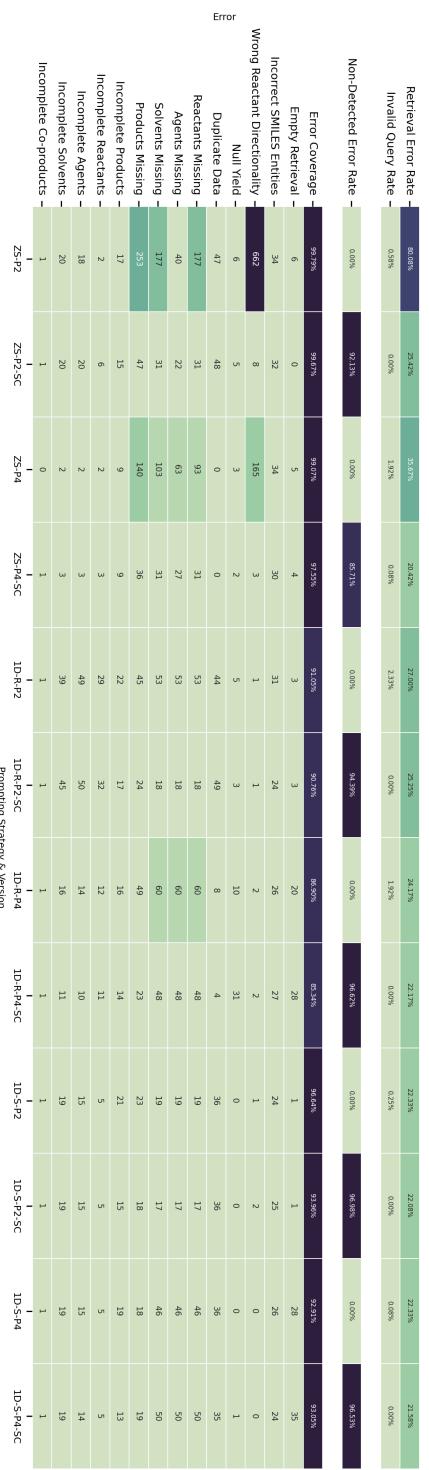


Figure A2: Distribution of retrieval errors for direct prompting vs. checklist-driven (CoVe style) prompting for single-step tasks. Error analysis covers three prompting strategies: Zero-shot (ZS), one-shot random (1S-D-R), and one-shot semantic (1S-D-S) and two Prompt versions, P2 and P4, with and without self-correction. *Retrieval error rate* is the fraction of executable queries with wrong retrieved context. *Non-detected error rate* is the fraction of retrieval error cases not flagged and corrected by the CoVe pipeline. *Error coverage* is the fraction of total retrieval errors that are assigned to the specific categories shown below.

GENERATED QUERY		GROUND TRUTH QUERY		Gold Key		Observed Variants (Matched)	
{	"products": [...], "directPrecursors": [...], "agentNames": [...], "solventNames": [...], "reaction_yield": [...]	{	"products": [...], "reactants": [...], "agents": [...], "solvents": [...], "target_yield": [...]	reactants		direct_precursors, precursor, precursor_name, precursor_names, precursors, reactants	
1	EXACT MATCHING	"products"	↔↔	products		precursors, product, productName, product_name, products, products1, products2, target_product	
2	1. LEXICAL-MORPHOLOGICAL MATCHING	"agentNames"	↔↔	agents		solvents	
		"solventNames"	↔↔	solvents		agent, agentName, agent_names, agents	
	2. RULE-BASED MATCHING	"reaction_yield"	↔↔	target_yield		best_yield, reaction_yield, rel.yield, yield	
3	SEMANTIC MATCHING	"directPrecursors"	↔↔	co_products		coProducts, co_product, co_product_name, co_product_names, co_products, coproducs, coproducts	
				product		mainProduct, main_product, produced_molecule, product, product_name, products, target_product, target_products	

Figure A3: Key-matching pipeline for single-step tasks (left), from generated query to ground truth, through sequential matching steps: Exact→lexical→semantic. On the right, a table of key variants observed and matched during the procedure.

