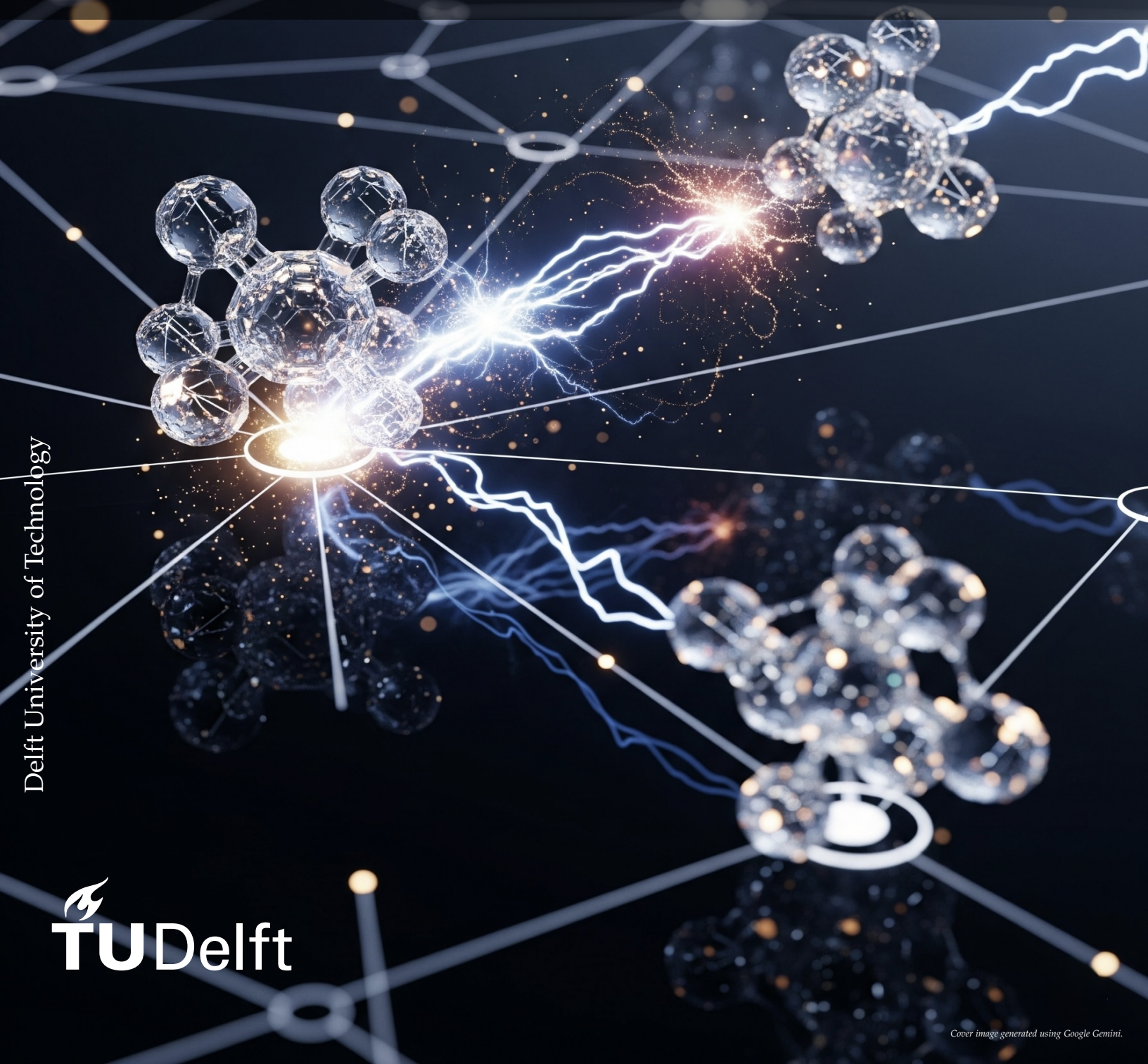
Grounding Large Language Models in Reaction Knowledge Graphs for Synthesis Retrieval

Master Thesis
Olga Bunkova



Delft University of Technology

TU Delft

Cover image generated using Google Gemini.

Grounding Large Language Models in Reaction Knowledge Graphs for Synthesis Retrieval

by

Olga A. Bunkova

to obtain the degree of Master of Science
at Delft University of Technology,
to be defended publicly on
September 15th, 2025 at 10:00.

Student number: 4915917
Project duration: December 2024 – September 2025

Thesis committee:

Prof. dr. ir. M. J. T. Reinders	Thesis advisor
Dr. J. M. Weber	Daily supervisor
L. Di Fruscia, MSc	Daily co-supervisor
S. Rupprecht, MSc	Daily co-supervisor
Dr. C. Lofi	External Committee Member

Preface

I am deeply grateful to Prof. dr. ir. M. J. T. Reinders, who took over the supervision of my thesis project and provided invaluable feedback; to Dr. Jana Weber for supervising the initial phase of this project and for her early guidance; to Lorenzo Di Fruscia and Sophia Rupprecht for their guidance, brainstorming sessions, and support; and to Dr. C. Lofi for being part of my thesis committee. I also sincerely thank my boyfriend, family, and friends for their constant support.

Olga Bunkova
Delft, the Netherlands
September 9, 2025

Leveraging Molecular Reaction Graphs and Large Language Models for Advanced Reaction Planning

O. Bunkova, L. Di Fruscia, S. Rupprecht, M.J.T. Reinders, J. Weber

Abstract—Grounding Large Language Models (LLMs) in chemical knowledge graphs offers a promising way to support synthesis planning, but reliably retrieving information from these complex structures remains a challenge. Therefore, this work addresses that gap by constructing a bipartite KG and evaluating Text2Cypher query generation across both single- and multi-step retrieval tasks. Different prompting strategies were tested, including zero-shot, one-shot with static, random, or embedding-based example selection, and a checklist-driven self-correction pipeline. Results indicate that one-shot prompting is most effective when the exemplar aligns with the query both structurally and logically. When such an exemplar is provided as context to the Cypher generation prompt, self-correction does not yield significant performance gains. Overall, this study introduces a reproducible setup for Text2Cypher experimentation and evaluation.

1 Introduction

Large language models (LLMs) have led to significant advances in the domain of cheminformatics thanks to their ability to process and interpret molecular data [1]. However, current research applying LLMs to tasks such as reaction and retrosynthesis planning remains limited. In particular, unlike massive amounts of data available for natural language, chemical reaction datasets are still relatively small. For instance, one of the main datasets, the USPTO (United States Patent and Trademark Office reactions), only contains 1.7 million reactions in the Open Reaction Database (ORD) [2]. In contrast, the text corpora used to train LLMs are several orders of magnitude larger, such as the training dataset of LLAMA 3 consisting of 15 trillion tokens [3].

Furthermore, data quality poses yet another challenge here due to incomplete or missing information, annotation errors, standardization issues, low chemical diversity, and duplicate entries [4]. Encoding chemistry also requires specialized representation formats, e.g., Simplified Molecular Input Line Entry System (SMILES) notation [1]. LLMs, however, still lack an extensive inherent understanding of SMILES in the same way that they are able to understand natural language [5], [6].

Another key aspect to consider is that LLMs do not only have limited domain-specific knowledge in chemistry, but also suffer from hallucinations. In particular, they might produce plausible yet factually incorrect outputs [7]. Additionally, pre-trained knowledge could potentially become stale or outdated, which presents a major challenge for rapidly evolving fields [8]. Therefore, different knowledge augmentation methods have been proposed, for instance, retrieval augmented generation (RAG) [9]. Even though this approach ensures information-freshness and reduces hallucinations, it does not make use of the global structure of the underlying data [10], [11].

A promising alternative is knowledge graph (KG)-enhanced LLMs, which involves using KGs as an external knowledge base to enhance the generation process [12]. Unlike naive RAG that only leverages local information, KGs provide a structured knowledge representation that explicitly encodes entities, relations, and properties, preserving global, long-range dependencies within the data. As a result, they enable path-constrained retrieval and multi-hop reasoning [13], [14], which is of particular interest for the task of synthesis planning.

This motivates the central premise of this work, that is, synthesis planning could benefit from coupling LLMs with a reaction KG. Such coupling grounds model outputs in domain knowledge, reduces hallucinations, and eliminates the need for task-specific fine-tuning. In this setting, the main performance bottleneck becomes retrieval via query generation, or in other words, whether an LLM can produce syntactically valid and semantically correct Cypher queries. Query generation is chosen primarily because of its flexibility, which is discussed in more detail in Sec. 2.4.3.

Therefore, this study aims to answer the following research questions:

- **RQ1:** How can knowledge graphs be structured and utilized for reaction prediction and synthesis planning?
- **RQ2:** To what extent can LLMs grounded in a reaction knowledge graph generate syntactically valid and semantically accurate Cypher queries?
- **RQ3:** How does the in-context learning strategy (zero-shot vs. few-shot, static vs. semantic selection) affect Cypher generation quality and retrieval performance across synthesis tasks?
- **RQ4:** What is the frequency and distribution of retrieval errors, and how are they influenced by different prompt strategies?

2 BACKGROUND

2.1 LLMs, In-Context Learning, and Prompt Engineering

Large Language Models (LLMs) refer to large-scale, pre-trained, transformer-based models with advanced capabilities on various natural language processing (NLP) tasks, such as question answering and text summarization [8].

A notable phenomenon observed in LLMs is in-context learning (ICL), which is the process of performing a task conditioned on specific instructions and/or demonstrations provided within the input context, without requiring any parameter updates. A related concept is prompt engineering, defined as the design and refinement of such inputs (called prompts) to optimize the model's output [15]. Different prompting techniques have been introduced in prior research, among which are zero-shot, few-shot, and self-criticism strategies [15], [16].

2.1.1 Zero-Shot Prompting

In zero-shot prompting, an LLM is given a task along with helpful guidelines, and it generates a response using only its pre-trained knowledge. A common subtype is role prompting, where a specific role is assigned to the LLM, e.g., acting as a helpful assistant for Cypher query generation. In particular, this can help guide the model toward domain-relevant reasoning, which may lead to more accurate responses in some cases [15], [17].

2.1.2 Few-Shot Prompting

Few-shot prompting is a technique, where an LLM is provided a few input-output examples (exemplars) to guide the generation process. Typically, when the chosen examples are similar to the test sample, the performance of the model improves [18]. Overall, selecting demonstrations is a difficult task with a variety of approaches. Some of the most common ones include static hand-crafted selection, random sampling, or similarity-based retrieval of exemplars. [19]. The former two are fairly simple to set up, and are often used as baselines in ICL research [19], [20].

It is important to note that there is no universal strategy for effective demonstration selection. It remains an empirical process that is often dependent on the dataset and task at hand [21]. Similarly, prompt formatting and style can significantly impact an LLM's performance, and thus also require an experimental approach [15].

2.1.3 Self-Criticism

The Chain-of-Verification (CoVe) 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, the final refined response is produced [15].

2.2 Retrosynthesis Prediction & Synthesis Planning

Chemical synthesis planning is a fundamental task in the domain of chemistry and cheminformatics, and is defined as the process of designing a sequence of chemical reactions to produce a target molecule from available starting materials [22], [23]. Single-step reaction planning focuses on predicting products given a set of reactants, whereas single-step retrosynthesis planning is the inverse problem of predicting reactants given a target product [23].

Single-step retrosynthesis prediction methods are typically divided into *template-based*, which apply atom-mapped reaction rules (templates), and *template-free* or data-driven, which include generative sequence-to-sequence (seq2seq) and graph-based models [23]. Multi-step planning is formulated as a two-stage search problem. First, a single-step retrosynthesis model predicts sets of candidate precursors. Second, a search algorithm (e.g., Monte Carlo Tree Search or A*) evaluates these predictions and expands upon them to assemble a complete route [23], [24].

The goal of this study is not to replace state-of-the-art methods, but to examine whether off-the-shelf, non-fine-tuned LLMs can effectively be used for synthesis tasks. Recent work shows that they struggle with direct chemical generation [25], [26], [27], but might be of added value inside augmented pipelines, e.g., leveraging external tools or guiding the search process. Hence, the project focuses on grounding an LLM in external knowledge, specifically a reaction knowledge graph, and evaluating its ability to reliably interact with that KG through retrieval.

2.3 Knowledge Graphs

Knowledge graphs (KGs) are a type of directed graphs that store data as triples in a structured fashion. Formally, KG is defined as:

$$KG = \{(h, r, t) \subseteq E \times R \times E\}, \tag{1}$$

where h, r, t represent head entity, relation, and tail entity respectively, E denotes the set of entities and R corresponds to the set of relations [12], [28].

2.4 Knowledge Graph-enhanced LLMs

2.4.1 Limitations of LLMs: Integrating External Knowledge

Despite their remarkable capabilities in natural language processing (NLP), LLMs are not without limitations. In particular, they might not always generalize effectively in specialized domains due to knowledge gaps, biases, and inaccuracies in the training data. Out of the box, LLMs also lack access to up-to-date information unless, e.g., they are retrained or augmented with external tools. As a result, these limitations make them prone to hallucinations, which are responses that seem plausible but are factually incorrect [7], [29], [30].

2.4.2 RAG vs KG-enhanced LLMs & GraphRAG

One way to address this issue is to incorporate external knowledge through RAG, which consists of two key components: a retriever and an LLM. Here, the retriever computes an embedding for the input query and performs semantic search to fetch the most relevant documents based on embedding similarity. These documents are provided as context for the LLM, guiding the generation of the final response [9]. Nonetheless, a key limitation of the standard RAG approach is that it fails to preserve hierarchical or conceptual dependencies within the data. This is because documents are treated as independent units, that is, they are retrieved in isolation. [29], [30].

An alternative solution is KG-enhanced LLMs, where a KG serves as an external source of structured knowledge during inference. These graphs explicitly encode relationships between entities, providing global relational and structural context beyond the local scope of individual documents/data samples [31]. Therefore, such systems can support **multi-hop reasoning**, by incorporating information across interconnected nodes and their edges [29], [30], [32].

This could be of particular interest for the task of reaction and retrosynthesis planning, where the underlying data is inherently graph-structured. By preserving both the order and directionality of each transformation, the framework enables the traversal and retrieval of multi-step reaction pathways.

2.4.3 Retrieval Patterns in KG-enhanced LLMs

Multiple retrieval patterns have been explored in the literature for extracting information from KGs and incorporating it as context during the generation process. In particular, these include graph-enhanced vector search (*hybrid retrieval*), global community summary retrieval (*vector-based retrieval*), and Text2Cypher (*query generation*) [33], which are illustrated in Fig. 1. Note that this list is not exhaustive, but captures several of the simplest and most representative approaches.

Graph-enhanced vector search (Fig. 1a) is a two-stage hybrid retrieval strategy, which may also include additional pre-processing and post-processing steps. In general, it first performs vector similarity search to, e.g., find top-k nodes corresponding to relevant documents. In the second step, graph traversal is performed to retrieve additional context, such as the immediate neighborhood of each node. When integrated with a graph database, this traversal is usually implemented by executing a query on the KG. Finally, the retrieved context, along with the original question, is fed into the LLM to guide the generation process [33]. An example of such is LightRAG [34], which extracts keywords from queries followed by vector similarity search to match them to relevant entities in the KG. The 1-hop neighborhoods of these entities are then gathered as additional context. HybridRAG is also based on a similar idea [35], but instead the traversal occurs simultaneously with vector search. Both the relevant graph context and top-k retrieved documents are jointly fed into the LLM.

One of the earliest and most cited works on KG-enhanced LLMs is Microsoft's GraphRAG [11], which introduced a global community summary retrieval pattern (Fig. 1b). Essentially, the retrieval itself resembles that of the Naive RAG due to the unique way the KG is built: it is multi-level with a hierarchical structure. This KG is constructed by extracting entities and relations through multiple LLM calls, and is subsequently partitioned into hierarchical communities using community detection algorithms. Each community is summarized, once again, using an LLM, and during inference, semantic search is performed over these summaries to retrieve the most relevant ones.

Text2Cypher (along with its equivalents such as Text2SPARQL) represents a highly flexible retrieval pattern [33], depicted in (Fig. 1c). Here, an LLM is used to generate queries to be executed on the KG, which can be achieved through fine-tuning ([36], [37]) or prompt engineering ([38],

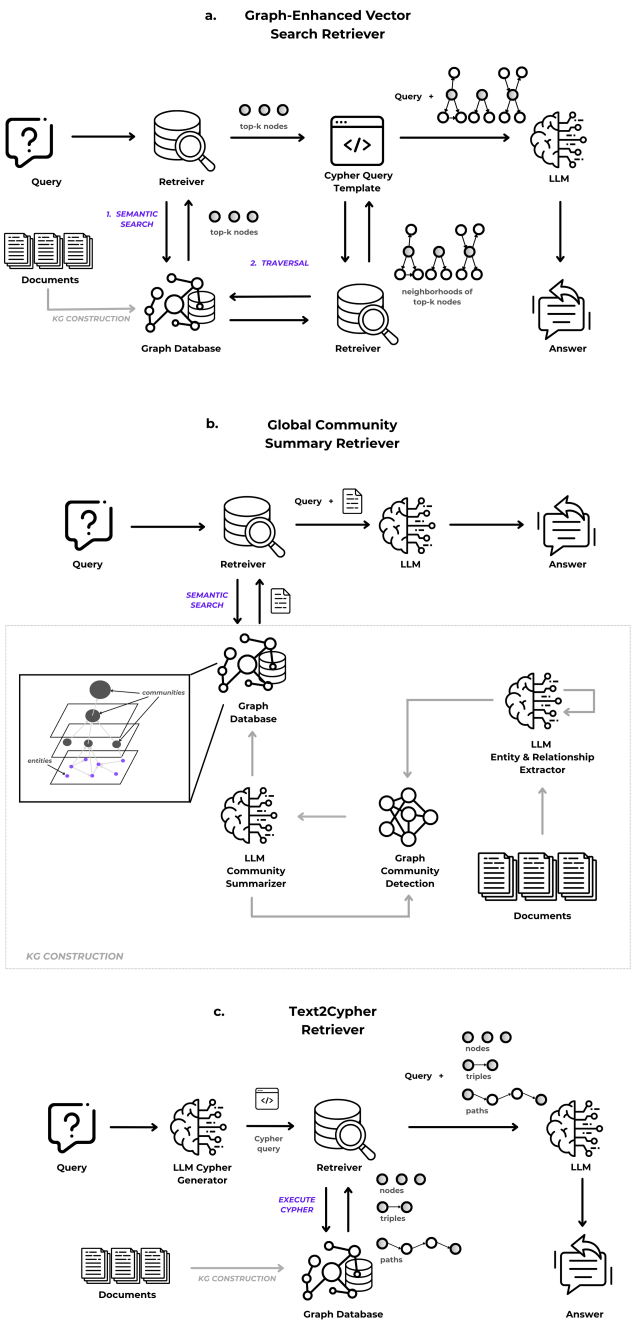


Fig. 1: Retrieval Patterns in KG-enhanced LLMs. (a) Graph-Enhanced Vector Search: This hybrid pipeline retrieves top-k nodes via semantic search, then executes a fixed graph language query template to extract each node's neighborhood. These neighborhoods and original query are passed to the LLM to generate the final answer. (b) Global Community Summary: This architecture constructs a multi-level KG by extracting entities and relations through multiple rounds of LLM calls. A community detection algorithm identifies related node groups, which are also summarized. At inference, semantic search retrieves relevant summaries that, along with the query, are passed to the LLM to generate the final answer. (c) Text2Cypher: An LLM directly generates a Cypher query from the user query, which is executed on the KG. The retrieved subgraph (or individual nodes, triples, and paths) and original query are then passed to another LLM to generate the final response.