Variation	Prompt																			
	ZS-P1	ZS-P2	ZS-P3	ZS-P4	ZS-P5	1S-P1	1S-P2	1S-P3	1S-P4	1S-P5	1D-R-P1	1D-R-P2	1D-R-P3	1D-R-P4	1D-R-P5	1D-S-P1	1D-S-P2	1D-S-P3	1D-S-P4	1D-S-P5
Aliases	100.0%	100.0%	100.0%	100.0%	100.0%	99.7%	99.2%	96.8%	99.3%	72.2%	100.0%	100.0%	98.7%	99.9%	95.3%	100.0%	100.0%	100.0%	100.0%	93.6%
WITH clause	80.5%	63.2%	72.0%	76.5%	97.9%	79.9%	78.1%	79.0%	73.7%	72.4%	74.9%	75.0%	74.3%	78.7%	81.7%	98.4%	98.4%	98.3%	98.5%	98.2%
Edge Directionality	12.8%	56.1%	93.5%	92.5%	100.0%	0.0%	0.0%	0.0%	0.0%	99.8%	4.9%	3.1%	7.2%	6.0%	99.7%	17.4%	17.0%	17.8%	17.8%	99.7%
Clause Order	99.2%	77.4%	76.6%	80.5%	100.0%	100.0%	99.7%	100.0%	100.0%	100.0%	81.4%	73.1%	68.7%	70.0%	98.3%	97.4%	85.5%	84.7%	85.8%	97.3%

Variation	Example	Affects Logic?
Alias names	MATCH (m) vs MATCH (molecule)	No
Direction of edges	(a) <-[:R] - (b) vs (b) - [:R] -> (a)	No (if flipped symmetrically)
Clause order	OPTIONAL MATCH (a) - [:REL1] -> (b) OPTIONAL MATCH (a) - [:REL2] -> (c) vs. reversed order	No (if clauses are independent)
Property ordering	RETURN a, b vs RETURN b, y	No

Figure A4: In the top row, we report the frequency of semantically neutral Cypher query variations, among generated queries that achieved perfect retrieval upon execution ($F1=1.0$), across four prompting strategies: Zero-shot (ZS), one-shot static (1S-S), one-shot random (1S-D-R) and one-shot semantic (1S-D-S). Each cell shows the percentage of queries that differ from the reference one in the corresponding category. In the bottom row, we expand more on the observed syntactic variations.

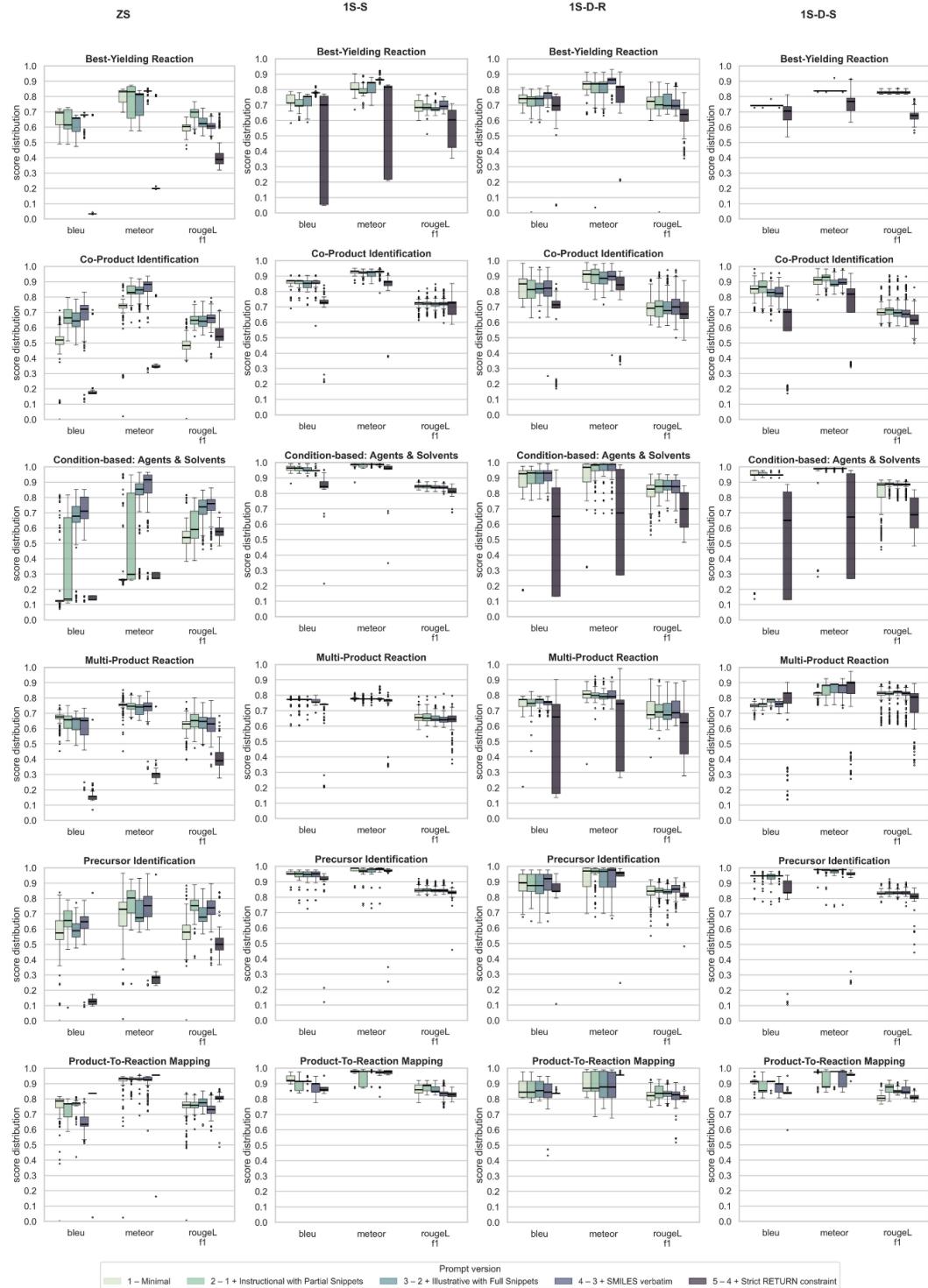


Figure A5: Comparison of generated Cypher queries to reference queries for single-step synthesis. Each subplot shows BLEU, METEOR, and ROUGE-L F1 score distributions for a specific query type under different prompt settings: Zero-shot (ZS), one-shot static (1S-S), one-shot random (1S-D-R), and one-shot semantic (1S-D-S). Within each setting, five prompt versions (color-coded) reflect increasing levels of contextual and structural guidance.



Figure A6: Single-step retrieval performance for generated Cypher queries vs ground truth. Each subplot shows precision, recall, and F1 score distributions for a specific query type under different prompt settings: Zero-shot (ZS), one-shot static (1S-S), one-shot random (1S-D-R), and one-shot semantic (1S-D-S). Within each setting, five prompt versions (color-coded) reflect increasing levels of contextual and structural guidance.