[39]). Recent research on the task is presented in more detail in Section 2.5.

Among fully vector-based retrieval, hybrid retrieval, and query generation approaches, this work focuses on the latter, in other words, Text2Cypher. This choice is motivated by the nature of the task at hand (reaction and retrosynthesis planning), which often requires the flexibility to support diverse queries and constraints. The main limitations of the other two approaches are the following:

1) Global community summary pattern might not be able to reliably preserve reaction order or path connectivity, compressing dependencies into a single embedding vector. Additionally, it lacks a mechanism to apply query-time constraints (e.g., filtering by maximum yield).
2) Graph-enhanced vector search relies on a fixed traversal logic, which is not easily adaptable to query-specific constraints or goals.

2.5 Text2Cypher

Text2Cypher still remains an underexplored task, with benchmarks such as Text2Cypher [36] and CypherBench [40] only appearing recently. Overall, state-of-the-art methods for Cypher generation are mostly LLM-based. For example, a study by [36] demonstrates the effectiveness of fine-tuning of both open-source and proprietary LLMs. GraphRAFT [37] also investigates fine-tuning, introducing a novel approach to generate synthetic training data. Here, an LLM instantiates candidate Cypher queries based on the entities extracted from the question. After that, these queries are re-ranked by retrieval accuracy, and the best one is selected as ground truth.

In contrast, another study [38] explores a prompt engineering approach, refining the instructions in the prompt based on error analysis. Prompt2Cypher [39] adopts a more complex approach by decomposing the task into smaller subtasks. It prompts the LLM to first identify relevant nodes and relationships in the schema, and only then to generate the final Cypher query. SyntheT2C [41] introduces Cypher generation with an auto-validation pipeline: with execution- and schema-level validation, and LLM-based semantic and coherence validation. The latter is applied to the query results, verifying whether they contain the expected answer. Another work [42] also incorporates self-correction for Cypher generation, but only for non-valid queries with execution errors. On the contrary, [43] attempts to apply automatic query correction, but at the string level, which may not be robust and applicable for all scenarios. More recent research [44] focuses on fine-grained improvements, such as graph schema pruning to reduce noise in the prompt, which smaller models benefit the most from.

Most existing studies on Text2Cypher do not explicitly investigate prompt engineering, particularly in the context of reaction knowledge graphs. In contrast, this work addresses that gap, recognizing that retrieval is a critical bottleneck for any downstream reasoning step.

2.6 Knowledge Graph-enhanced LLMs in Cheminformatics

KG-enhanced LLMs are not yet widely adopted in the domain of chemistry and, specifically, cheminformatics. Instead, current applications are primarily focused on the task of biomedical question answering, such as identifying gene–drug, drug–drug, or drug–disease interactions [45], [46], [47], [48].

In general, there are only a few studies that employ KG-enhanced LLMs for reaction prediction and retrosynthesis planning. For instance, Cat-KG [49] is designed to recommend multi-step relay catalytic pathways. In particular, the system first retrieves all possible pathways of a certain length to a target product using predefined Cypher query templates. Then, hard-coded rules are applied to filter and rank these pathways based on three scoring components: individual reaction quality (e.g. selectivity, completeness), step compatibility (e.g. catalyst similarity), and overall pathway consistency (e.g. reaction type consistency). Another study [50] also adopts a similar approach, as pathways are first recursively expanded starting from the target molecule. Once again, post-processing begins with rule-based filtering, whereas the remaining candidates are re-ranked by the LLM using Chain-of-Thought [51] (CoT) reasoning.

Overall, the main focus of prior research has been on the post-processing of retrieved results, which means that the retrieval itself remains fixed. This work, on the other hand, investigates whether and to what extent the retrieval process can be made more adaptive. In particular, it evaluates how prompt engineering and self-correction strategies influence an LLM's ability to generate syntactically valid and semantically accurate structured queries. Hence, the aim is to provide empirical insight into interaction between KGs and LLMs in the domain of cheminformatics.

2.7 Chemical Reaction Knowledge Graphs

Individual chemical reactions can naturally be modeled using different graph-based representations, each with its own advantages and limitations. These models include graphs, hypergraphs and bipartite graphs, which can either be undirected or directed. As one would expect, directionality here is crucial to accurately capture reaction pathways and irreversible chemical transformations [52], [53], [54].

By aggregating multiple individual reaction subgraphs, a reaction **knowledge graph** can be formed, which encodes shared reaction components and relationships between them, which might include `"reacts with"`, `"produces"`, `"catalyzes"`, etc. Additionally, KGs capture interconnected pathways constituting the entire reaction network. Nevertheless, there is no known prior research on LLMs leveraging such molecular reaction graphs for advanced reaction planning.

2.7.1 Directed Unweighted Graph

One of the simplest structures to model chemical reactions is a directed graph, as illustrated in Fig. 2b. Here, nodes are typically individual molecules, such as reactants and products, whereas edges represent functional relationships between them. Directed graphs, however, do not inherently

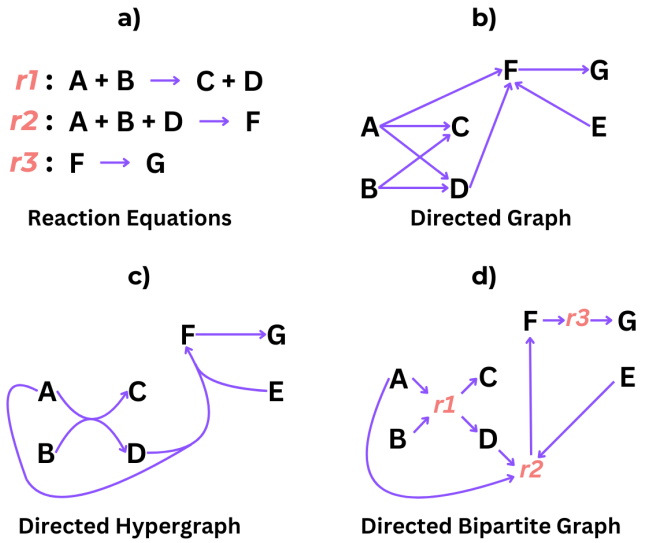


Fig. 2: Different graph representations for chemical reactions. (a) Reaction equations (b) Directed graph representation, where nodes are molecules and edges represent reactions. (c) Directed hypergraph that captures reactions with multiple reactants and products. (d) Directed bipartite graph, where nodes correspond to reaction and molecules.

capture which molecules react together to produce other chemicals [54].

When constructing a reaction knowledge graph, another challenge for this representation would be to incorporate agents (such as catalysts) and solvents. Since all participating molecules must be connected to or through them, this introduces unnecessary redundancy, making the graph more cluttered. Once again, the same issue arises, as it remains unclear how molecules interact in a given reaction.

2.7.2 *Hypergraph*

Reactions can also be represented by a hypergraph framework. This includes the polyadic relationship, in which multiple reactants together can form multiple products. In other words, simple graphs fail to accurately model more complex chemical systems. To address this limitation, chemical reactions can be represented by hypergraphs instead, as depicted in Fig.2c. Specifically, hypegraphs consist of a set of nodes and hyperedges, each of which is defined as a multiset of interacting nodes. This approach provides a more accurate representation of the interactions between molecules compared to simple graphs [55].

The main drawback of this model in terms of scaling to reaction KGs is that most databases, such `Neo4j`, do not natively support hypegraphs. Furthermore, most widely used graph traversal and path-finding algorithms (BFS, Dijkstra, etc) do not generalize to this representation. In particular, the whole definition of a "path" here is ambiguous, since a single hyperedge connects multiple nodes. Finally, research on the use of hypergraphs for knowledge encoding still remains fairly limited.

2.7.3 *Bipartite Graph*

Bipartite graphs are an alternative representation of hypergraphs that preserve the reaction context. A chemical bipartite graph contains two types of nodes: reactions and molecules, as shown in Fig. 2d. Here, each reaction node is connected to all its corresponding reactants and products, which allows to define multi-component interactions explicitly [54], [56]. Hence, additional molecules involved in the reaction, such as agents or solvents, can also be easily incorporated.

Overall, molecules participating in multiple reactions will inherently connect those reactions as shared graph nodes. As a result, this facilitates the discovery of multi-step synthesis pathways within a datasets initially composed of individual, unconnected reactions.

Consequently, the bipartite KG ensures both accurate reaction representation with full reaction context and efficient querying. This is due to its compatibility with graph databases and the applicability of well-defined traversal and search algorithms.

3 Methodology

This section describes the experimental setup of this research study, including pre-processing and cleaning of the dataset used. In addition, it covers the construction of the molecular knowledge graph, highlighting key design considerations. The section also discusses strategies to improve Cypher query generation, which remains a major bottleneck for the entire pipeline. The, the experimental pipeline is described. Finally, the evaluation framework used to access the setup in the task of reaction prediction and planning is outlined.

3.1 Dataset

This research study uses USPTO dataset, which contains chemical reactions from US patents [4]. Data preprocessing involved removing duplicates, canonicalizing molecular SMILES strings, and filtering out rare reactions and molecules. Furthermore, only reactions with at most four reactants, four products, four agents, and four solvents were retained, accounting for 95% of the entire dataset.

3.2 Knowledge Graph Construction

Each individual reaction is represented as a bipartite graph with two distinct node types:

- **(:Reaction)** — identified by a unique `id`.
- **(:Molecule)** — uniquely identified by a canonicalized SMILES `name`.

The full reaction schema is shown in Fig. 3. This design treats reaction nodes solely as structural entities, linking all molecule nodes. Therefore, neither the reaction SMILES strings nor natural language descriptions were added as reaction node properties. During retrieval, these text-based descriptors would introduce unnecessary redundancy, since the relevant information is already embedded in the graph topology.

Moreover, the implemented knowledge graph structure abstracts away the complexity of reaction SMILES strings, which LLMs might potentially struggle to interpret [6], [57].

That is, instead of returning reactions in a linear SMILES format, for instance, `CCO.CC(=O)O >> CC(=O)OCC`, the graph provides a structured representation with already explicit role separation, such as: { `reactants: [CCO, CC(=O)O], products: [CC(=O)OCC]` }.

Reaction Graph Schema

Node Properties

```
(:Reaction {id: INTEGER})
(:Molecule {name: STRING})
```

Relationship Properties

```
(:Reaction)-[:USES_SOLVENT]->(:Molecule)
(:Reaction)-[:USES_AGENT]->(:Molecule)
(:Reaction)-[:PRODUCES {yield:
FLOAT}]->(:Molecule)
(:Molecule)-[:REACTS_IN]->(:Reaction)
```

Fig. 3: Schema of the Reaction Knowledge Graph.

In total, 50k reactions were added to the `Neo4j` database, as querying the full dataset proved too inefficient. Since this work primarily focuses on retrieval quality itself, the latency reduction allowed for faster and more iterative experimentation.

3.3 Test Data Generation

3.3.1 Single-step Reaction Tasks

The test dataset for single-step reaction tasks consists of 1,200 queries, 200 per task type, as summarized in Table 1. These tasks revolve around different aspects of retrosynthesis and aim to evaluate the abililty of the LLM to generate valid Cypher of varying syntactical complexity.

For each task, a template of the ground ground truth Cypher queries was created: classic Cypher, with sequantial `MATCH/OPTIONAL MATCH` clauses followed by `RETURN` statements with aggregated collections (using `COLLECT`).

Task	Query	Samples
Precursor Identification	What are the direct precursors used to synthesize $\{X\}$?	200
Product-to-Reaction Mapping	What are all reactions that directly produce $\{X\}$?	200
Best-Yielding Reaction	What is the best-yielding reaction that produces $\{X\}$?	200
Multi-Product Reaction	What reactions produce all of $\{X_i, X_{i+1}, \ldots, X_n\}$ in a single step?	150 (n=2), 45 (n=3), and 5 (n=4)
Co-product Identification	What are the co-products of $\{X\}$ in the reactions directly synthesizing it?	200
Condition-Based: Agents & Solvents	What agents/solvents are used in the synthesis of $\{X\}$?	100 (agents), 100 (solvents)

TABLE 1: Retrosynthesis task types for single-step reaction retrieval, along with natural language queries and number of test samples per task.

In order to generate instances for the *Precursor Identification*, *Product-to-Reaction Mapping*, and *Condition-Based: Agents& Solvents* tasks, product molecules are randomly sampled from the KG. These sampled molecules are then inserted into both the natural language question templates and Cypher query templates. All of these query types are designed to assess whether the LLM is able to retrieve full reaction context needed for the downstream task.

The previously described tasks all share the same Cypher template, whereas the following ones require both additional modifications to it and a more constrained data sampling strategy. In case of *Best-Yielding Reaction* task, the product molecules were sampled such that each appears in at least two reactions, with at least one having a non-null yield value. Similarly, for *Multi-Product Reaction*, product sets were sampled to contain the exact number of required molecules. Here, the goal is to evaluate whether the LLM can apply property-based filtering logic. Molecules for the *Co-Product Identification* task, were sampled to ensure that 80% of queries correspond to true multi-product reactions, while the remaining 20% are negative cases of single-product reactions. This approach allows to assess the robustness of grouping/aggregation logic.

3.3.2 Multi-step Reaction Tasks

The test dataset for multi-step reactions includes four task types, with 300 queries per type, 1200 in total, as detailed in Table 2. The product molecules and, similarly, the precursor molecules, were randomly sampled, making sure that each of them belonged to reaction pathways of desired length $n \in \{2, 3, 4\}$. The constraint was necessary, as a search for all pathways of length $\leq n$ can explode combinatorially, resulting in slow queries. This study, is not intended to benchmark database efficiency, but aims to evaluate the capability of the LLM to generate multi-hop Cypher queries.

3.4 KG-enhanced LLM

As illustrated in Fig.4, two types of prompting strategies were evaluated. The first, direct prompting (Fig. 4a), includes both zero-shot and one-shot approaches. Under one-shot, three distinct example selection techniques were tested: static, dynamic random, and dynamic semantic. In this strategy,

Task	Query	Samples
Reaction Chains to Target	What are the possible multi-step reaction chains that can lead to the synthesis of $\{X\}$ in exactly $\{n\}$ reaction steps?	300
Reaction Chains Between Molecules	What are the possible reaction pathways that synthesize $\{B\}$ starting from $\{A\}$ in exactly $\{n\}$ steps?	300
Intermediate Molecule Identification	What are the intermediate molecules involved in synthesizing $\{X\}$ in exactly $\{n\}$ reaction steps? $\{X\}$?	300
Multi-step Precursor Identification	What are the precursors that can produce $\{X\}$ in exactly $\{n\}$ steps?	300

TABLE 2: Retrosynthesis task types for multi-step reaction retrieval ($n \in \{2, 3, 4\}$), 300 total test samples (100 per n), with natural language queries and per-task counts.

an LLM generates a Cypher query that is then executed on the reaction KG. The objective for single-step synthesis is to retrieve the full reaction context, whereas for multi-step synthesis, it is to retrieve ordered sequences of relevant reaction nodes.

To investigate whether an LLM can self-correct errorsthat arise during Cypher generation, a CoVe-style prompting pipeline was tested. After the LLM generates a candidate query, its excitability is checked with `EXPLAIN` in `Neo4j`. If the query is not executable, an LLM corrector is called to fix it based on the error message. If it is executable, a deterministic directionality corrector is applied. 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 to the LLM corrector to apply minimal edits based on the identified issues. The process repeats until either the validator labels the query as correct or a limit of three correction attempts is reached.

3.5 Cypher Generation: Zero-Shot Setting

3.5.1 Single-step Synthesis Prompts

In total, 5 prompts were tested during the prompt engineering experiments (see Prompts P1)- P5), where each prompt was designed following OpenAI's guidelines on prompt engineering [58]. Here, prompt engineering started with a baseline, minimal prompt. Based on subsequent error analysis, additional instructions and constraints were incrementally introduced to improve the model's output further.

The user prompt only contained the input question, whereas the system prompt defined the model's role as a helpful assistant for generating Cypher queries from natural language, for use with `Neo4j`. Additionally, it includes relevant instructions with varying levels of complexity, such as schema and formatting rules, dataset-specific nuances, and guidance on the retrieval of full reaction contexts. The latter refers to retrieving all relevant reaction components, including not just the target product entities, but also precursors, agents, solvents, even when they are not explicitly mentioned in the question. This ensures that any downstream task has the complete information needed to answer the question.

3.5.2 Multi-Step Synthesis Prompts

Prompt engineering (see Prompts P6)- P10) for multi-step synthesis followed the same approach, starting from a minimal prompt that contains only the task description, schema, and KG-specific assumptions. Then, additional instructions were added and evaluated incrementally. Here, however, the required output is no longer the full reaction context, but the relevant reaction nodes (see Sec.4.3 for the motivation).

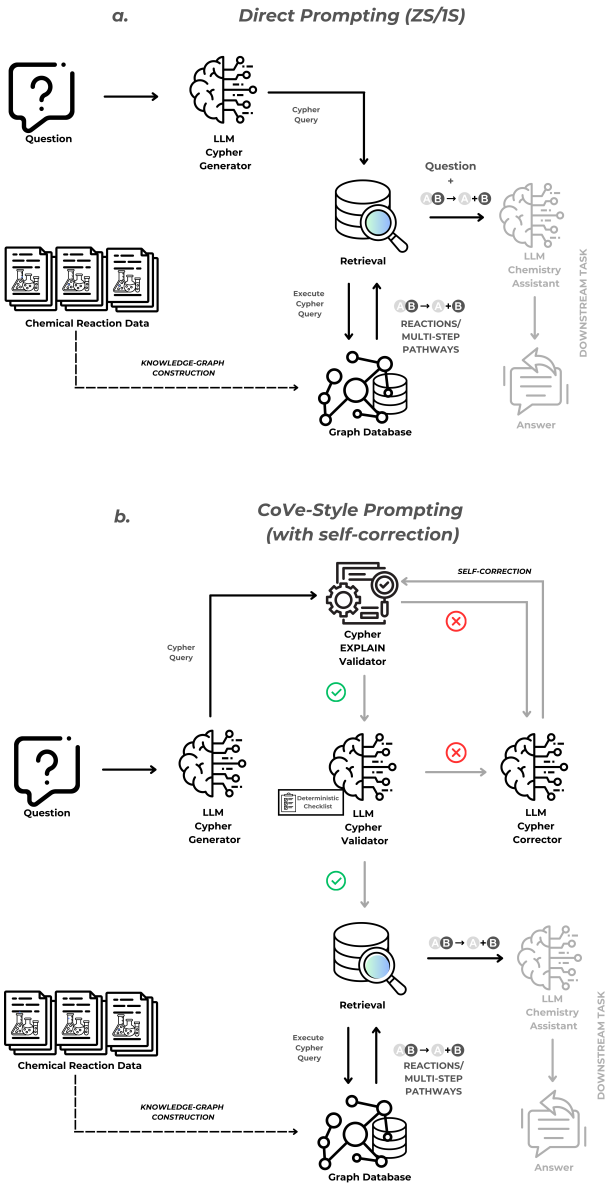


Fig. 4: (a) Direct Prompting (ZS/1S) Approach: Based on the user question, an LLM generates a Cypher query query, with the goal of retrieving relevant reactions or multi-step pathways. The query is then executed on the reaction knowledge graph in Neo4j. (b) CoVe-style Prompting Approach: First, an LLM generates a candidate query. A validator LLM checks it against a fixed, task-specific checklist derived from prior error analysis, 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 validator classifies the query as valid or a maximum of 3 correction attempts have been reached.

3.6 Cypher Generation: Few-Shot Setting

In one-shot prompting strategies, the system prompt remained unchanged, containing the same guidelines as in a zero-shot setting. The user prompt, however, also included example pairs of natural language questions and the corresponding Cypher queries. The following one-shot strategies were explored: static one-shot prompting, dynamic one-shot prompting with random example selection, and dynamic one shot-prompting with embedding-based example selection.

3.6.1 Example Bank for Single-Step Reactions

The example bank 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. These structurally related but distinct tasks are summarized in Table 3. Note, that SMILES strings were intentionally excluded from the examples to not influence Cypher generation by chemical context.

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.
Reactant-to-Reaction Mapping	Which one-step reactions involve molecule X as a reactant?	Find all reactions where the given molecule participates.
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-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.
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.

TABLE 3: Example Bank for Single-Step Retrieval: forward synthesis single-step planning tasks, together with their natural language queries and corresponding logical intent

3.6.2 Example Bank for Multi-Step Reactions

The multi-step reaction example bank mostly consists of examples focused on forward synthesis, except for *Reaction Chains Between Molecules*. Here, the natural language question are rephrased relative to the test set, and SMILES strings are replaced with neutral placeholders. The path length is explicitly set to 5, as the prompt only provides step-to-hop mappings for 2–4 steps. Additionally, Cypher does not allow variables in traversal lengths (i.e., `-[:REACTS_IN|PRODUCES*..n]->` is invalid, whereas `-[:REACTS_IN|PRODUCES*..10]->` is correct), and thus specifying the bound is necessary to prevent potential errors.

3.6.3 Static One-Shot Prompting

In a one-shot static prompting strategy, a single representative example was selected for all tasks. In particular, this example corresponds to the most common query pattern, *Product Identification* for the single-step reaction dataset, as its template is shared across three out of six query types. The goal of the demonstration was to emphasize contextual retrieval, guiding the model to return full reaction context. For the multi-step synthesis dataset, the one-shot examples tested were *Multi-Step Product Discovery* and *Forward Synthesis Intermediate Identification*. The rationale behind this choice is analogous, as the prompt explicitly instructs the model to return only reaction nodes, even though it is not implied in the question itself.

Task	Query	Intent
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.
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.
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.
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.

TABLE 4: Example Bank for Multi-Step Retrieval: multi-step planning tasks, together with their natural language queries and corresponding logical intent.

3.6.4 Dynamic One-Shot: Random Selection

This strategy randomly selects an example from the predefined example bank to use as a one-shot demonstration. It is needed to establish a baseline for the embedding-based example selection strategy. Since the chosen example may not be semantically or structurally aligned with a given input question, the setting allows to evaluate how much example quality influences Cypher generation.

3.6.5 Dynamic One-Shot: Embedding-Based Selection

In dynamic embedding-based selection, examples are stored in a vector database and retrieved based on vector similarity. Since the example tasks are in the opposite direction, direct question-to-question comparison does not guarantee structurally similar queries to be retrieved in this case. Therefore, logical intent descriptions were added to all demonstrations. This method prioritizes alignment on the underlying task structure over surface-level semantic similarity. A general-purpose sentence encoder was used for this project to evaluate whether effective matching can be achieved without task- and domain-specific models.

3.7 Validator & Corrector Prompts

The validator prompts (Prompts P11, P12) contain a checklist of the most frequent error types derived directly from prior error analysis for single- and multi-step tasks, respectively. The corrector prompts (Prompts P13, P14) are mostly validator-guided, as they make only the tasked to introduce the smallest edits possible to address the identified issues. Both of these prompts, however, restate the task intent—retrieve the full reaction context for single-step

queries and enforce a reaction-nodes-only output for multi-step queries.

3.8 Evaluation: Retrieved Results for Single-Step Retrieval

3.8.1 Single-step Retrieval Results

The retrieved results are represented as lists of dictionaries with keys and their values, with an example of the gold answer shown in Fig. 5a. Here, the values are usually lists of SMILES strings, except for `yield` and `reaction_id` keys. However, this exact structure is not guaranteed, due to inherent variability in Cypher queries generated by the LLM.

The generated results are evaluated by computing precision, recall, and F1 scores. These metrics are then micro-averaged per data sample across all relevant keys found in ground truth, e.g., `"reactants"`, `"products"`, etc. Such approach allows global performance to be evaluated effectively as it captures how the system behaves on individual instances, i.e., on a value basis. That is of particular importance in synthesis planning tasks, where incomplete information retrieval might negatively affect the overall quality of the final outcome.

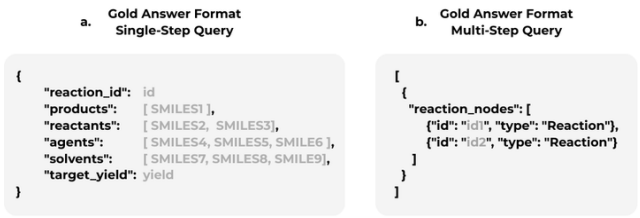


Fig. 5: Format of gold answers for a. single-step and b. multi-step retrieval tasks.

For a given key, true positives (TP), false positives (FP), and false negatives (FN) can be defined using set operations:

$$\begin{aligned} \text{TP} &= |y_{\text{true}} \cap y_{\text{pred}}| \\ \text{FP} &= |y_{\text{pred}} \setminus y_{\text{true}}| \\ \text{FN} &= |y_{\text{true}} \setminus y_{\text{pred}}| \end{aligned} \tag{2}$$

where y_{true} represents the set of ground-truth values and y_{pred} is the set of predicted values for a given key. From these, the metrics are computed:

$$\begin{aligned} \text{Precision} &= \frac{\text{TP}}{\text{TP} + \text{FP}} \\ \text{Recall} &= \frac{\text{TP}}{\text{TP} + \text{FN}} \\ \text{F1} &= 2 \cdot \frac{\text{Precision} \cdot \text{Recall}}{\text{Precision} + \text{Recall}} \end{aligned} \tag{3}$$

For each sample, TP, FP, FN are aggregated across all keys k:

$$\begin{aligned} \text{Micro-Precision}_{\text{sample}} &= \frac{\sum_k \text{TP}_k}{\sum_k \text{TP}_k + \sum_k \text{FP}_k} \\ \text{Micro-Recall}_{\text{sample}} &= \frac{\sum_k \text{TP}_k}{\sum_k \text{TP}_k + \sum_k \text{FN}_k} \\ \text{Micro-F1}_{\text{sample}} &= 2 \cdot \frac{\text{Micro-P}_{\text{sample}} \cdot \text{Micro-R}_{\text{sample}}}{\text{Micro-P}_{\text{sample}} + \text{Micro-R}_{\text{sample}}} \end{aligned} \tag{4}$$

3.8.2 Key-Maching Procedure

Since both the structure of the dictionaries and the exact key names can vary between the generated and ground truth queries, the key matching procedure is required. Before that, each key is normalized by lowercasing, trimming whitespace, converting camelCase to snake_case, and removing a trailing "name"/"names". The matching process involves 3 stages, illustrated in Fig. 6:

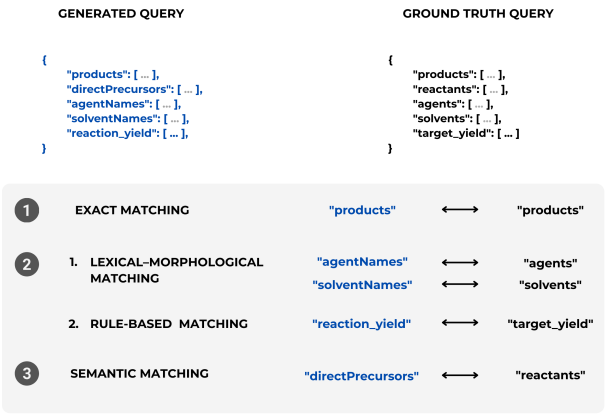


Fig. 6: Key-matching pipeline (generated query to ground truth): exact → lexical → semantic.

1) **Exact Matching:** Normalized predicted keys are first matched directly to ground truth keys based on string equality.
2) **Rule-based Matching and Lexical–morphological matching:**
   - If the predicted key contains "yield", it is matched to the corresponding ground-truth key if applicable.
   - Stemming is performed to reduce keys to their root form (e.g. agents → agent).
3) **Embedding-Based Similarity Matching:** Remaining unmatched predicted keys are compared to ground truth keys using embedding similarity. Each prediction is assigned to the highest-scoring ground-truth key if the score $\geq$ threshold.

Notably, multiple predicted keys may map to a single ground-truth key; in such cases, their values are aggregated during evaluation.

3.9 Evaluation: Retrieved Results for Multi-Step Retrieval

The gold answer, as illustrated in Fig.5b, is a list of distinct reaction paths, each represented by the ordered sequence of reaction nodes along the path. Two paths are considered identical iff their ordered id sequences are the same. As a result, this eliminates the need for a key matching procedure, as a result simplifying the evaluation.

Exact-path retrieval is evaluated using precision, recall, and F_1, as defined in Eq. 3, where each *item* is an *ordered* tuple of reaction-node ids. These metrics are intentionally strict, and thus truncated paths, which are partially correct, are assigned a score of 0.

Hence, partial path recall was introduced to provide graded credit when a retrieved path matches a continuous terminal fragment (either a prefix or a suffix) of a ground-truth path:

$$\begin{aligned}\mathrm{tcf}(p,g) &= \max\{\, k :\ p_{1:k} = g_{1:k} \text{ or } \\ &\qquad p_{|p|-k+1:|p|} = g_{|g|-k+1:|g|} \,\} \\ \mathrm{PPR}(P,G) &= \frac{1}{|P|}\sum_{p\in P}\max_{g\in G}\frac{\mathrm{tcf}(p,g)}{|g|}\end{aligned} \tag{5}$$

where P and G are the sets of predicted and ground-truth paths respectively, p and g represent ordered tuples of reaction-node ids, with $|p|$ and $|g|$ corresponding to their lengths, $p_{1:k}$ is the prefix of p of length k (resp. $g_{1:k}$), $p_{|p|-k+1:|p|}$ is the suffix of p of length k (resp. $g_{|g|-k+1:|g|}$), and tcf denotes the length of the longest terminal common fragment.

3.10 Evaluation: Generated Cypher

The generated Cypher queries were compared to ground truth queries using BLEU, METEOR, and ROUGE-L scores. Since Text2Cypher is a text-to-text generation task, these metrics are appropriate for evaluating their lexical and structural similarity [36]. Prior to that, a custom regex-based tokenization step was applied to Cypher queries. It preserves quoted strings (e.g., SMILES) as single tokens and splits the rest of the query into keywords (e.g. `MATCH`, `RETURN`), symbols (e.g. {}, ->), and identifiers (e.g. `Molecule`, `PRODUCES`).

3.10.1 BLEU (Bilingual Evaluation Understudy)

BLEU measures a geometric mean of n-gram precision between the generated sequence and the reference, which is adjusted by a brevity penalty:

$$\mathrm{BLEU} = \mathrm{BP}\cdot e^{\left(\sum_{n=1}^{N} w_n \log p_n\right)} \tag{6}$$

$$\mathrm{BP} = \begin{cases} 1 & \text{if } c > r \\ e^{\left(1-\frac{r}{c}\right)} & \text{if } c \le r \end{cases} \tag{7}$$

where BP stands for brevity penalty, p_n corresponds to the modified n-gram precision, w_n represents the weight for each n-gram (commonly $\frac{1}{N}$), c denotes the length of the candidate senquence, and r is the length of the reference respectively [59], [60].

3.10.2 METEOR (Metric for Evaluation of Translation with Explicit ORdering)

METEOR is based on the harmonic mean of unigram (single custom token) precision and recall, weighting recall higher than precision. Additionally, the metric supports semantic matching due to the incorporation of stemming and synonym mapping. This makes it a more flexible score compared to BLEU, which only relies on exact n-gram matches.

$$\mathrm{METEOR} = F_{\mathrm{mean}}\cdot(1-\mathrm{penalty}) \tag{8}$$

$$F_{\mathrm{mean}} = \frac{10\cdot P\cdot R}{R+9P} \tag{9}$$

$$\mathrm{penalty} = \frac{1}{2}\cdot\left(\frac{\#\mathrm{chunks}}{\#\mathrm{unigram\ matches}}\right)^{3} \tag{10}$$

where P and R denote unigram precision and recall, respectively, #unigram matches is the total number of matched unigrams, whereas #chunks correponds to the number of contiguous matched sequences [61].

3.10.3 ROUGE-L (Recall-Oriented Understudy for Gisting Evaluation)

ROUGE-L measures the longest common subsequence (LCS) between the generated and reference text. It is important to mention that the tokens in the LCS do not need to be contiguous as long as they are in order.

$$\text{ROUGE-L F1} = F_{\mathrm{LCS}} = \frac{P_{\mathrm{LCS}}\cdot R_{\mathrm{LCS}}}{R_{\mathrm{LCS}}+P_{\mathrm{LCS}}} \tag{11}$$

$$P_{\mathrm{LCS}} = \frac{\mathrm{LCS}(X,Y)}{\mathrm{len}(X)},\quad R_{\mathrm{LCS}} = \frac{\mathrm{LCS}(X,Y)}{\mathrm{len}(Y)} \tag{12}$$

where $\mathrm{LCS}(X,Y)$ denotes the length of the longest common subsequence between candidate sentence X and reference sentence Y, P_{LCS} and R_{LCS} are the LCS-based precision and recall, β represents a weighting factor to balance precision and recall [62].

3.11 Implementation Details

3.11.1 Data Pre-processing

In order to improve data quality and reduce inconsistencies, the data was pre-processed using the ORDerly[1] library.

3.11.2 LLM

The LLM used is `GPT-4.1-mini-2025-04-14` [63], accessed via the OpenAI API. The temperature parameter was fixed at $T = 0$ to ensure deterministic outputs.

3.11.3 Demonstration Selection

For semantic demonstration selection, the Sentence Transformer model `all-mpnet-base-v2` [64]. This is consistent with the overall premise of this study to rely on out-of-the box models. The encoder is available through `Hugging Face`[2].

3.11.4 Key Matching Procedure

To normalize lexical variants, Porter Stemmer [65] was applied, as it is lightweight and the keys typically differ only in their endings (singular vs plural). For semantic matching, `BiomedNLP-BiomedBERT` [66] (also available via `Hugging Face`) was used, due to its pretraining on PubMed data. The model is expected to more reliably distinguish reaction-related terms, particularly in low-context scenarios. A cosine-similarity threshold of 0.93 was enforced, and the matched keys with similarity < 0.93 were rejected.

3.11.5 Workflow Orchestration and LLM Integration

The system is implemented using `LangChain` for LLM integration and prompt management, and `LangGraph` for workflow orchestration, which enables conditional branching and error handling.

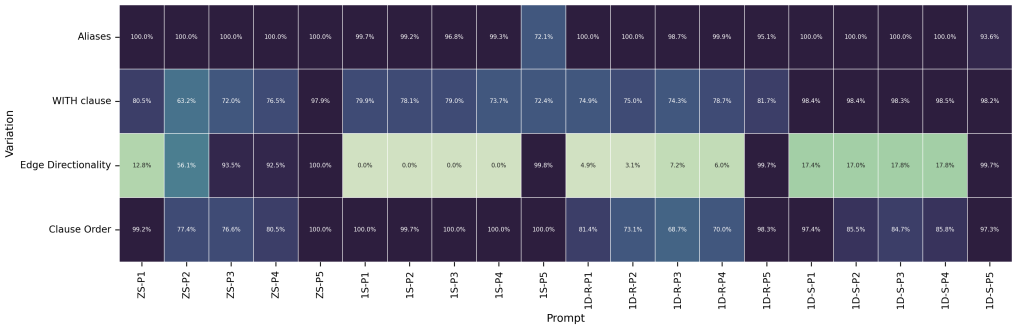


Fig. 7: Frequency of semantically neutral Cypher query variations among queries achieving perfect retrieval (F1 = 1.0), across four prompting strategies (zero-shot (ZS), one-shot static (1S-S), one-shot random (1S-D-R), and one-shot semantic (1S-D-S)) and five prompt versions. The prompt versions increase in contextual and structural guidance: (1) – base contextual query, (2) – (1) + explicit MATCH instructions, (3) – (2) + directionality check, (4) – (3) + yield-filtering, and (5) – (4) + full context rules. Each cell shows the percentage of F1=1 queries that differ from the ground truth in one of four variation types: alias usage, clause ordering, edge direction, and use of WITH clauses. These variations do not impact query logic, but reveal stylistic and structural differences introduced by each prompt.

3.12 Code Availability

All code needed to reproduce the experiments is available at: `github.com/olgabunkova/GReacRAG`.

4 Results & Discussion

4.1 Single-step Synthesis: Cypher Queries

In the zero-shot setting, prompt engineering helps improve string-overlap scores or syntactic similarity between generated and target queries, but up to a point, as shown in Fig. 8. Specifically, it occurs in a task-dependent manner: median values per metric increase and plateau for the *Best-Yielding Reaction*, *Co-Product Identification*, and *Condition-based: Agents&Solvents* tasks, whereas *Precursor Identification* retains similar score distributions. Performance, however, consistently degrades with Prompt 5, which contains verbose guidelines and full contextual rules for Cypher generation. These instructions likely over-constrain the model, enforcing rigid outputs and thus limiting helpful variation. A similar inverted U-shaped trend has been observed in Text2SQL tasks [67], suggesting the finding might be part of a broader pattern of structured generation.

Notably, *Multi-Product Reaction* is among of the worst-performing tasks across all prompt variants in the zero-shot setting, likely due to its more complex filtering logic. *Product-to-Reaction*, on the other hand, appears to have the best performance for all three evaluation metrics. This may be due to the provided instructions aligning most closely with the task's intent.

In contrast, one-shot prompting, generally leads to better Cypher generation performance, obtaining higher medians and tighter metric distributions on certain tasks. Interestingly, the LLM is able to recover from the overly complex prompt version 5, e.g., in case of *Precursor Identification*. This suggests that the LLM relies more on structural cues from the demonstration itself, rather than the provided instructions. Among one-shot strategies, evaluation metrics of the random selection strategy demonstrate on average wider score distributions with lower medians. Static selection yields tight but more lower-centered distributions compared to embedding-based selection on tasks such as *Multi-Product Reaction* task and *Co-Product Identification*. A likely explanation is that a fixed example biases the model towards a specific surface-level structure, e.g., a certain clause ordering, which is not identical across all the demonstrations.

This highlights the importance of demonstration quality and structural alignment in Cypher generation. Improvements appear to be the largest when a demonstration is structurally more similar to the target query, especially evident in the *Best-Yielding Reaction* task under embedding-based selection. This is likely because the selected example shares otherwise an identical template, requiring only a single *reactant/product* condition to be flipped. The observed effect is examined in more detail in the retrieval-performance analysis. A similar pattern was reported in a Text2SPARQL study [68], in which embedding-based few-shot example selection significantly improved performance compared to static few-shot selection. Once again, this sensitivity to demonstration quality might be part of a more general trend. Lastly, none of the queries achieved a perfect score on any of the metrics, even with a perfect retrieval performance of $F1 = 1$. This may stem from syntactic variations that do not necessarily affect the semantics or logical correctness of the query. Such variations are summarized in Table 5, whereas their occurrences in generated Cypher can be found in Fig. 7. Given that at least one such variation appears in every query, the chosen metrics might not be most optimal for comparison. As mentioned earlier, this is because BLEU, METEOR, and ROUGE-L F1 only capture surface-level alignment.