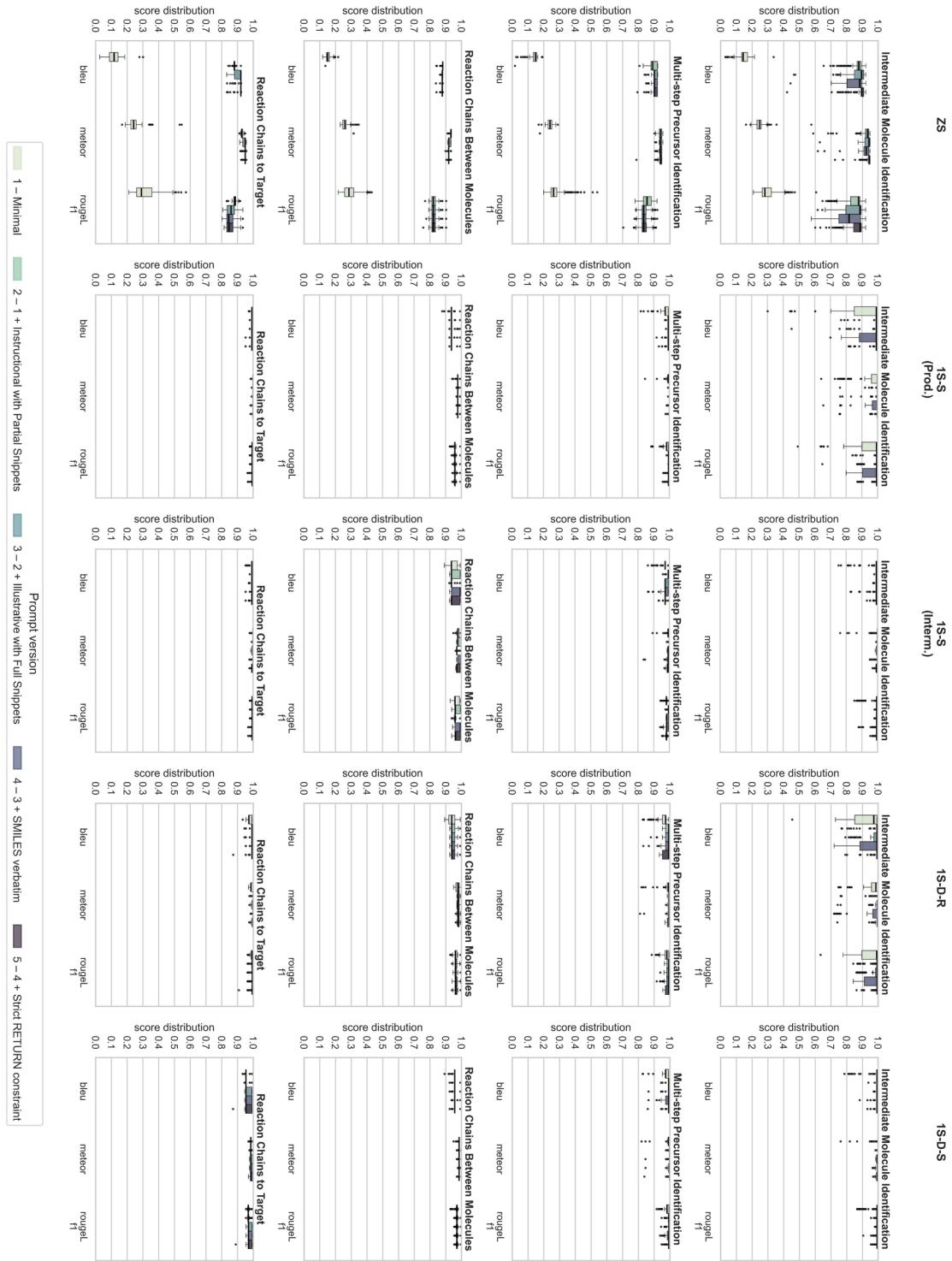


Figure A7: Comparison of generated Cypher queries to reference queries for multi-step synthesis. Each subplot shows BLEU, METEOR, and ROUGE-L F1 score distributions for a specific query type under different prompt settings: Zero-shot (ZS), one-shot static (1S-S; check 5.1 for more), one-shot random (1S-D-R), and one-shot semantic (1S-D-S). Within each setting, five prompt versions (color-coded) reflect increasing levels of contextual and structural guidance.