4.2 Single-step Synthesis: Retrieval Performance

Retrieval-based evaluation provides a more meaningful assessment of prompt performance, as multiple syntactically different Cypher formulations can yield identical results. The performance of different prompting strategies at the retrieval level is thus summarized in Figure 9. An error analysis was conducted to identify the most common issues impacting the

[1]https://github.com/sustainable-processes/ORDerly
[2]https://huggingface.co/

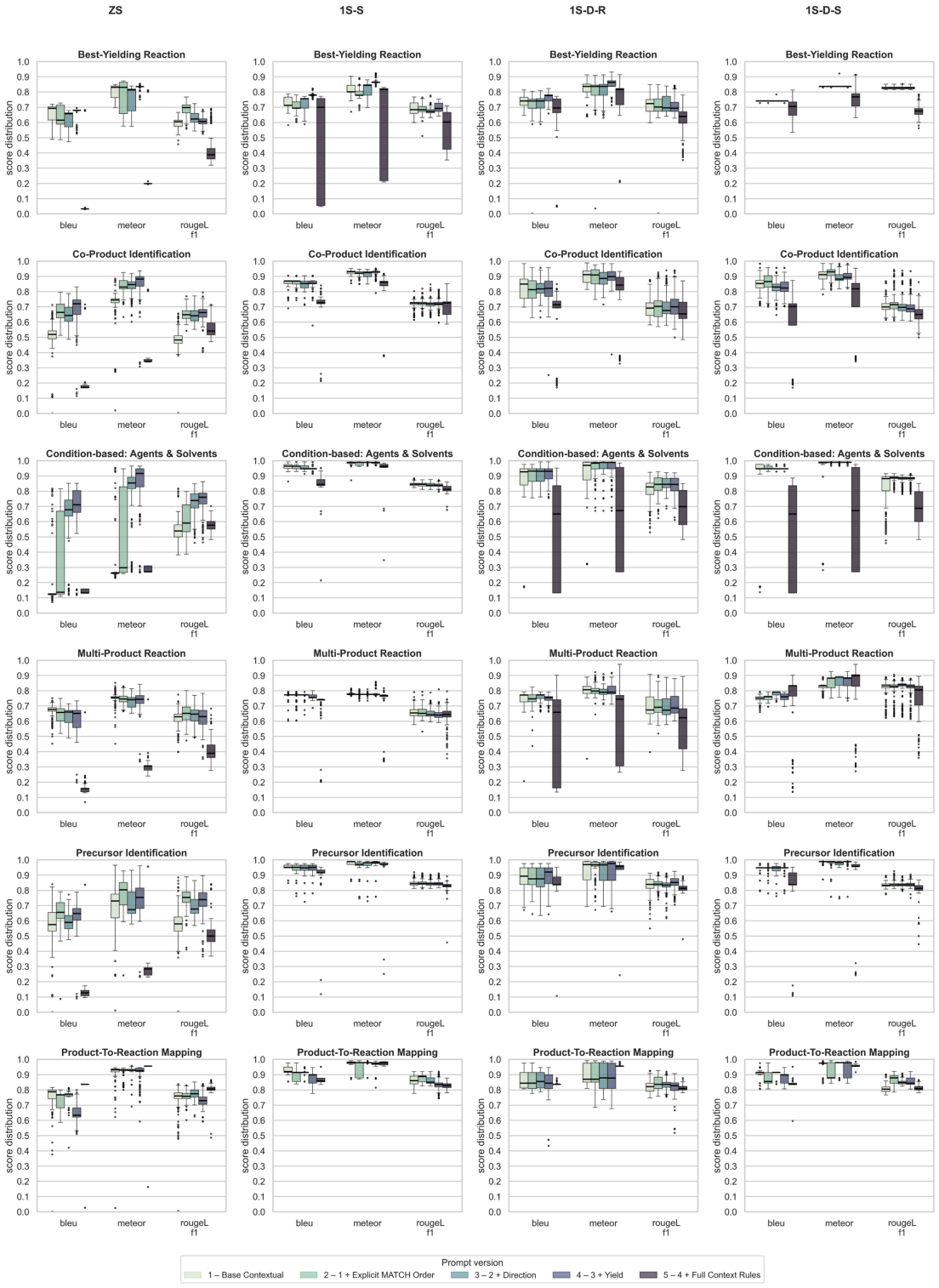


Fig. 8: Comparison of generated Cypher queries to ground truth queries for Single-step Synthesis. Each subplot shows BLEU, METEOR, and ROUGE-L F1 score distributions for a specific query type under different prompt settings: zero-Shot, one-shot static, one-shot random, and one-shot semantic. Within each setting, five prompt versions (color-coded) reflect increasing levels of contextual and structural guidance.

performance of the system. These findings are summarized in Fig. 11, with Table 6 providing the definitions and causes for each identified error type.

4.2.1 *Zero-shot Setting*

Consistent with Cypher query trends, zero-shot prompting performance improves with increasing context until Prompt 5, where performance in recall (and thus F1) drops again. Here, the LLM loses track of the task and relevant instruc-

Fig. 9: Comparison of 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.

tions, no longer retrieving the full reaction context. As shown in Fig. 11, this pattern is reflected in the most frequent errors: missing reactants (574 out of 1200 test queries), missing agents 759), missing solvents (958), and missing products (764).

In contrast, Prompt 1 with minimal contextual guidelines produced the highest rate of invalid queries (9.74%), resulting in the largest number of empty results. Adding the explicit instruction in Prompt 2 – first use `MATCH` to bind nodes and relationships, then aggregate with `collect` – reduced this

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
WITH + RETURN COLLECT	`WITH r, COLLECT(...) AS x RETURN x` vs `RETURN COLLECT(...)`	No

TABLE 5: Common syntactic variations in Cypher queries that typically do not affect the semantics or logical correctness of the query, but may be penalized by surface-level or structural metrics.

error rate drastically to 0.58%. The majority of errors under Prompt 1 can be attributed to two main factors: 1) the use of a less common Cypher pattern comprehension syntax without `MATCH` clauses (28.28% out of all invalid queries), which the LLM was likely less exposed to during training, and 2) from variable binding mistakes (73.9%), which could be mitigated by following a structured query generation process.

4.2.2 Directionality Errors in the Zero-shot Setting

Prompt 3, which guides the model on correct relationship directionality, achieved the best performance cross the *Co-Product Identification*, *Condition-Based: Agents & Solvents*, and *Precursor Identification* tasks (Fig. 11). In particular, it yielded statistically significant improvements in F1 on the Wilcoxon signed-rank test (Fig. S4, S5, S7). These results indicate that the LLM struggles primarily with enforcing correct relationship directionality, specifically and exclusively for reactants, even when the graph schema is provided as context.

The trend could be likely due to the fact that `-[REACTS_IN]->` belongs to a non-dominant directional pattern in the graph schema, pointing from molecule node to reaction node, and not vice versa. In other words, the LLM might be biased towards the majority label, a phenomenon previously observed in demonstration selection ([69], [70]), which might potentially be applicable here. To investigate this further, Cypher generation experiments were performed using "fake" schemas combined with Prompt 2 that suffered from this error the most. When all relationships from molecule to reaction nodes used the same `*_IN` naming (`REACTS_IN`, `AGENT_IN`, `SOLVENT_IN`), directionality errors disappeared. Here, the only minority label `(:Reaction)-[:PRODUCES]->(:Molecule)` was handled correctly in all cases. Nevertheless, replacing `REACTS_IN` with `USES_REACTANT` and renaming `USES_AGENT` as `AGENT_IN` caused directionality errors to reemerge again, though at a lower rate (258/1200 cases). This indicates that the LLM is sensitive to the lexical wording of relation names, the semantic roles of entities (reactant vs agent), and the degree of consistency within in-going and out-going relations with respect to each node. In other words, it goes beyond majority-label bias: schema design appears to influence error patterns in the zero-shot setting, which should be investigated more systematically in future work.

Error Type	Description	Interpretation
Empty Retrieval	No results retrieved despite a valid query being generated	Query is invalid, or incorrect semantically or filtering by yield is incorrectly applied
Incorrect SMILES entities	SMILES strings from the input question are altered or malformed (e.g., `[Cl]` → `Cl`)	LLM alters chemical tokens, failing to preserve exact SMILES notation required for retrieval
Wrong Reactant Directionality	`(R)-[:REACTS_IN]->(M)` instead of `(M)-[:REACTS_IN]->(R)` `M` = molecule node, `R` = reaction node.	LLM flips the `REACTS_IN` relationship likely due to majority edge-direction bias
Null Yield	Yield property missing or set to null	LLM fails to apply yield≠null filtering correctly
Duplicate Data	Near-identical reactions returned, differing only in agents or solvents	Inevitable data-level artifact, not resolved by pre-processing
Missing Reaction Context	Reactants, agents, solvents, or products not retrieved	LLM fails to follow full reaction-retrieval instructions
Incomplete Reaction Context	Retrieved results only partially correct (some but not all) expected entities	Query is incorrect semantically, commonly due to variable reuse

TABLE 6: Error types observed in Cypher query generation across prompt versions and prompting strategies.

4.2.3 Zero-shot vs One-shot Setting

One-shot prompting generally improves retrieval performance for Cypher generation compared to the zero-shot baseline, resulting in higher mean precision, recall, and F1 scores, with more concentrated distributions (Fig. 11). This indicates effective in-context learning from the provided demonstrations, supported by a lower rate of incorrect reactant directionality and a lower retrieval rate of incomplete reaction context due to semantically incorrect queries (Table 6, Fig. 11). Additionally, each prompt version (1–5) produces fewer missing reaction components in the one-shot setting than in the corresponding zero-shot version (Fig. 9), implying that the LLM follows contextual retrieval instructions more reliably when given an example.

The effect is particularly evident with minimal Prompt 1 and instruction-heavy Prompt 5, both of which achieve significantly higher F1-scores in the one-shot setting compared to zero-shot (Fig. S3-S8). Notably, minimal instructions lead to greater improvements, which is consistent with findings from prior studies [71].

4.2.4 Complex Cypher Queries Benefit Most from Demonstrations

As expected, the simpler task *Product-to-Reaction Mapping* showed showed little to no statistically significant improvement from zero-shot to one-shot when comparing the same

prompt version. This is likely because it aligns most closely with the intent of retrieving the full reaction context. In contrast, the *Condition-Based: Agents & Solvents* and *Precursor Identification* tasks, which share the same Cypher query template, did achieve significant performance gains, but mainly in recall (Fig. 9, Fig.S7, Fig.S5). The more complex tasks that require additional filtering logic benefited from the demonstrations the most, with improvements not just in recall, corresponding to correct contextual retrieval, but also precision, indicating accurate query generation (Fig. 9, Fig.S4, Fig.S6, Fig.S3). Overall, this suggests that the value of demonstrations scales with the task complexity.

4.2.5 Demonstration Similarity Leads to Better Performance

Dynamic semantic selection on average achieved the best retrieval performance, except for Prompt 5, as visualized in Fig. 10. This could be attributed to the fact that the selected example aligns more closely with the question, or in other words, that it is structurally similar to the gold query, even if its intent differs. With random demonstration selection, perfect alignment on the tasks involving more complex filtering logic (*Multi-Product Reaction*, *Co-product Identification*, and *Best-yielding Reaction*) occurs on average in only in $50\pm1\%$ of cases across 100,000 runs. In contrast, embedding-based selection achieves a higher alignment rate of 98.2%, nearly double that of the random approach.

To test whether structural alignment of demonstrations systematically improved retrieval, results across the aforementioned tasks were pooled and compared between aligned (under embedding-based selection) and misaligned (under random selection) cases using Wilcoxon signed-rank tests. F1 scores were significantly higher under alignment for Prompts 1–3 ($W = 4210$, $p = 2.3 \times 10^{-7}$; $W = 4212$, $p = 6.6 \times 10^{-10}$; $W = 4941.5$, $p = 1.4 \times 10^{-4}$), whereas Prompt 4 showed no significant difference ($W = 4967$, $p = 0.125$). This could potentially be due to duplicate data issues, which appeared 4.5 times less frequently in the misaligned cases.

4.2.6 Duplicate Data Issues

Interestingly, Prompt 4, which includes the dataset-specific guidelines to filter out reactions with null yields, performs better in a zero-shot setting for the *Best-yielding Reaction* task. Under static and random one-shot selection, however, both the mean recall and F1 scores decrease, while their variance increases. Even though embedding-based selection partially restores performance, it still remains significantly worse than the zero-shot baseline. Error analysis revealed the cause is unlikely to be the prompt design itself but duplicate entries in the dataset, only differing in a single agent or solvent. As shown in Fig. S1, duplicate data is a major source of error for all one-shot strategies, as the task requires only the top reaction to be selected using `LIMIT 1`. This outcome could be explained by chance, as result ordering in `neo4j` is non-deterministic. In particular, query execution can follow different plans with concurrent processing [72], leading to duplicates appearing unpredictably across runs.

4.2.7 Chemical Entities

Another common types of error, most frequently occurring in the *Multi-product Reaction* task (Fig.S1), are altered SMILES strings. These range from ions such as `[Cl-]` being "normalized" to `Cl`, to less common atoms substitutions (e.g., `S` → `C`), or even the loss of entire SMILES parts. As ions often appear as byproducts in the USPTO dataset, the aforementioned task was affected the most.

A likely explanation is related to tokenization and bias in the training data. The model breaks down text into tokens, where `[Cl-]` gets tokenized[3] into `[`, `Cl`, `]`, and `-`. Chloride ions are probably more commonly represented as `Cl-` in the training data, while brackets are widely used in non-chemical contexts. As a result, the bracketed sequence might have a lower probability of being reproduced exactly as is. Similarly, atoms and groups that occur less frequently in the pretraining corpus could be more prone to substitutions or modifications.

4.2.8 Evaluation Bias

Among factors that might bias the evaluation are the expected format of the retrieved results and the matching procedure itself. However, when analyzing matched and unmatched keys, no unexpected combinations were observed, as can be seen in Table 7. The only unmatched keys encountered were: (i) singular forms such as `agent`, `precursor`, and `product`, which remain unmatched only when their plural counterparts are matched; (ii) `yield`-related keys for cases where the query does not explicitly ask about product yield; and (iii) `precursors/ direct_precursors` in cases where reactants were already matched.

Gold Key	Observed Variants (Matched)
reactants	`direct_precursors`, `precursor`, `precursor_name`, `precursor_names`, `precursors`, `reactants`
products	`precursors`, `product`, `productName`, `product_name`, `products`, `products1`, `products2`, `target_product`
solvents	`solvents`
agents	`agent`, `agentName`, `agent_names`, `agents`
target_yield	`best_yield`, `reaction_yield`, `rel.yield`, `yield`
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`

TABLE 7: Mapping from gold keys to observed key variants in reaction-graph query results.

Similarly, in the *Co-product Identification* task, the gold answers include both product and co-product keys, which are not always returned by the generated queries. Hence, the recall and F1 scores are consistently clustered in a narrow band just below the perfect score of 1.0 (Fig. 9). This design choice introduces a structural limitation; however, it does not necessarily affect semantic correctness.

4.3 Multi-step Synthesis vs Single-Step Synthesis

Initial attempts to retrieve the full reaction context for the multi-step reaction data resulted in a high ($> 95\%$) rate

[3]https://platform.openai.com/tokenizer

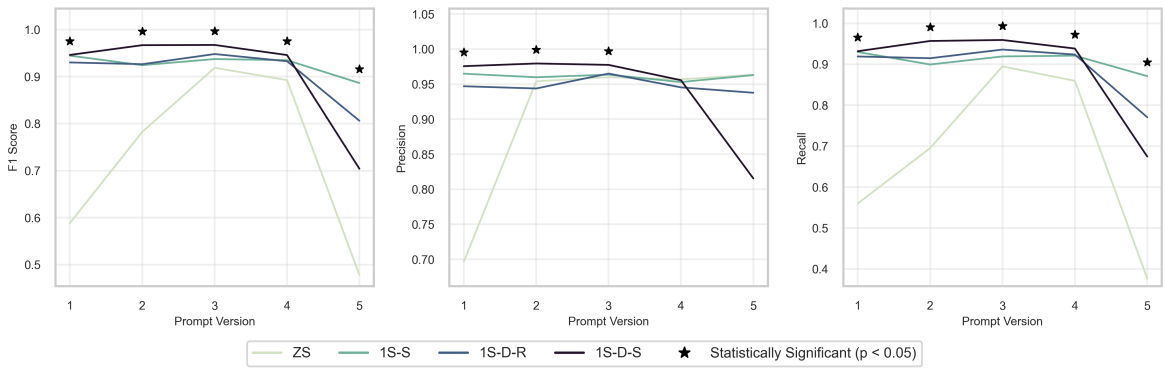


Fig. 10: Mean F1, precision, and recall, which each prompt is ranked by, are shown for five prompt versions under four settings: zero-shot (ZS), one-shot static (1S-S), one-shot random (1S-D-R), and one-shot semantic (1S-D-S). The x-axis represents the prompt version, while the y-axis shows the corresponding evaluation metric. Ranking at each prompt version is determined by the corresponding mean score. The top-ranked strategy is compared against the second-best, with a star (*) marking cases where this difference is statistically significant after Holm–Bonferroni correction.

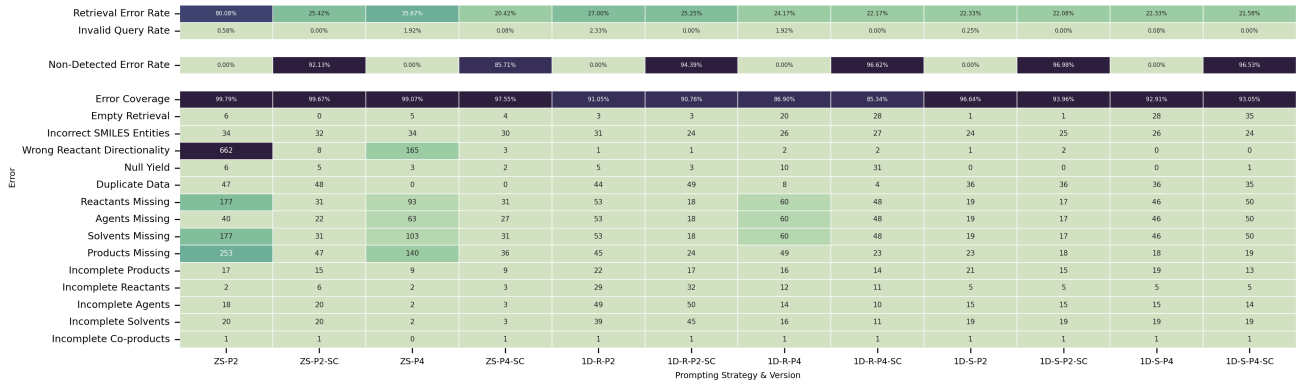


Fig. 11: Retrieval Error Category Distribution. Retrieval error = proportion of samples with a non-perfect F1 score. Query validity = proportion of valid queries relative to all samples. Error coverage = percentage of samples containing one of the errors reported. Below are the main categories of retrieval errors across four prompting strategies (zero-shot, one-shot static, one-shot random, and one-shot semantic.) and five prompt versions of varying contextual and structural guidance: (1) – base contextual query, (2) – (1) + explicit MATCH instructions, (3) – (2) + directionality check, (4) – (3) + yield-filtering, and (5) – (4) + additional full context rules. Each cell shows the number of results containing the given error out of 1200 data samples in total.

of invalid or semantically incorrect queries. This could be attributed to the complexity of the required Cypher. Therefore, the task was reformulated to retrieve only reaction nodes relevant to the input question. Consequently, this change decreased the number of potential error sources, which are summarized in Table 8. The main results are reported in Fig.12, with the corresponding error analysis in Fig.13.

Once again, one-shot prompting outperforms zero-shot, as reflected by higher exact path precision, recall, and F1, along with partial path recall (Fig. 9, Fig.S6, Fig.S12, Fig.S11). In contrast to single-step retrieval, the pattern is reversed here, as precision in the zero-shot setting has greater variance than recall. Because the LLM fails to enforce traversal direction (Prompts 3-5), paths in both directions with respect to the target product are returned. In other words, it includes reaction chains leading to both upstream precursors and downstream derivatives of the target. As a result, precision and F1 scores are no longer strictly 0 or 1 for such cases.

Similarly, partial recall that rewards overlapping subpaths is higher compared to the exact path metrics mainly in zero-shot. Quantitatively, one-shot reduces hop-count errors by 100% for nearly all prompt versions; exceptions are 1S-P1-p (78%), 1S-P1-i (45.7%), 1D-R-P1 (54.2%), 1D-S-P1(89.8%), 1D-R-P2 (96.6%), and 1D-R-P5 (99.1%), see Fig. 13. Notably, the effect holds even given that the number of reaction steps is outside of the stated mapping (e.g., $2 \rightarrow 4$, $3 \rightarrow 6$, $4 \rightarrow 8$, while the example is $n = 5$). These results indicate that structural cues from the demonstration, rather than explicit instruction, are what drives the improvement.

4.3.1 *Intermediate-Molecule Identification is the Hardest Task*

Intermediate Identification was the weakest task, specifically under zero-shot, one-static static with the product-discovery exemplar, and one-shot random selection. Unlike reaction pathway tasks (*Reaction Chain to Target*, *Reaction Chains Between Molecules*), it does not naturally imply returning

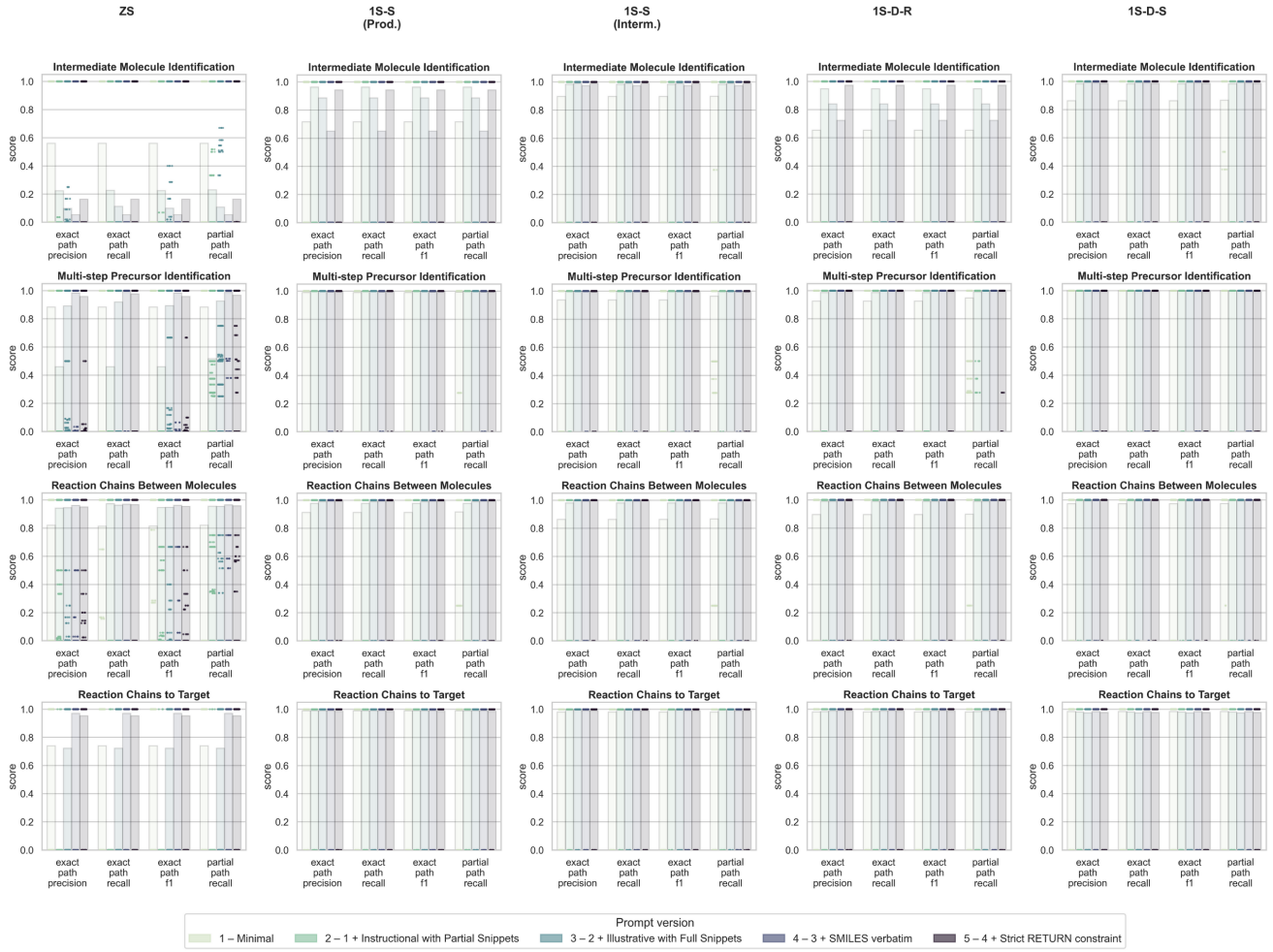


Fig. 12: Comparison of multi-step retrieval performance for generated Cypher queries vs ground truth. Each subplot reports the distributions of exact-path precision, recall, F1, and partial-path recall for a given query type under different prompting strategies: zero-shot, one-shot static, one-shot random, and one-shot semantic. Within each setting, five prompt versions (four for zero-shot) reflect increasing levels of contextual and structural guidance.

reaction nodes, but *molecule* nodes. In other words, this introduces a mismatch between the required output of "reaction nodes only" and the input question. In contrast, the model demonstrated strong performance on *Precursor Identification*, a task that also does not inherently require synthesis pathways to be returned. Evidence for this includes near-perfect one-shot metrics across all prompt versions (Fig.12) and low retrieval error rates (Fig. S2). A plausible explanation is that retrosynthesis is conventionally framed around precursors/reactants, so it likely creates a strong semantic bias that reinforces the given instruction.

Therefore, alignment mattered most for this task, as a single intermediate discovery example (even for forward synthesis) resolved the ambiguity in the "return reaction nodes only" constraint. As a result, the Cypher template generalized naturally to the other tasks, which reuse the same traversal logic but match the objective more closely. This also explains why semantic demonstration selection achieved similar performance, as the intermediate example was selected in 89% of cases. In contrast, random and static strategies based on other examples resulted in lower precision, recall, and F1 across all prompts. A similar trend has been observed in recent studies on hard-to-easy consistency in in-context learning [73]. In particular, it was shown that exposure to harder demonstration can sometimes improve performance on simpler tasks, however, only verified on certain domains and smaller models. Overall, further investigation is required to determine whether this effect reflects a broader pattern.

4.3.2 *Endpoint Anchoring Errors Are a Zero-Shot Problem*

As expected, the zero-shot setting prompts perform significantly worse compared to the corresponding variants under each one-shot strategy. This primarily stems from endpoint anchoring errors, where the target molecule is incorrectly bound as the start node with subsequent variable reuse or undirected exploration. A single demonstration in the one-shot setting prevents this by implicitly establishing the convention that reactants are always positioned to the left of the (`-[:REACTS_IN|PRODUCES*..n]->`) relation.

To investigate the anchoring issues further, the LLM was separately prompted to classify the role of the mentioned molecule for each question in the *Intermediate Discovery* and *Reaction-To-Target* tasks. In particular, it labeled the molecule as "product" 100% of the time, indicating correct semantic

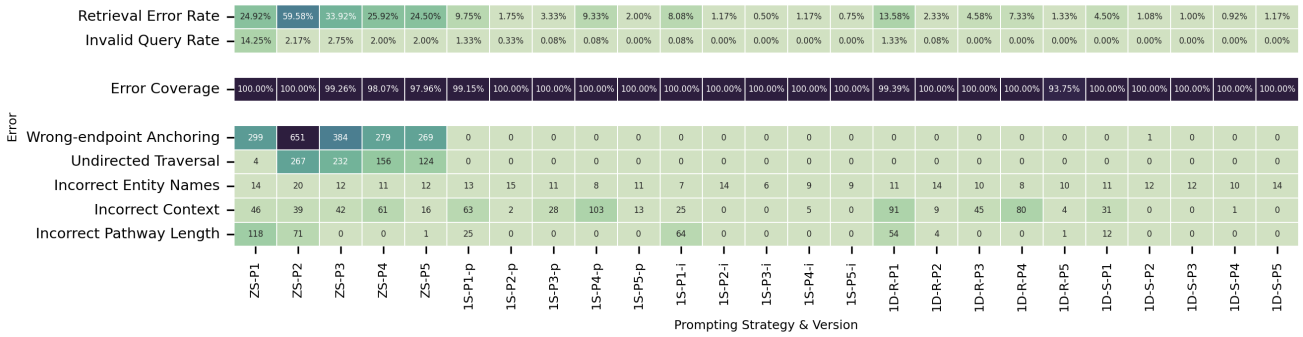


Fig. 13: Retrieval Error Category Distribution. Retrieval error = proportion of samples with a non-perfect F1 score. Query validity = proportion of valid queries relative to all samples. Error coverage = percentage of samples containing one of the errors reported. Below are the main categories of retrieval errors across four prompting strategies (zero-shot, one-shot static, one-shot random, and one-shot semantic.) and five prompt versions of varying contextual and structural guidance: (1) – minimal prompt, (2) – (1) + instructional with partial Cypher snippets, (3) – (2) + illustrative with full Cypher Snippets, (4) – (3) + SMILES verbatim, and (5) – (4) + strict RETURN constraint. Each cell shows the number of results containing the given error out of 1200 data samples in total.

Error Type	Description	Interpretation
Wrong-endpoint Anchoring	Target molecule is incorrectly anchored as the starting node of the traversal	Causes the query to retrieve reversed or cyclic paths, typically non-existent
Undirected Traversal	`-[...]-` vs. `-[...]->`	LLM fails to enforce direction, potentially causing a combinatorial explosion
Incorrect Pathway Length	Reaction steps are not correctly translated to hop counts in the KG, given correct context is retrieved	Step–hop mapping from the prompt is applied incorrectly, leading to paths that are typically shorter
Incorrect Context	Molecule nodes (e.g., precursors or intermediates) are returned instead of reaction nodes	LLM fails to follow retrieval instructions, likely biased by the wording of the input question

TABLE 8: Error types observed in multi-step Cypher query generation across prompt versions and prompting strategies.

interpretation. This suggests the one-shot example provides structural grounding and an "anchoring convention", rather than making the question itself less ambiguous.

4.3.3 SMILES Fidelity vs Context Trade-Off

Not surprisingly, SMILES fidelity remains a recurring challenge for multi-step Cypher generation in reaction KG (Fig. 13). The incorporation of the "copy SMILES verbatim" instruction (Prompt 3 vs Prompt 4) yielded a negligible 0.08pp average reduction in entity errors (Fig. 13). The relative change in entity-error rate (P3→P4) was: ZS -8.33%; 1S-p -27.33%; 1S-i +33.3%; 1D-R -20.0%; 1D-S -16.7%. Nonetheless, incorrect-context errors, i.e. cases where the LLM fails to return the required reaction chains, increased by 2.25pp, with the relative change (P3→P4) of: ZS +45.2%; 1S-p +267%; 1S-i +100%; 1D-R +77.7%; 1D-S +100%. Restating the output contract in Prompt 5 restored performance, yielding a statistically significant gain over Prompt 3 but not over Prompt 2 (Fig.S9).

Overall, these results indicate that prompt engineering alone is insufficient for handling SMILES. Future research should explore more robust approaches, such as decoupling entity processing from Cypher generation. For example, this could be achieved by extracting SMILES from the input question, masking them with placeholders during generation, and reinserting the original strings post-hoc.

4.3.4 Nearly Perfect Surface-Level Metric Scores for Cypher Queries

As visualized in Fig. 14, adding a single demonstration leads to nearly perfect scores on BLEU, METEOR, and ROUGE-L F1, with instruction differences across prompts becoming negligible. This is expected, since all examples share an almost identical template with the corresponding target query, typically only requiring a precursor↔product condition flip. In zero-shot, Prompts 2-5 outperform Prompt 1, simply because they instruct the model to use the compact path alternation pattern (`[:REACTS_IN|PRODUCES*..n]`), consistent with the ground truth queries. Here, Prompt 1 does not overly constrain the output, as a result producing multiple valid query variants, e.g., with step-by-step chaining. This explains lower-centered distributions and higher variance.

Once, again the effect of structural alignment of examples on string-based metrics is apparent. For *Multi-step Precursor Identification* (and similarly for *Intermediate Identification*), the tightest, near-perfect distributions occur when the example matches the task's logical intent, which is the case under one-shot static selection with a matching example and under dynamic semantic selection (Fig. 14).

Finally, METEOR is the highest-scoring metric across both zero-shot and one-shot settings, as it rewards single token-level coverage. BLEU and ROUGE-L F1 are slightly lower in certain cases, as they are more sensitive to token-order variation. Most importantly, these string-based metrics can overestimate query quality and do not necessarily translate into semantic correctness. For

instance, small modifications, like making a directed pattern undirected (`-[:REACTS_IN|PRODUCES*..n]->` vs `-[:REACTS_IN|PRODUCES*..n]-`), yield a high token overlap yet cause large retrieval errors. As concluded previously, BLEU/ROUGE-L/METEOR only measure surface-form alignment, and are not suitable to reliably assess retrieval accuracy.

4.4 Cypher Validation & Correction

4.4.1 Effectiveness of Checklist-Driven CoVe Self-correction Is Dependent on Initial Query Quality

To evaluate whether Cypher generation performance for single-step retrieval could be improved further, a checklist-driven CoVe-style self-correction was applied to Prompt 2 and Prompt 4 (Fig. 15). The setup was augmented with an experimental, rule-based directionality corrector from the `LangChain_Neo4j` library to fix relationship directions deterministically. In the zero-shot setting, 98% of directionality errors were resolved compared to the corresponding prompt variant without self-correction (Fig.16). Prompt 4 was selected for its strong baseline performance, whereas Prompt 2 was kept as a simplified version of Prompt 3, since explicit directionality instructions were no longer required.

Analysis revealed that checklist-driven CoVe self-correction is most effective in the zero-shot setting, where the baseline performance is the weakest. Under one-shot random prompting, performance gains (if present) were modest and task-specific, in particular, for *Best-yielding Reaction* and *Co-Product Identification*. Combined with one-shot demonstration selection for Cypher generation, self-correction failed to yield significant improvement, and instead, occasionally reintroduced errors. In particular, no new error types emerged, only trade-offs among existing variants.

The main point of failure in this setup was the LLM validator. The observed non-detected error rates were exceptionally high, ranging from 85.71% to 94.98% (Fig. 15). This suggests that the validator is too generic for the specific demands of the task at hand and potentially requires explicit few-shot examples of the main error types. Once again, the objective here is not simply to generate a valid and semantically correct query, but also to fetch the full reaction context.

Ultimately, when a model is already producing high-quality outputs, a generic correction loop becomes infective. A potential solution is to use specialized validators for distinct error types or to decompose the task itself: first, generating a query that answers the question and then expanding it to retrieve the full context. Future research should test these alternatives.

4.4.2 Exemplars, Not Self-Correction, Drive Performance

In the multi-step retrieval, self-correction yielded higher evaluation metrics, but almost exclusively for Prompt 1. This is the case for all tasks, except for *Intermediate Identification* under one-shot static setting, in which the exemplar was aligned with the intent of both the task and the question. Analysis of per-query Cypher error history revealed that the validator often oscillates between: (1) correctly identifying non-reaction-only outputs in one iteration; and (2) detecting semantic drift from the question's intent in the next iteration (e.g., the question asks about intermediate molecules, not reaction pathways).

Although the validator was meant to check only the structure of the generated queries, its prompt also contained the original input question to extract the required step count. Hence, this likely implicitly biased the validator towards the semantics of the question, which contradicts with the reaction-only output constraint. Additionally, the corrector only performs minimal edits as flagged by the validator, initially designed this way to avoid introducing new errors. These observations are consistent with the fact that examplar alignment, both structural and semantic with the task's goal, is the primary driver of multi-step performance. Here, generic self-correction yields diminishing returns beyond already a strong one-shot prompt.

5 Conclusion

This thesis examined whether LLMs can be reliably grounded in a reaction knowledge graph to support the task of synthesis planning via Text2Cypher query generation. In particular, a bipartite reaction KG was designed and populated, which effectively enables single- and multi-step retrieval. Prompt engineering strategies for Cypher generation were analyzed, where for each task, a reproducible data generation pipeline and a task-specific evaluation protocol were developed. In addition, a retrieval error taxonomy was identified and discussed.

Across both single- and multi-step retrieval, one-shot prompting typically outperformed zero-shot prompting. Single-step queries, especially those with more complex filtering/aggregation, benefited the most from a demonstration that closely aligned with their target query's structure and logical intent. Therefore, embedding-based example selection achieved the highest evaluation scores on Prompts 1-4. When a prompt contains instructions that are too verbose and rigid on query generation, performance can drop significantly and is not always recoverable with a demonstration.

Overall, certain errors detected, e.g., "directionality of reactants", were almost exclusively specific to zero-shot prompting for single-step retrieval. Schema design might help mitigate these errors, but its effect should be investigated more systematically. Ultimately, this reemphasizes the importance of demonstrations for grounding structured query generation. Furthermore, the matching evaluation procedure also proved to be robust, as no unexpected matches were observed.

For multi-step retrieval, the task was reformulated to return only reaction nodes rather than the full reaction context (i.e., all relevant reaction components even those not explicitly mentioned in the question). This choice standardizes outputs for downstream use, preserving their utility under partial retrieval. Hence, the number of potential error sources reduced, but the resulting ones differed from those in the single-step setting.