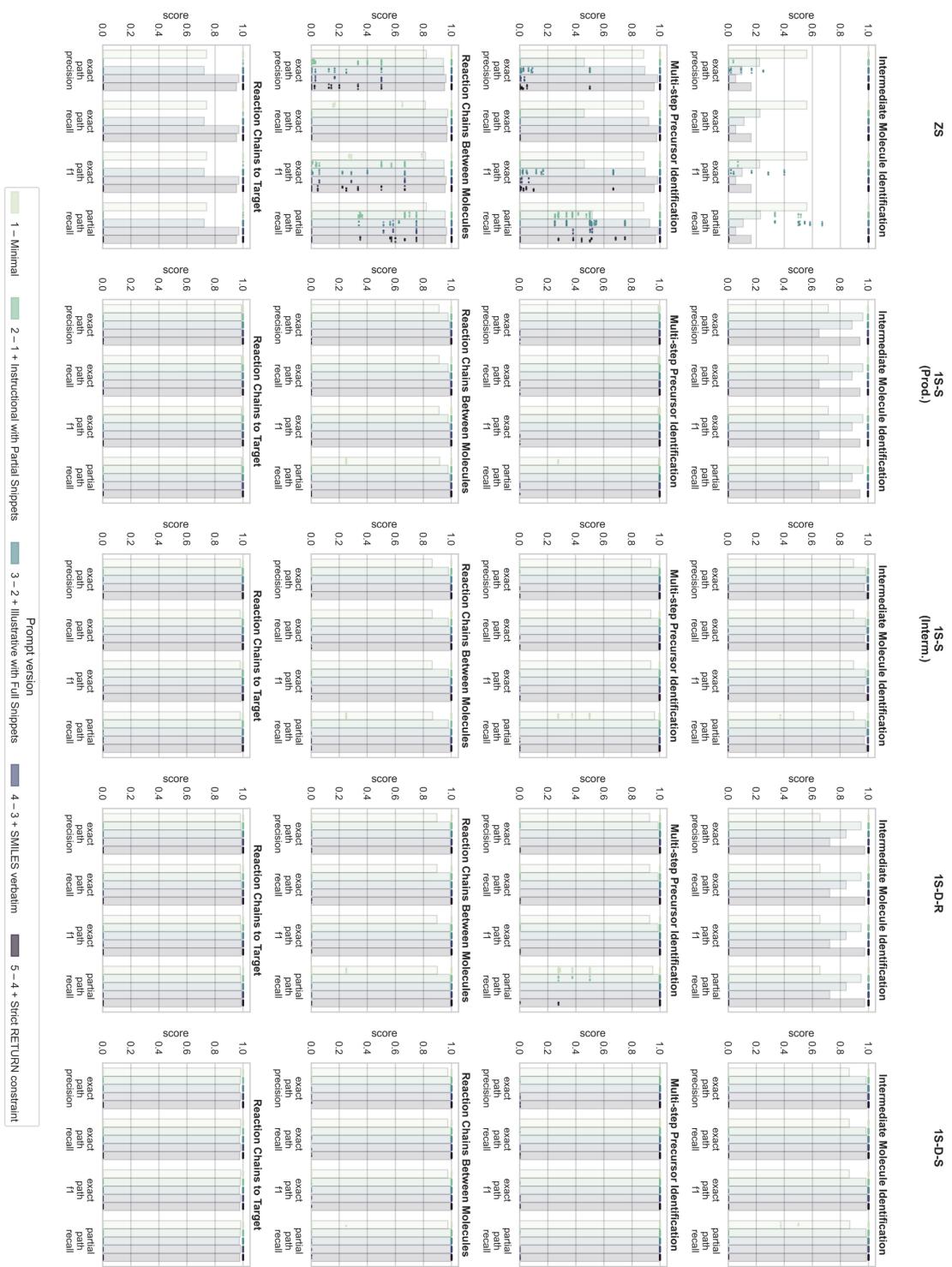


Figure A8: Multi-step retrieval performance for generated Cypher queries vs ground truth. Each subplot shows exact-path precision, recall, F1, and partial-path recall for a specific query type. We use different prompting strategies: Zero-shot (ZS), one-shot static (1S-S; check 5.1 for more), one-shot random (1S-D-R), and one-shot semantic (1S-D-S). Within each setting, five prompt versions (color-coded) reflect increasing levels of contextual and structural guidance.

P1: Single-Step Synthesis Prompt – Base Contextual

You are an expert in generating Cypher queries for a Neo4j knowledge graph with the following schema:

```
<schema>
{schema}
</schema>
```

Your task is to generate **valid, semantically meaningful** Cypher queries based solely on user input and the schema.

General Guidelines

- Always adhere to correct Cypher syntax.
- Return only the Cypher query enclosed in triple backticks — nothing else.

Contextual Retrieval

For each Reaction node retrieved, always include full context:

- Always use MATCH or OPTIONAL MATCH to bind variables for reactants, products, agents, and solvents.
- After binding, retrieve their names using collect(DISTINCT <variable>.name) in the RETURN clause.

User Question

```
<user_question>
{question}
</user_question>
```

P2: Single-Step Synthesis Prompt 2 – (1) + Explicit MATCH Order

You are an expert in generating Cypher queries for a Neo4j knowledge graph with the following schema:

```
<schema>
{schema}
</schema>
```

Your task is to generate **valid, semantically meaningful** Cypher queries based solely on user input and the schema.

General Guidelines

- Always adhere to correct Cypher syntax.
- Return only the Cypher query enclosed in triple backticks — nothing else.

Contextual Retrieval

For each Reaction node retrieved, always include full context:

- Always use MATCH or OPTIONAL MATCH to bind variables for reactants, products, agents, and solvents.
- After binding, retrieve their names using collect(DISTINCT <variable>.name) in the RETURN clause.