Here, the largest performance gains of one-shot prompting compared to zero-shot arose from eliminating zero-shot specific anchoring and traversal-direction errors. Semantic alignment offered little additional benefit, as a single static exemplar of the hardest task, *Intermediate Identification*, achieved comparable performance. The effect extends

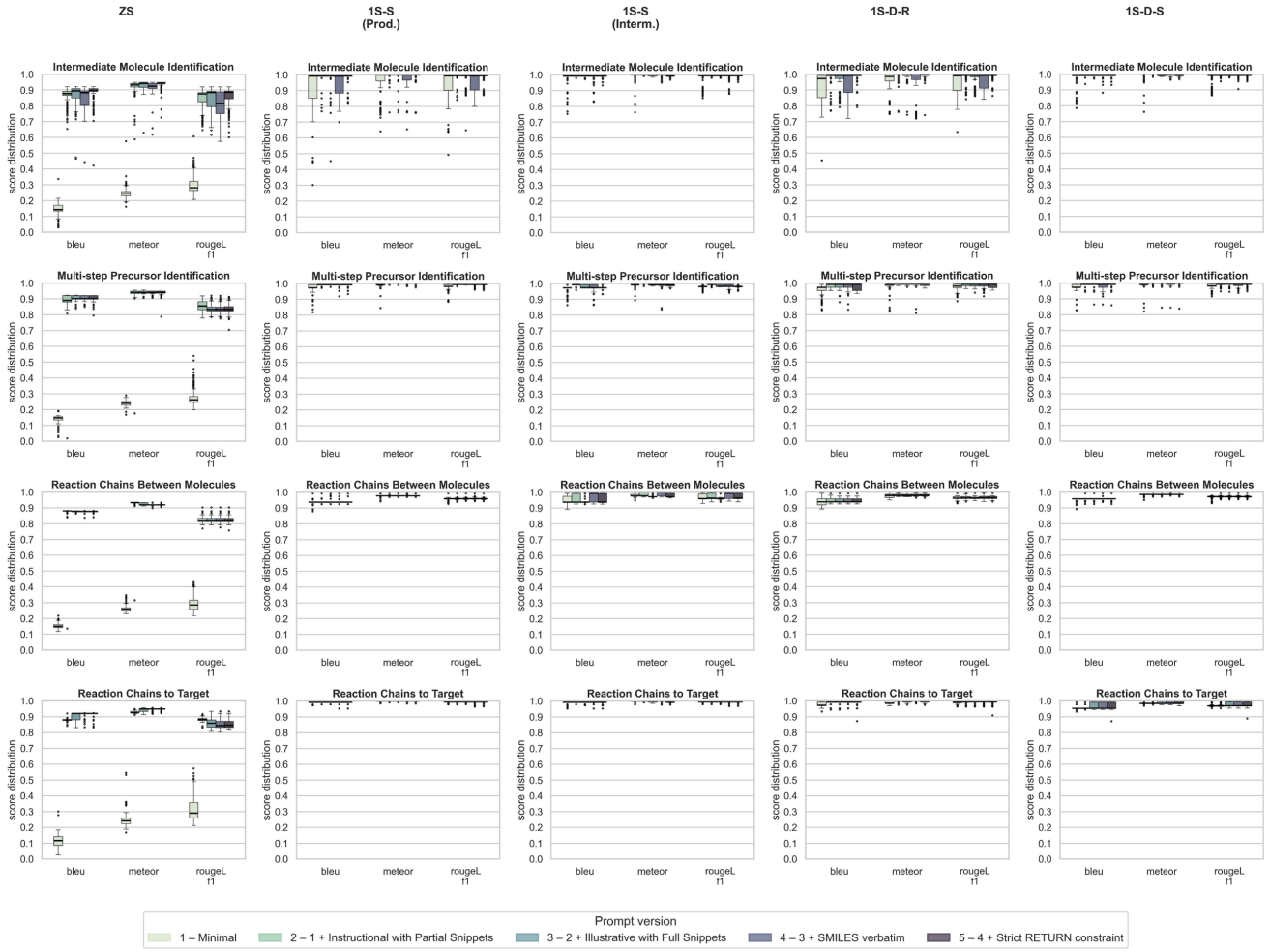


Fig. 14: Comparison of generated Cypher queries to ground truth queries for Multi-step Synthesis. Each subplot shows BLEU, METEOR, and ROUGE-L F1 score distributions for a specific query type under different prompt settings: zero-Shot, one-shot static, one-shot random, and one-shot semantic. Within each setting, five prompt versions (color-coded) reflect increasing levels of contextual and structural guidance.

beyond structural alignment, indicating that exposure to correct question–output mapping is critical for the Cypher generation task under the given conditions.

While checklist-driven self-correction was evaluated, its impact remained limited compared to strong one-shot prompting. Performance improves drastically when the baseline is weak initially, but not once the examplar grounds the model in structural and semantic task requirements.

To conclude, this work shows that a general purpose LLM of the GPT family (`GPT-4.1-mini`) can interact with reaction KGs to support synthesis planning through Text2Cypher, though its performance remains imperfect. This suggests that LLMs hold promise for this domain, but alternative approaches, specifically prompt-engineering, should be explored further.

5.1 Limitations

This study has several limitations:

- **Model Specificity:** The study relied exclusively on a single LLM, `GPT-4.1-mini`. This implies that the findings on query generation might not be generalizable to other models with different architectures or training data.
- **Dataset:** Both evaluation datasets contained 1200 reactions, which limits statistical power and generalizability to larger corpora.
- **Prompting Techniques:** This work mainly focused on zero-shot, one-shot prompting with three distinct demonstration selection strategies, and checklist-driven CoVe-Style prompting. Other prompting techniques, such as CoT or few-shot prompting with multiple demonstrations, were not evaluated for the given task.
- **SMILES Handling** Persistent errors in copying of SMILES verbatim were observed. These are irreducible under current prompting, but the work did not experiment with more robust alternatives.
- **Validator–Corrector Setup:** The prompt engineering experiments for the validator and corrector LLMs were not explored extensively. Prompt design was guided by error analysis, and mainly focused on the validator. Different setups with multiple specialized

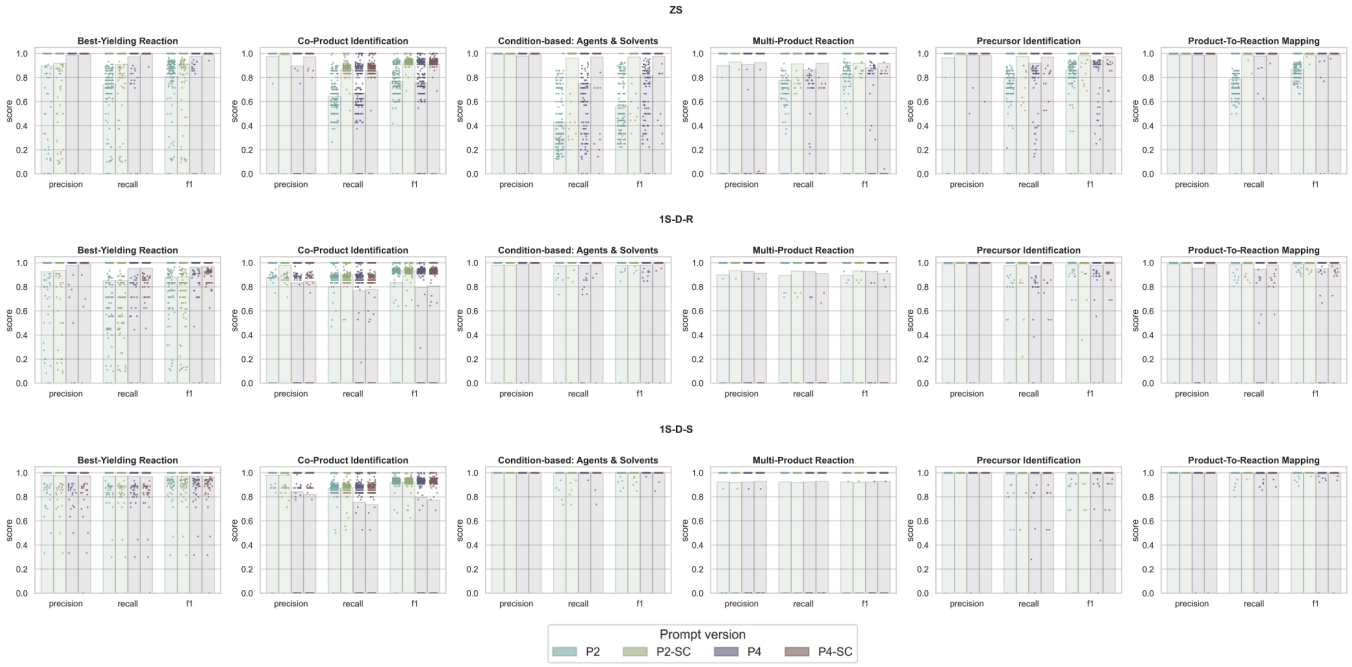


Fig. 15: Comparison of single-step retrieval performance for generated Cypher queries vs ground truth under Direct Prompting vs Checklist-Driven CoVe Style Prompting. Each subplot reports the distributions of exact-path precision, recall, F1, and partial-path recall for a given query type under different prompting strategies: zero-shot, one-shot random, and one-shot semantic. Within each setting, prompts P2 and P4 are evaluated with and without self-correction.

validators were not tested.

- **Resource constraints** Several choices (prompt grid size, validator–corrector pipeline, one model family, 1,200-reaction subsets) were bounded by a fixed compute/API budget. Experiments with the highest expected impact were prioritized.

5.2 Future Work

This study opens several avenues for further research:

- **Few-shot Prompting:** Evaluate few-shot prompting and exemplar-selection strategies, test order/diversity effects and compare structural vs. semantic alignment against the one-shot baseline.
- **SMILES Handling:** Incorporate a separate entity validator or mask-and-restore SMILES during generation.
- **Task Decomposition** Evaluate an alternative two-stage pipeline: (1) generating a query to answer the given question; (2) modifying/expanding it to retrieve reaction context.
- **Validator–Corrector** Evaluate performance of multiple specified validators, one per each error type.

6 Disclosure

In line with TU Delft publishing policies[4], I acknowledge that certain passages were rephrased using ChatGPT (OpenAI)

[4]https://www.tudelft.nl/library/actuele-themas/open-publishing/about/policies

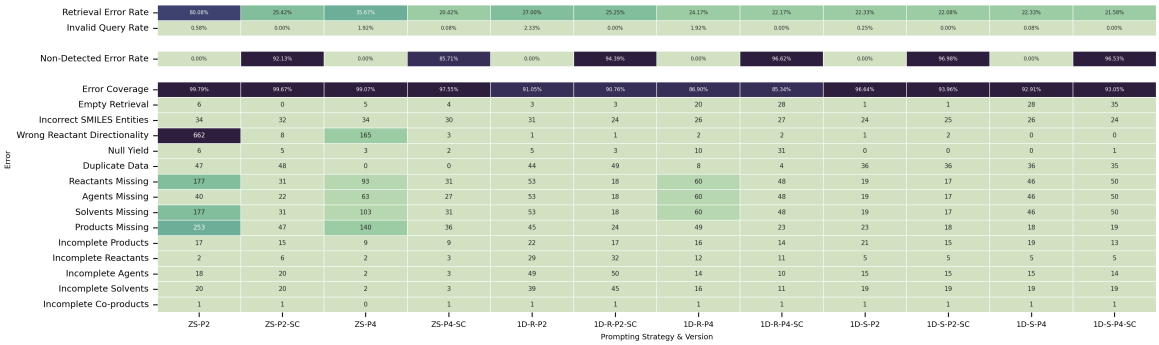


Fig. 16: Distribution of retrieval errors for direct prompting vs. a checklist-driven (CoVe style) approach. Error analysis covers three prompting strategies (zero-shot, one-shot random, and one-shot semantic) and two prompt versions, P3 and P4, with and without self-correction.

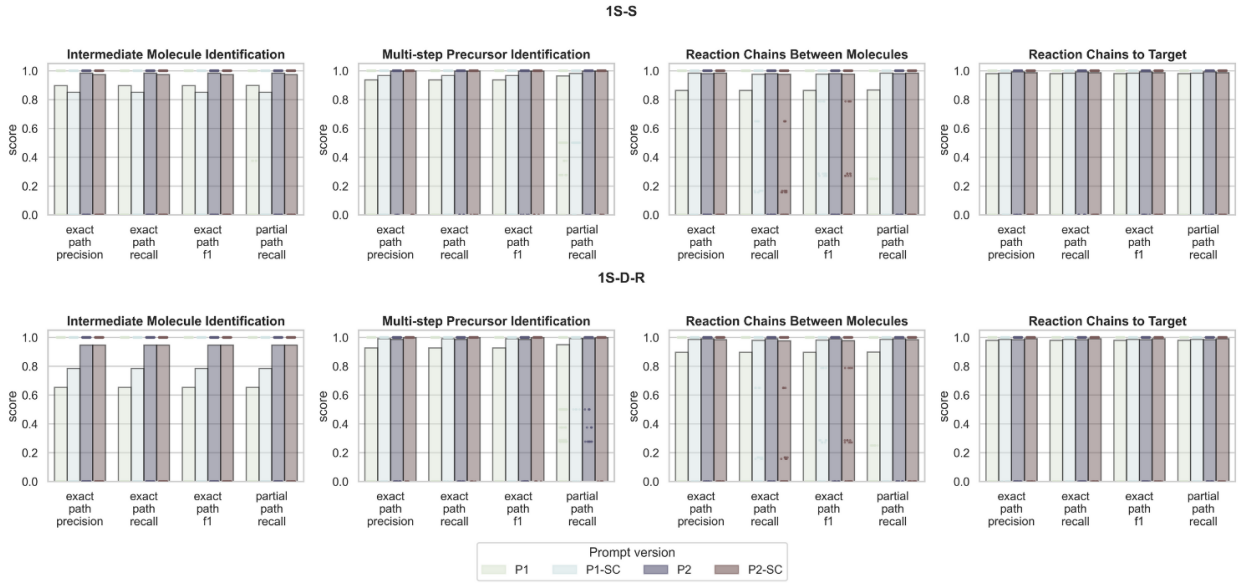


Fig. 17: Comparison of multi-step retrieval performance for generated Cypher queries vs ground truth under Direct Prompting vs Checklist-Driven CoVe Style Prompting. Each subplot reports the distributions of exact-path precision, recall, F1, and partial-path recall for a given query type under different prompting strategies: zero-shot, one-shot random, and one-shot semantic. Within each setting, prompts P1 and P2 are evaluated with and without self-correction.

and that GitHub Copilot was used to assist with code generation. All AI generated context, however, was critically reviewed and verified.

REFERENCES

[1] Q. Zhang *et al.*, "Scientific Large Language Models: A Survey on Biological & Chemical Domains," 1 2024. [Online]. Available: https://arxiv.org/abs/2401.14656v2

[2] S. M. Kearnes *et al.*, "The open reaction database," *Journal of the American Chemical Society*, vol. 143, no. 45, pp. 18 820–18 826, 11 2021. [Online]. Available: https://pubs.acs.org/doi/full/10.1021/jacs.1c09820

[3] A. Grattafiori *et al.*, "The Llama 3 Herd of Models," 7 2024. [Online]. Available: https://arxiv.org/abs/2407.21783v3

[4] D. S. Wigh, J. Arrowsmith, A. Pomberger, K. C. Felton, and A. A. Lapkin, "ORDerly: Data Sets and Benchmarks for Chemical Reaction Data," *Journal of Chemical Information and Modeling*, vol. 64, no. 9, pp. 3790–3798, 5 2024. [Online]. Available: https://pubs.acs.org/doi/full/10.1021/acs.jcim.4c00292

[5] T. Guo *et al.*, "What can Large Language Models do in chemistry? A comprehensive benchmark on eight tasks," *Advances in Neural Information Processing Systems*, vol. 36, pp. 59 662–59 688, 12 2023. [Online]. Available: https://github.com/ChemFoundationModels/ChemLLMBench.

[6] B. Yan, A. Chen, and K. Cho, "Inconsistency of LLMs in Molecular Representations," 12 2024. [Online]. Available: https://chemrxiv.org/engage/chemrxiv/article-details/675b9de27be152b1d0ced2b5

[7] S. M. T. I. Tonmoy *et al.*, "A Comprehensive Survey of Hallucination Mitigation Techniques in Large Language Models," 1 2024. [Online]. Available: https://arxiv.org/abs/2401.01313v3

[8] S. Minaee *et al.*, "Large Language Models: A Survey," 2 2024. [Online]. Available: https://arxiv.org/abs/2402.06196v2

[9] P. Lewis *et al.*, "Retrieval-Augmented Generation for Knowledge-Intensive NLP Tasks," *Advances in Neural Information Processing Systems*, vol. 2020-December, 5 2020. [Online]. Available: https://arxiv.org/abs/2005.11401v4

[10] B. Perozzi *et al.*, "Let Your Graph Do the Talking: Encoding Structured Data for LLMs," 2 2024. [Online]. Available: https://arxiv.org/abs/2402.05862v1

[11] D. Edge *et al.*, "From Local to Global: A Graph RAG Approach to Query-Focused Summarization," 4 2024. [Online]. Available: http://arxiv.org/abs/2404.16130

[12] S. Pan *et al.*, "Unifying Large Language Models and Knowledge Graphs: A Roadmap," *IEEE Transactions on Knowledge and Data Engineering*, vol. 36, no. 7, pp. 3580–3599, 6 2023. [Online]. Available: http://arxiv.org/abs/2306.08302http://dx.doi.org/10.1109/TKDE.2024.3352100

[13] A. Kau *et al.*, "Combining Knowledge Graphs and Large Language Models," 7 2024. [Online]. Available: https://arxiv.org/abs/2407.06564v1

[14] N. Ibrahim, S. Aboulela, A. Ibrahim, and R. Kashef, "A survey on augmenting knowledge graphs (KGs) with large language models (LLMs): models, evaluation metrics, benchmarks, and challenges," *Discover Artificial Intelligence*, vol. 4, no. 1, pp. 1–28, 12 2024. [Online]. Available: https://link.springer.com/article/10.1007/s44163-024-00175-8

[15] S. Schulhoff *et al.*, "The Prompt Report: A Systematic Survey of Prompt Engineering Techniques," 6 2024. [Online]. Available: https://arxiv.org/pdf/2406.06608

[16] P. Sahoo *et al.*, "A Systematic Survey of Prompt Engineering in Large Language Models: Techniques and Applications," 2 2024. [Online]. Available: https://arxiv.org/pdf/2402.07927

[17] B. Chen, Z. Zhang, N. Langrené, and S. Zhu, "Unleashing the potential of prompt engineering for large language models," *Patterns*, vol. 6, no. 6, 6 2025. [Online]. Available: http://arxiv.org/abs/2310.14735http://dx.doi.org/10.1016/j.patter.2025.101260

[18] Y. Zhou *et al.*, "The Mystery of In-Context Learning: A Comprehensive Survey on Interpretation and Analysis," *EMNLP 2024 - 2024 Conference on Empirical Methods in Natural Language Processing, Proceedings of the Conference*, pp. 14 365–14 378, 11 2024. [Online]. Available: https://arxiv.org/pdf/2311.00237

[19] M. Luo, X. Xu, Y. Liu, P. Pasupat, and M. Kazemi, "In-context Learning with Retrieved Demonstrations for Language Models: A Survey," *Transactions on Machine Learning Research*, vol. 2024, 1 2024. [Online]. Available: https://arxiv.org/pdf/2401.11624

[20] C. Qin, A. Zhang, C. Chen, A. Dagar, and W. Ye, "In-Context Learning with Iterative Demonstration Selection," *EMNLP 2024 - 2024 Conference on Empirical Methods in Natural Language Processing, Findings of EMNLP 2024*, pp. 7441–7455, 10 2023. [Online]. Available: https://arxiv.org/pdf/2310.09881

[21] D. Shu and M. Du, "Comparative Analysis of Demonstration Selection Algorithms for LLM In-Context Learning," *UPRISE*, pp. 29 490–29 492, 10 2024. [Online]. Available: https://arxiv.org/pdf/2410.23099

[22] F. Strieth-Kalthoff *et al.*, "Artificial Intelligence for Retrosynthetic Planning Needs Both Data and Expert Knowledge," *Journal*

of the American Chemical Society, 2024. [Online]. Available: https://pubmed.ncbi.nlm.nih.gov/38598363/

[23] G. Gricourt, P. Meyer, T. Duigou, and J. L. Faulon, "Artificial Intelligence Methods and Models for Retro-Biosynthesis: A Scoping Review," *ACS Synthetic Biology*, vol. 13, no. 8, pp. 2276–2294, 8 2024. [Online]. Available: /doi/pdf/10.1021/acssynbio.4c00091?ref=article_openPDF

[24] S. Liu, D. Zhang, Z. Tu, H. Dai, and P. Liu, "Evaluating Molecule Synthesizability via Retrosynthetic Planning and Reaction Prediction," 11 2024. [Online]. Available: https://arxiv.org/pdf/2411.08306

[25] A. M. Bran, T. A. Neukomm, D. P. Armstrong, Z. Jončev, and P. Schwaller, "Chemical reasoning in LLMs unlocks strategy-aware synthesis planning and reaction mechanism elucidation," 7 2025. [Online]. Available: https://arxiv.org/pdf/2503.08537v1

[26] C. Zhang *et al.*, "SynAsk: Unleashing the Power of Large Language Models in Organic Synthesis," *Chemical Science*, vol. 16, no. 1, pp. 43–56, 6 2024. [Online]. Available: https://arxiv.org/pdf/2406.04593