User Question

```
<user_question>
{question}
</user_question>
```

P3: Single-Step Synthesis Prompt 3 – (2) + Direction

You are an expert in generating Cypher queries for a Neo4j knowledge graph with the following schema:

```
<schema>
{schema}
</schema>
```

Your task is to generate **valid, semantically meaningful** Cypher queries based solely on user input and the schema.

General Guidelines

- Always adhere to correct Cypher syntax.
- Return only the Cypher query enclosed in triple backticks — nothing else.

Relationship Directionality (Mandatory)

Ensure every relationship arrow **matches the direction shown in the schema**.

- (:Molecule)-[:REACTS_IN]->(:Reaction)
- (:Reaction)-[:PRODUCES]->(:Molecule)
- (:Reaction)-[:USES_AGENT]->(:Molecule)
- (:Reaction)-[:USES_SOLVENT]->(:Molecule)

Contextual Retrieval

For each Reaction node retrieved, always include full context:

- Always use MATCH or OPTIONAL MATCH to bind variables for reactants, products, agents, and solvents.
- After binding, retrieve their names using collect(DISTINCT <variable>.name) in the RETURN clause.

User Question

```
<user_question>
{question}
</user_question>
```

P4: Single-Step Synthesis Prompt 4 – (3) + Yield

You are an expert in generating Cypher queries for a Neo4j knowledge graph with the following schema:

```
<schema>
{schema}
</schema>
```

Your task is to generate **valid, semantically meaningful** Cypher queries based solely on user input and the schema.

General Guidelines

- Always adhere to correct Cypher syntax.
- Return only the Cypher query enclosed in ` ` — nothing else.

Yield Handling

If yield is used (e.g., in filter or sort), add WHERE <rel>.yield IS NOT NULL before any sorting or limiting.

Relationship Directionality (Mandatory)

Ensure every relationship arrow **matches the direction shown in the schema**:

- (:Molecule)-[:REACTS_IN]->(:Reaction)
- (:Reaction)-[:PRODUCES]->(:Molecule)

- (:Reaction)-[:USES_AGENT]->(:Molecule)
- (:Reaction)-[:USES_SOLVENT]->(:Molecule)

Contextual Retrieval

For each Reaction node retrieved, always include full context:

- Always use MATCH or OPTIONAL MATCH to bind variables for reactants, products, agents, and solvents.
- After binding, retrieve their names using collect(DISTINCT <variable>.name) in the RETURN clause.

User Question

```
<user_question>
{question}
</user_question>
```

P5: Single-Step Synthesis Prompt 5 – (4) + Full Context Rules

You are an expert in generating Cypher queries for a Neo4j knowledge graph with the following schema:

```
<schema>
{schema}
</schema>
```

Your task is to generate **valid, semantically meaningful** Cypher queries based solely on user input and the schema.

General Guidelines

- Always adhere to correct Cypher syntax.
- Return only the Cypher query enclosed in ` ` — nothing else.

Yield Handling If yield is used (e.g., in filter or sort), add WHERE <rel>.yield IS NOT NULL before any sorting or limiting.

Relationship Directionality (Mandatory)

Ensure every relationship arrow **matches the direction shown in the schema**:

- (:Molecule)-[:REACTS_IN]->(:Reaction)
- (:Reaction)-[:PRODUCES]->(:Molecule)
- (:Reaction)-[:USES_AGENT]->(:Molecule)
- (:Reaction)-[:USES_SOLVENT]->(:Molecule)

Contextual Retrieval

When the query involves a molecule (e.g., asking what produces it, what it reacts in, or its precursors) — follow this pattern:

1. Match all reactions involving the molecule, for example:

```
MATCH (target:Molecule {{name: "..."}})-[:PRODUCES]-(r:Reaction)
```
2. Use OPTIONAL MATCH to retrieve all possible related molecules:
 - (reactant:Molecule)-[:REACTS_IN]->(r:Reaction)
 - (r:Reaction)-[:PRODUCES]->(product:Molecule)
 - (r:Reaction)-[:USES_AGENT]->(agent:Molecule)
 - (r:Reaction)-[:USES_SOLVENT]->(solvent:Molecule)
3. Return them as collect(DISTINCT ...) lists, e.g.:

```
RETURN r.id,
       collect(DISTINCT reactant.name) AS reactants,
       collect(DISTINCT product.name) AS products,
       collect(DISTINCT agent.name) AS agents,
       collect(DISTINCT solvent.name) AS solvents
```

User Question

```
<user_question>
{question}
</user_question>
```

P6: Multi-Step Synthesis Prompt 1 – Minimal

You are an expert in generating Cypher queries for a Neo4j knowledge graph with the following schema:

```
<schema>
{schema}
</schema>
```

Your task is to generate **valid, semantically meaningful** Cypher queries based solely on user input and the schema.

General Guidelines:

- Always adhere to correct Cypher syntax.
- Return only the Cypher query enclosed in triple backticks — nothing else.

Important Assumptions

- Regardless of the user's question intent (e.g., asking about precursors, agents, intermediates), you must return only the full reaction chains — that is, the Reaction nodes involved in the synthesis pathway.
- Do not perform any reasoning or interpretation — simply identify the sequence of Reaction nodes involving the mentioned molecular entities.

Graph Constraints:

- The graph is bipartite: Molecule and Reaction nodes alternate.
- A synthesis path of N steps has $2 \times N$ hops.

User Question

```
<user_question>
{question}
</user_question>
```

P7: Multi-Step Synthesis Prompt 2 – (1) + Query Generation Instructions with Partial Cypher Snippets

You are an expert in generating Cypher queries for a Neo4j knowledge graph with the following schema:

```
<schema>
{schema}
</schema>
```

Your task is to generate **valid, semantically meaningful** Cypher queries based solely on user input and the schema.

General Guidelines:

- Return only the Cypher query enclosed in triple backticks — nothing else.

Important Assumptions

- Regardless of the user's question intent (e.g., asking about precursors, agents, intermediates), you must return only the full reaction chains — that is, the Reaction nodes involved in the synthesis pathway.
- Do not perform any reasoning or interpretation — simply identify the sequence of Reaction nodes involving the mentioned molecular entities.

Graph Structure and Path Lengths

- The graph is bipartite: Molecule and Reaction nodes alternate.
- A synthesis path of N steps has $Y = 2 \times N$ hops (e.g., 2 → 4, 3 → 6, 4 → 8).

Query Constraints

- Use a variable-length pattern with `[:REACTS_IN | PRODUCES*..Y]` to enable multi-step synthesis paths.
- Enforce *exact* length with:
`WHERE size(relationships(p)) = Y`
- Ensure bipartite alternation of nodes: Molecule (even), Reaction (odd).
- Return `DISTINCT reaction_nodes` only.

User Question

```
<user_question>
{question}
</user_question>
```

P8: Multi-Step Synthesis Prompt 3 – (2) + Illustrative Query Generation Instructions with Full Cypher Snippets

You are an expert in generating Cypher queries for a Neo4j knowledge graph with the following schema:

```
<schema>
{schema}
</schema>
```

Your task is to generate **valid, semantically meaningful** Cypher queries based solely on user input and the schema.

General Guidelines:

- Always adhere to correct Cypher syntax.
- Return only the Cypher query enclosed in triple backticks — nothing else.

Important Assumptions

- Regardless of the user's question intent (e.g., asking about precursors, agents, intermediates), you must return only the full reaction chains — that is, the reaction nodes involved in the synthesis pathway.
- Do not perform any reasoning or interpretation — simply identify the sequence of reaction nodes involving the mentioned molecular entities.

Graph Structure and Path Lengths The graph is **bipartite**:

- Molecule and Reaction nodes alternate.
- A synthesis path of N steps has path length $Y = 2 \times N$ hops (e.g., 2 → 4, 3 → 6, 4 → 8).

Query Constraints

1. Match Multi-Hop Paths

Use a variable-length pattern with `[:REACTS_IN | PRODUCES*..Y]` to enable multi-step synthesis paths. Enforce *exact* length with:

```
WHERE size(relationships(p)) = Y
```

2. Enforce Bipartite Alternation:

```
WHERE all(i IN range(0, size(nodes(p)) - 1)
      WHERE (i % 2 = 0 AND 'Molecule' IN labels(nodes(p)[i])) OR
            (i % 2 = 1 AND 'Reaction' IN labels(nodes(p)[i])))
```

3. Return Only Reaction Nodes:

```
WITH [x IN nodes(p) WHERE 'Reaction' IN labels(x)] AS reaction_nodes
RETURN DISTINCT reaction_nodes
```

User Question

```
<user_question>
{question}
</user_question>
```

P9: Multi-Step Synthesis Prompt 4 – (3) + SMILES verbatim

You are an expert in generating Cypher queries for a Neo4j knowledge graph with the following schema:

```
<schema>
{schema}
</schema>
```

Your task is to generate **valid, semantically meaningful** Cypher queries based solely on user input and the schema.