[27] H. Wang *et al.*, "LLM-Augmented Chemical Synthesis and Design Decision Programs," 5 2025. [Online]. Available: https://arxiv.org/pdf/2505.07027

[28] C. Peng, F. Xia, M. Naseriparsa, and F. Osborne, "Knowledge Graphs: Opportunities and Challenges," *Artificial Intelligence Review*, vol. 56, no. 11, pp. 13 071–13 102, 11 2023. [Online]. Available: https://link.springer.com/article/10.1007/s10462-023-10465-9

[29] G. Agrawal, T. Kumarage, Z. Alghamdi, and H. Liu, "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, NAACL 2024*, vol. 1, pp. 3947–3960, 11 2023. [Online]. Available: https://arxiv.org/abs/2311.07914v2

[30] B. Peng *et al.*, "Graph Retrieval-Augmented Generation: A Survey," *J. ACM*, vol. 37, no. 4, 8 2024. [Online]. Available: https://arxiv.org/pdf/2408.08921

[31] S. Pan *et al.*, "Unifying Large Language Models and Knowledge Graphs: A Roadmap," *IEEE Transactions on Knowledge and Data Engineering*, vol. 36, no. 7, pp. 3580–3599, 1 2024. [Online]. Available: http://arxiv.org/abs/2306.08302http://dx.doi.org/10.1109/TKDE.2024.3352100

[32] H. Han *et al.*, "RAG vs. GraphRAG: A Systematic Evaluation and Key Insights," 2 2025. [Online]. Available: https://arxiv.org/pdf/2502.11371

[33] A. Cossette, Z. Blumenfeld, and D. Sanoja, *The Developer's Guide to GraphRAG*, 2025. [Online]. Available: https://neo4j.com/whitepapers/the-developers-guide-to-graphrag/

[34] Z. Guo, L. Xia, Y. Yu, T. Ao, and C. Huang, "LightRAG: Simple and Fast Retrieval-Augmented Generation," 10 2024. [Online]. Available: https://arxiv.org/pdf/2410.05779

[35] B. Sarmah *et al.*, "HybridRAG: Integrating Knowledge Graphs and Vector Retrieval Augmented Generation for Efficient Information Extraction," *ICAIF 2024 - 5th ACM International Conference on AI in Finance*, pp. 608–616, 11 2024. [Online]. Available: https://arxiv.org/pdf/2408.04948

[36] M. G. Ozsoy, L. Messallem, J. Besga, and G. Minneci, "Text2Cypher: Bridging Natural Language and Graph Databases," 12 2024. [Online]. Available: https://arxiv.org/pdf/2412.10064

[37] A. Clemedtson and B. Shi, "GraphRAFT: Retrieval Augmented Fine-Tuning for Knowledge Graphs on Graph Databases," 4 2025. [Online]. Available: https://arxiv.org/pdf/2504.05478

[38] I. Mandilara, C. Maria Androna, E. Fotopoulou, A. Zafeiropoulos, and S. Papavassiliou, "Decoding the Mystery: How Can LLMs Turn Text Into Cypher in Complex Knowledge Graphs?" *IEEE Access*, vol. 13, pp. 80 981–81 001, 2025.

[39] S. Soleymani, N. Gravel, K. Kochut, and N. Kannan, "Task Splitting and Prompt Engineering for Cypher Query Generation in Domain-Specific Knowledge Graphs," *bioRxiv*, p. 2025.04.23.649790, 4 2025. [Online]. Available: https://www.biorxiv.org/content/10.1101/2025.04.23.649790v1https://www.biorxiv.org/content/10.1101/2025.04.23.649790v1.abstract

[40] Y. Feng, S. Papicchio, and S. Rahman, "CypherBench: Towards Precise Retrieval over Full-scale Modern Knowledge Graphs in the LLM Era," pp. 8934–8958, 12 2024. [Online]. Available: https://arxiv.org/pdf/2412.18702

[41] Z. Zhong *et al.*, "SyntheT2C: Generating Synthetic Data for Fine-Tuning Large Language Models on the Text2Cypher Task," 6 2024. [Online]. Available: https://arxiv.org/pdf/2406.10710

[42] M. Hornsteiner *et al.*, "Real-Time Text-to-Cypher Query Generation with Large Language Models for Graph Databases," *Future Internet 2024, Vol. 16, Page 438*, vol. 16, no. 12, p. 438, 11 2024. [Online]. Available: https://www.mdpi.com/1999-5903/16/12/438/htmhttps://www.mdpi.com/1999-5903/16/12/438

[43] L. Pusch and T. O. F. Conrad, "Combining LLMs and Knowledge Graphs to Reduce Hallucinations in Question Answering," 9 2024. [Online]. Available: https://arxiv.org/abs/2409.04181v2

[44] M. G. Ozsoy, "Enhancing Text2Cypher with Schema Filtering," 5 2025. [Online]. Available: https://arxiv.org/pdf/2505.05118v1

[45] Y. Feng *et al.*, "Knowledge graph–based thought: a knowledge graph–enhanced LLM framework for pan-cancer question answering," *GigaScience*, vol. 14, pp. 1–12, 1 2025. [Online]. Available: https://dx.doi.org/10.1093/gigascience/giae082

[46] J. Delile, S. Mukherjee, A. Van Pamel, and L. Zhukov, "Graph-Based Retriever Captures the Long Tail of Biomedical Knowledge," 2 2024. [Online]. Available: https://arxiv.org/pdf/2402.12352

[47] F. L. Wang *et al.*, "LLM-KGMQA: large language model-augmented multi-hop question-answering system based on knowledge graph in medical field," *Knowledge and Information Systems*, vol. 67, no. 8, pp. 6461–6503, 8 2025. [Online]. Available: https://link.springer.com/article/10.1007/s10115-025-02399-1

[48] R. Yang *et al.*, "KG-Rank: Enhancing Large Language Models for Medical QA with Knowledge Graphs and Ranking Techniques," *BioNLP 2024 - 23rd Meeting of the ACL Special Interest Group on Biomedical Natural Language Processing, Proceedings of the Workshop and Shared Tasks*, pp. 155–166, 3 2024. [Online]. Available: https://arxiv.org/pdf/2403.05881

[49] F. Fu *et al.*, "Synergizing knowledge graph and large language model for relay catalysis pathway recommendation," *National Science Review*, p. 271, 7 2025. [Online]. Available: https://dx.doi.org/10.1093/nsr/nwaf271

[50] Q. Ma, Y. Zhou, and J. Li, "Automated Retrosynthesis Planning of Macromolecules Using Large Language Models and Knowledge Graphs," *Macromolecular Rapid Communications*, 4 2025. [Online]. Available: http://arxiv.org/abs/2501.08897http://dx.doi.org/10.1002/marc.202500065

[51] J. Wei *et al.*, "Chain-of-Thought Prompting Elicits Reasoning in Large Language Models," *Advances in Neural Information Processing Systems*, vol. 35, 1 2022. [Online]. Available: https://arxiv.org/abs/2201.11903v6

[52] S. Klamt, U. U. Haus, and F. Theis, "Hypergraphs and Cellular Networks," *PLoS Computational Biology*, vol. 5, no. 5, p. e1000385, 2009. [Online]. Available: https://pmc.ncbi.nlm.nih.gov/articles/PMC2673028/

[53] S. Müller, C. Flamm, and P. F. Stadler, "What makes a reaction network "chemical"?" *Journal of Cheminformatics*, vol. 14, no. 1, pp. 1–24, 12 2022. [Online]. Available: https://jcheminf.biomedcentral.com/articles/10.1186/s13321-022-00621-8

[54] A. Garcia-Chung, M. Bermúdez-Montaña, P. F. Stadler, J. Jost, and G. Restrepo, "Chemically inspired Erdős–Rényi hypergraphs," *Journal of Mathematical Chemistry*, vol. 62, no. 6, pp. 1357–1383, 7 2024. [Online]. Available: https://link.springer.com/article/10.1007/s10910-024-01595-8

[55] T. Gaudelet, N. Malod-Dognin, and N. Pržulj, "Higher-order molecular organization as a source of biological function," *Bioinformatics*, vol. 34, no. 17, pp. i944–i953, 9 2018. [Online]. Available: https://dx.doi.org/10.1093/bioinformatics/bty570

[56] A. Gogolińska and W. Nowak, "Bipartite Graphs—Petri Nets in Biology Modeling," *Mechanisms and Machine Science*, vol. 107, pp. 175–200, 2022. [Online]. Available: https://link.springer.com/chapter/10.1007/978-3-030-76787-7_9

[57] Y. Jang, J. Kim, and S. Ahn, "Improving Chemical Understanding of LLMs via SMILES Parsing," 5 2025. [Online]. Available: https://arxiv.org/pdf/2505.16340

[58] "OpenAI Prompt Engineering Guide." [Online]. Available: https://platform.openai.com/docs/guides/prompt-engineering/strategy-write-clear-instructions

[59] K. Papineni, S. Roukos, T. Ward, and W.-J. Zhu, "BLEU," *Proceedings of the 40th Annual Meeting on Association for Computational Linguistics - ACL '02*, p. 311, 2001. [Online]. Available: https://dl.acm.org/doi/pdf/10.3115/1073083.1073135

[60] D. Jurafsky and J. H. Martin, *Speech and Language Processing: An Introduction to Natural Language Processing, Computational Linguistics, and Speech Recognition with Language Models*, 3rd ed., 2025. [Online]. Available: https://web.stanford.edu/~jurafsky/slp3/

[61] S. Banerjee and A. Lavie, "METEOR: An Automatic Metric for MT Evaluation with Improved Correlation with Human Judgments," pp. 65–72, 2005. [Online]. Available: https://aclanthology.org/W05-0909/
[62] C.-Y. Lin, "ROUGE: A Package for Automatic Evaluation of Summaries," pp. 74–81, 2004. [Online]. Available: https://aclanthology.org/W04-1013/
[63] OpenAI, "Gpt-4.1-mini," https://platform.openai.com/docs/models#gpt-4-1-mini, 2025, model release, April 2025.
[64] Sentence-Transformers, "all-mpnet-base-v2," https://huggingface.co/sentence-transformers/all-mpnet-base-v2, 2021, hugging Face model card; accessed 2025-09-02.
[65] M. F. Porter, "An algorithm for suffix stripping," 1980.
[66] 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.
[67] S. Chang and E. Fosler-Lussier, "How to Prompt LLMs for Text-to-SQL: A Study in Zero-shot, Single-domain, and Cross-domain Settings," 5 2023. [Online]. Available: https://arxiv.org/pdf/2305.11853
[68] J. D. . Abramo, A. Zugarini, and P. Torroni, "Dynamic Few-Shot Learning for Knowledge Graph Question Answering," 7 2024. [Online]. Available: https://arxiv.org/pdf/2407.01409
[69] K. Gupta *et al.*, "How Robust are LLMs to In-Context Majority Label Bias?" *arXiv preprint arXiv:2312.16549*, 12 2023. [Online]. Available: https://arxiv.org/pdf/2312.16549
[70] L. Yoshida, "The Impact of Example Selection in Few-Shot Prompting on Automated Essay Scoring Using GPT Models," *Communications in Computer and Information Science*, vol. 2150 CCIS, pp. 61–73, 11 2024. [Online]. Available: http://arxiv.org/abs/2411.18924http://dx.doi.org/10.1007/978-3-031-64315-6_5
[71] A. Ajith, C. Pan, M. Xia, A. Deshpande, and K. Narasimhan, "InstructEval: Systematic Evaluation of Instruction Selection Methods," *Findings of the Association for Computational Linguistics: NAACL 2024 - Findings*, pp. 4336–4350, 7 2023. [Online]. Available: https://arxiv.org/pdf/2307.00259
[72] "Cypher Manual." [Online]. Available: https://neo4j.com/docs/cypher-manual/
[73] Z. Yang *et al.*, "Can Large Language Models Always Solve Easy Problems if They Can Solve Harder Ones?" *EMNLP 2024 - 2024 Conference on Empirical Methods in Natural Language Processing, Proceedings of the Conference*, pp. 1531–1555, 6 2024. [Online]. Available: https://arxiv.org/pdf/2406.12809

SUPPLEMENTARY INFORMATION

P1: Single-Step Synthesis Prompt – Base Contextual
Model: GPT-4.1-mini, T=0.0

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
Model: GPT-4.1-mini, T=0.0

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
Model: GPT-4.1-mini, T=0.0

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
Model: GPT-4.1-mini, T=0.0

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
Model: GPT-4.1-mini, T=0.0

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
Model: GPT-4.1-mini, T=0.0

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
Model: GPT-4.1-mini, T=0.0

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., $2 \rightarrow 4$, $3 \rightarrow 6$, $4 \rightarrow 8$).

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
Model: GPT-4.1-mini, T=0.0

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., $2 \to 4$, $3 \to 6$, $4 \to 8$).

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
Model: GPT-4.1-mini, T=0.0

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., $2 \to 4$, $3 \to 6$, $4 \to 8$).

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
Model: GPT-4.1-mini, T=0.0

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., $2 \to 4$, $3 \to 6$, $4 \to 8$).

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 `RETURN`s.
User Question

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

P11: Single-Step Synthesis Validator Prompt
Model: GPT-4.1-mini, T=0.0

You are a Cypher expert validating queries over a **bipartite** chemical reaction knowledge graph.
Your job is to check whether the query is syntactically valid, uses only schema-defined elements, retrieves the *complete reaction context*, and is semantically appropriate for the question.
You must check the following:

- Are there any syntax errors?
- Are all node labels, relationship types, and properties used in the query present in the schema?
- Does the query return the *full reaction context* — i.e., all `:Molecule` nodes connected to each `:Reaction` via `REACTS_IN`, `PRODUCES`, `USES_AGENT`, and `USES_SOLVENT` — regardless of the user's intent?
- Are variables reused incorrectly in a way that limits matching all relevant nodes (e.g., reusing the same variable for a single product and for a collection of all products)?

Schema

```
{schema}
```

User Question

```
{question}
```

Cypher

```
{cypher}
```

Output format (JSON only; no extra text, no code fences)

```
{"is_valid": true/false, "errors": ["<string>", "..."]}
```

Example

```
<question>
What is the reaction that produces X?
</question>
<cypher>
MATCH (r:Reaction)-[:PRODUCES]->(target_product:Molecule {name: $X})
RETURN
  r.id AS reaction_id,
  target_product.name AS product;
</cypher>
<output>
{
  "is_valid": false,
  "errors": [
    "The query does not return full reaction context. It omits reactants, agents, solvents,
    ↪  and other products connected to the same reaction."
  ]
}
</output>
```

P12: Multi-Step Synthesis Validator Prompt
Model: GPT-4.1-mini, T=0.0

You are a Cypher expert validating queries over a **bipartite** chemical reaction knowledge graph. Your job is to check whether the query is syntactically valid, uses only schema-defined elements, retrieves complete reaction context, and is semantically appropriate for the question.
Definitions

- `<pathVar>` = the variable bound to the variable-length traversal in `MATCH` (e.g., `MATCH p = (...)`). If multiple, use the one connecting start/target.

Checks

- **Context (reaction-only output):**
  - Do the `WITH/RETURN` clauses project only `:Reaction` nodes (e.g., `WITH [n IN nodes(<pathVar>) WHERE n:Reaction] AS reaction_nodes RETURN DISTINCT reaction_nodes`)?
  - They must *not* project molecule payloads (e.g., `intermediate_molecules`) or any `:Molecule` properties.

- **Step–hop mapping (only if the question specifies exactly K steps):**
  - Extract K from the question. The check **passes** if *any* of the following is true:
    * `size(relationships(<pathVar>)) = 2*K`, **or**
    * the pattern uses exact length `*{2*K}`.
  - Do *not* also complain about an upper bound like `*..(2*K)` if equality is already asserted.
  - Additionally require strict Molecule↔Reaction alternation *and* `size([n IN nodes(<pathVar>) WHERE n:Reaction]) = K`.

Schema

`{schema}`

User Question

`{question}`

Cypher

`{cypher}`

Return format (strict JSON only; no extra text)

```
{"is_valid": true/false, "errors": ["<string>", "..."]}
```

Example

```
<question>
What are the intermediate molecules involved in synthesizing {{TARGET_SMILES}} in exactly 4
↪  reaction steps?
</question>

<cypher>
WITH '{{TARGET_SMILES}}' AS targetName
MATCH path = (start:Molecule)-[:REACTS_IN|PRODUCES*..8]->(target:Molecule {name: targetName})
...
WITH [n IN nodes(path) WHERE n:Molecule AND n.name <> targetName AND n.name <> start.name] AS
↪  intermediate_molecules
RETURN DISTINCT intermediate_molecules;  // NOT reaction-only
</cypher>

<output>
{
  "is_valid": false,
  "errors": [
    "Reaction-only output required: the query projects :Molecule nodes via
    ↪  `intermediate_molecules`, which is NOT correct."
  ]
}
</output>
```

P13: Single-Step Synthesis Corrector Prompt
Model: GPT-4.1-mini, T=0.0

You are a Cypher expert assisting a junior developer working with a chemical reaction knowledge graph. Your job is to correct Cypher queries based on provided error messages and the schema structure.

Output contract

- Return *only* the corrected Cypher query.
- Wrap the entire query in ``` triple backticks.
- Do not include any other text.

Check for invalid syntax or semantics and return a corrected Cypher statement.

Schema

`{schema}`

Ensure the corrected query returns *full reaction context* (reactants, products, agents, solvents) when `:Reaction` nodes are queried.

Important: Respond *ONLY* with the corrected Cypher query, wrapped in triple backticks. Do not include any explanations or extra text.

User Question
`{question}`
Original Cypher
`{cypher}`
Errors
`{errors}`

P14: Multi-Step Synthesis Corrector Prompt
Model: GPT-4.1-mini, T=0.0

You are a Cypher expert assisting a junior developer working with a chemical reaction knowledge graph. Your job is to correct Cypher queries based on provided error messages and the schema structure.
Output contract

- Return *only* the corrected Cypher query.
- Wrap the entire query in ``` triple backticks.
- Do not include any other text.

Check for invalid syntax or semantics and return a corrected Cypher statement.
Check for invalid syntax or semantics and return a corrected Cypher statement.
Schema
`{schema}`
Edit policy

- Fix *ONLY* what `<errors>` describe. Make the smallest edits that fully resolve them. Do not add new logic.

Reaction-only output contract

- The final projection **MUST** be a single column named `reaction_nodes` defined from the path variable:
  ```
  WITH [x IN nodes(<pathVar>) WHERE x:Reaction] AS reaction_nodes
  RETURN reaction_nodes
  ```
- Do **NOT** use `UNWIND` anywhere. If present, remove it and return the list directly.
- Do **NOT** project `:Molecule` nodes or any molecule properties.
- Preserve the existing path variable from `MATCH` (e.g., `p`); replace `<pathVar>` with it.