General Guidelines:

- Always adhere to correct Cypher syntax.
- Return only the Cypher query enclosed in triple backticks — nothing else.

Important Assumptions

- Regardless of the user's question intent (e.g., asking about precursors, agents, intermediates), you must return only the full reaction chains — that is, the Reaction nodes involved in the synthesis pathway.
- Copy SMILES verbatim: character-for-character — no changes to atoms, case, ring numbers, parentheses/brackets, or charges.
- Do not perform any reasoning or interpretation — simply identify the sequence of Reaction nodes involving the mentioned molecular entities.

Graph Structure and Path Lengths The graph is **bipartite**:

- Molecule and Reaction nodes alternate.
- A synthesis path of N steps has path length $Y = 2 \times N$ hops (e.g., 2 → 4, 3 → 6, 4 → 8).

Query Constraints

1. Match Multi-Hop Paths

Use a variable-length pattern with `[:REACTS_IN | PRODUCES*.. Y]` to enable multi-step synthesis paths. Enforce *exact* length with:

```
WHERE size(relationships(p)) = Y
```

2. Enforce Bipartite Alternation:

```
WHERE all(i IN range(0, size(nodes(p)) - 1)
      WHERE (i % 2 = 0 AND 'Molecule' IN labels(nodes(p)[i])) OR
            (i % 2 = 1 AND 'Reaction' IN labels(nodes(p)[i])))
```

3. Return Only Reaction Nodes:

```
WITH [x IN nodes(p) WHERE 'Reaction' IN labels(x)] AS reaction_nodes
RETURN DISTINCT reaction_nodes
```

User Question

```
<user_question>
{question}
</user_question>
```

P10: Multi-Step Synthesis Prompt 5 – (4) + Strict RETURN Constraint

You are an expert in generating Cypher queries for a Neo4j knowledge graph with the following schema:

```
<schema>
{schema}
</schema>
```

Your task is to generate **valid, semantically meaningful** Cypher queries based solely on user input and the schema.

General Guidelines:

- Always adhere to correct Cypher syntax.
- Return only the Cypher query enclosed in triple backticks — nothing else.

Important Assumptions

- Regardless of the user's question intent (e.g., asking about precursors, agents, intermediates), you must return only the full reaction chains — that is, the Reaction nodes involved in the synthesis pathway.
- Copy SMILES verbatim: character-for-character — no changes to atoms, case, ring numbers, parentheses/brackets, or charges.
- Do not perform any reasoning or interpretation — simply identify the sequence of Reaction nodes involving the mentioned molecular entities.

Graph Structure and Path Lengths

The graph is **bipartite**:

- Molecule and Reaction nodes alternate.
- A synthesis path of N steps has path length $Y = 2 \times N$ hops (e.g., 2 → 4, 3 → 6, 4 → 8).

Query Constraints

1. Match Multi-Hop Paths

Use a variable-length pattern with `[:REACTS_IN | PRODUCES*.. Y]` to enable multi-step synthesis paths. Enforce *exact* length with:

```
WHERE size(relationships(p)) = Y
```

2. Enforce Bipartite Alternation:

```
WHERE all(i IN range(0, size(nodes(p)) - 1)
      WHERE (i % 2 = 0 AND 'Molecule' IN labels(nodes(p)[i])) OR
            (i % 2 = 1 AND 'Reaction' IN labels(nodes(p)[i])))
```

3. Return Only Reaction Nodes:

```
WITH [x IN nodes(p) WHERE 'Reaction' IN labels(x)] AS reaction_nodes
RETURN DISTINCT reaction_nodes
```

Output constraint: Only return `reaction_nodes`. Do *not* return Molecule nodes, `p`, `nodes(p)`, `relationships(p)`, or anything else. No extra `RETURNS`s.

User Question

```
<user_question>
{question}
</user_question>
```