User Question
`{question}`
Original Cypher
`{cypher}`
Errors
`{errors}`

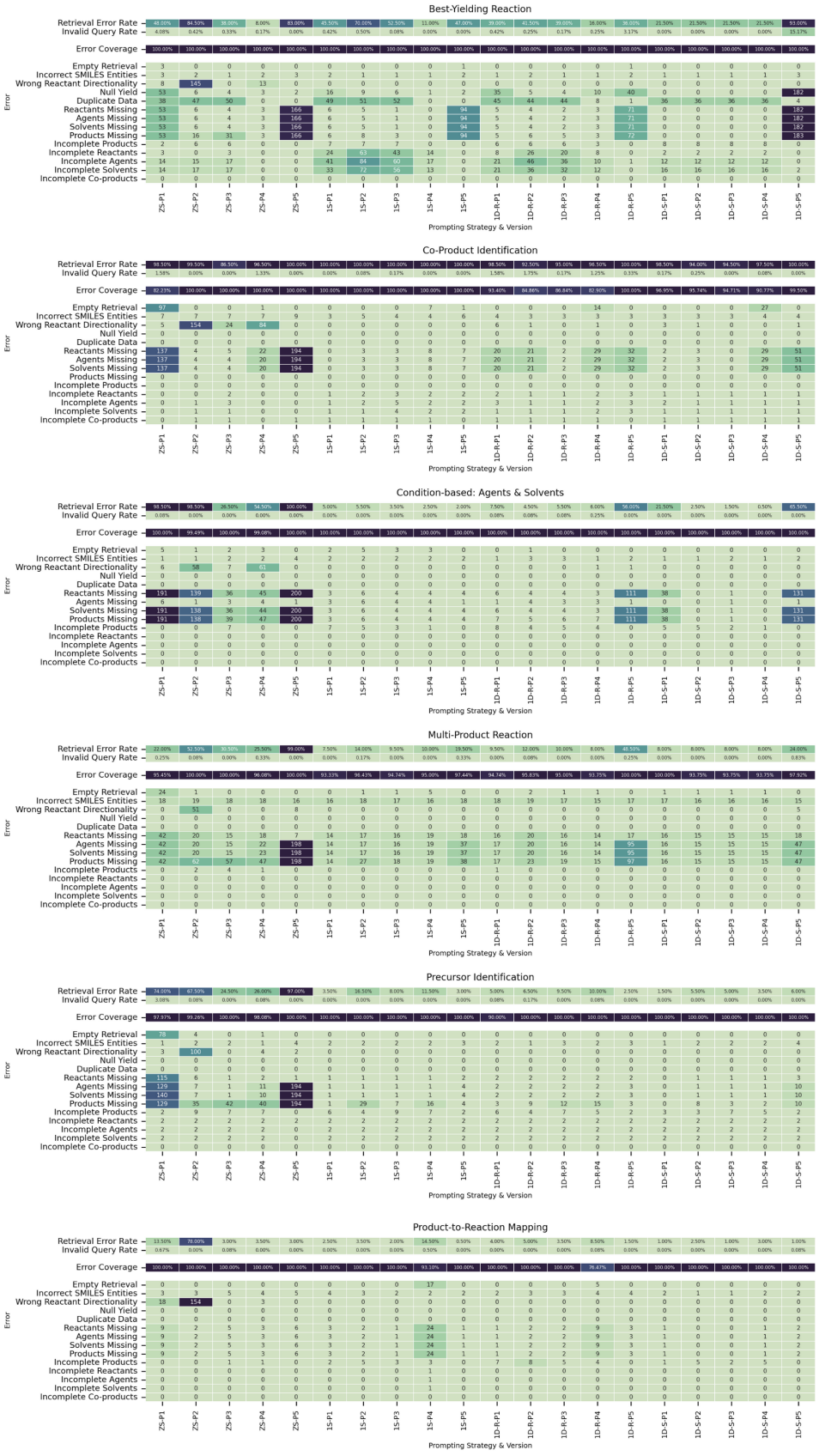


Fig. S1: Distribution of Retrieval Errors in Single-Step Reaction Retrieval. Retrieval error = proportion of samples with a non-perfect F1 score. Query validity = proportion of valid queries relative to all samples. Error coverage = percentage of samples containing one of the the errors reported. Below are the main categories of retrieval errors 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)) and five prompt versions of varying contextual and structural guidance: (1) – base contextual query, (2) – (1) + explicit MATCH instructions, (3) – (2) + directionality check, (4) – (3) + yield-filtering, and (5) – (4) + additional full context rules. Each cell shows the number of results containing the given error out of 1200 data samples in total.

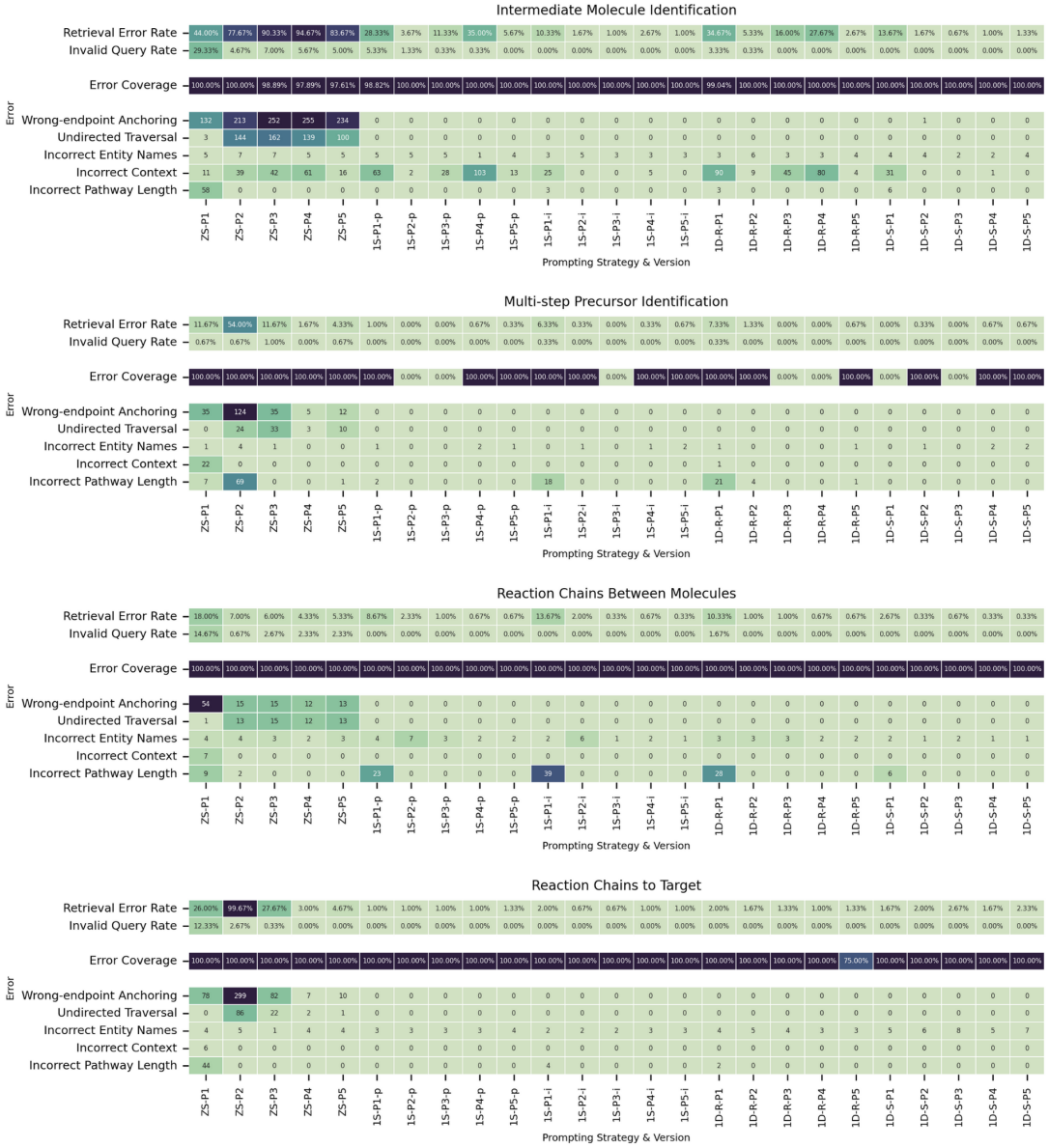


Fig. S2: Distribution of Retrieval Errors in Multi-Step Reaction Retrieval. Retrieval error = proportion of samples with a non-perfect F1 score. Query validity = proportion of valid queries relative to all samples. Error coverage = percentage of samples containing one of the the errors reported. Below are the main categories of retrieval errors across four prompting strategies (zero-shot (ZS), one-shot static (1S-S, which was tested using two different fixed examples: 1S-S-p = product identification and 1S-S-i=forward synthesis intermediate molecule discovery), one-shot random (1S-D-R), and one-shot semantic (1S-D-S)). Five prompt versions of varying contextual and structural guidance were tested: (1) – minimal prompt, (2) – (1) + instructional with partial Cypher snippets, (3) – (2) + illustrative with full Cypher Snippets, (4) – (3) + SMILES verbatim, and (5) – (4) + strict RETURN constraint. Each cell shows the number of results containing the given error out of 1200 data samples in total.

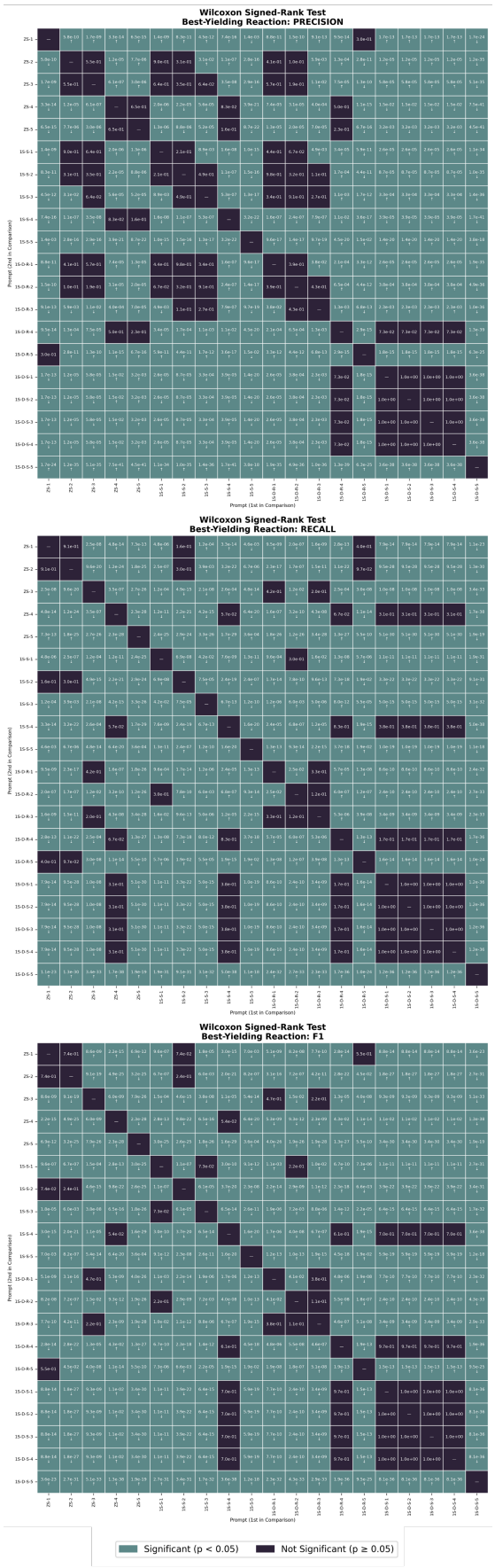


Fig. S3: Pairwise statistical significance tests for the *Best-yielding reaction* task. Each heatmap compares two prompt configurations (x-axis: first prompt, y-axis: second prompt). Cells show p-values from Wilcoxon Signed-Rank tests for Precision, Recall, and F1. Arrows within the cells indicate whether the first prompt (x-axis) significantly improves over or performs worse than the second prompt (y-axis).

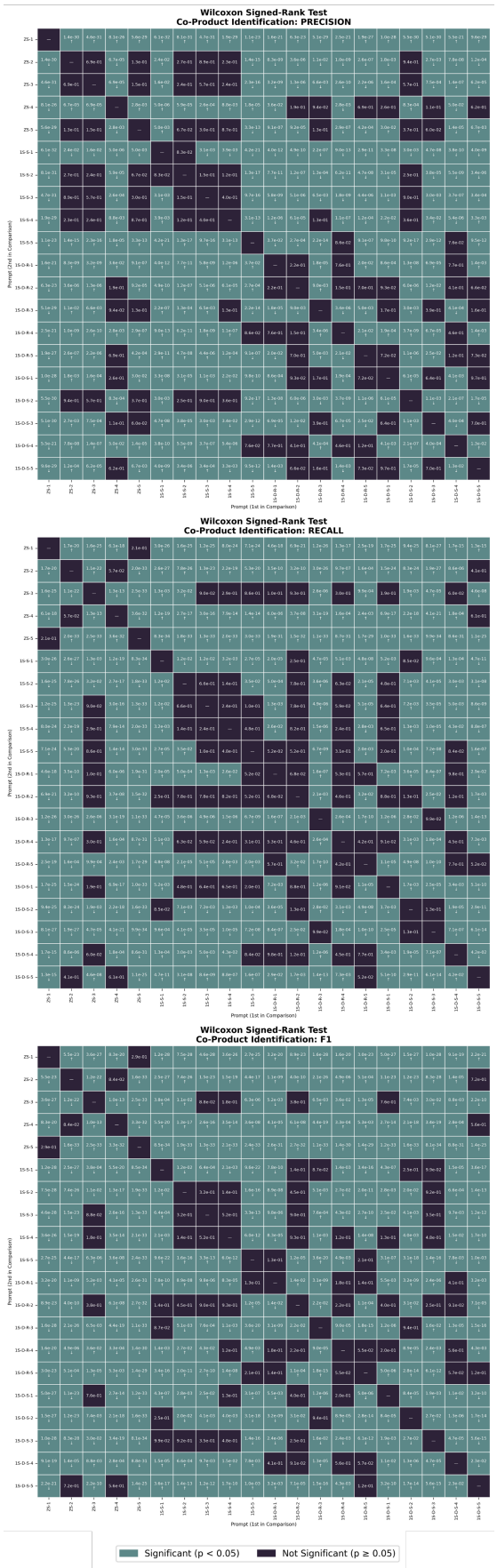


Fig. S4: Pairwise statistical significance tests for the *Co-Product Identification* task. Each heatmap compares two prompt configurations (x-axis: first prompt, y-axis: second prompt). Cells show p-values from Wilcoxon Signed-Rank tests for Precision, Recall, and F1. Arrows within the cells indicate whether the first prompt (x-axis) significantly improves over or performs worse than the second prompt (y-axis).

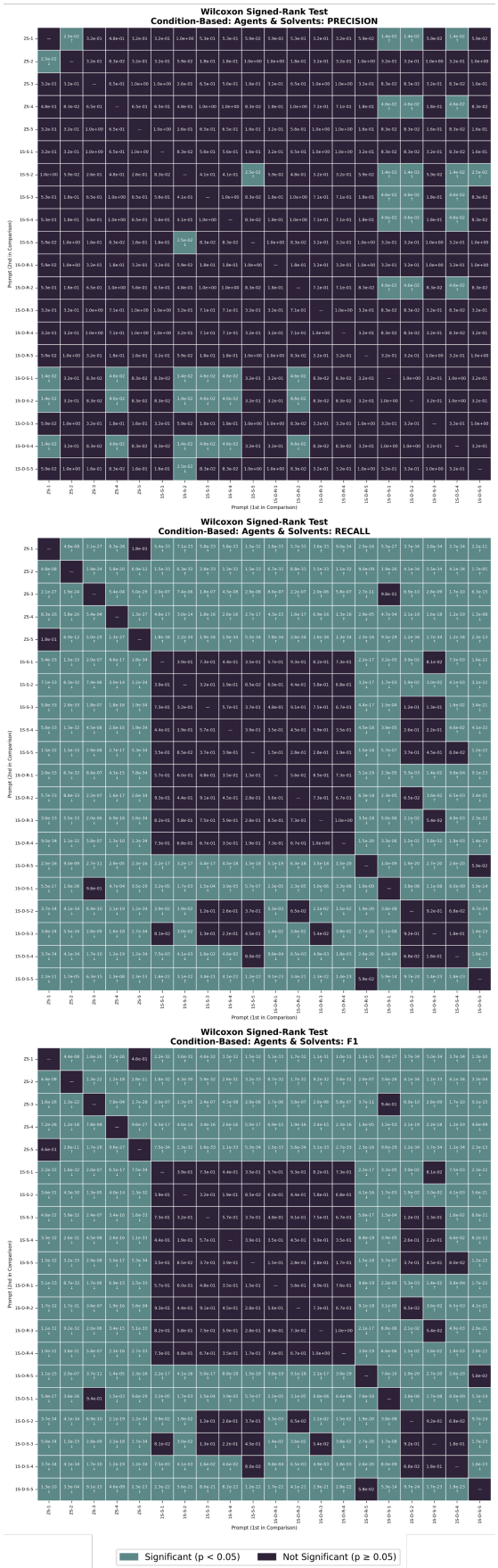


Fig. S5: Pairwise statistical significance tests for the *Condition-Based: Agents& Solvents* task. Each heatmap compares two prompt configurations (x-axis: first prompt, y-axis: second prompt). Cells show p-values from Wilcoxon Signed-Rank tests for Precision, Recall, and F1. Arrows within the cells indicate whether the first prompt (x-axis) significantly improves over or performs worse than the second prompt (y-axis).

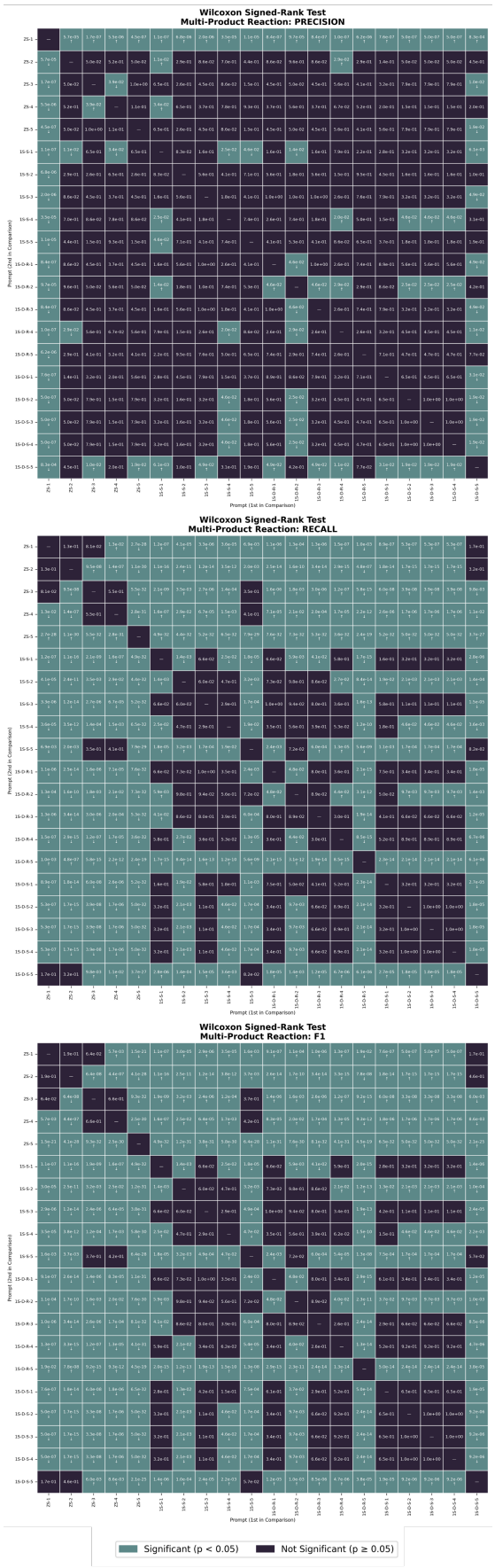


Fig. S6: Pairwise statistical significance tests for the *Multi-Product Reaction* task. Each heatmap compares two prompt configurations (x-axis: first prompt, y-axis: second prompt). Cells show p-values from Wilcoxon Signed-Rank tests for Precision, Recall, and F1. Arrows within the cells indicate whether the first prompt (x-axis) significantly improves over or performs worse than the second prompt (y-axis).

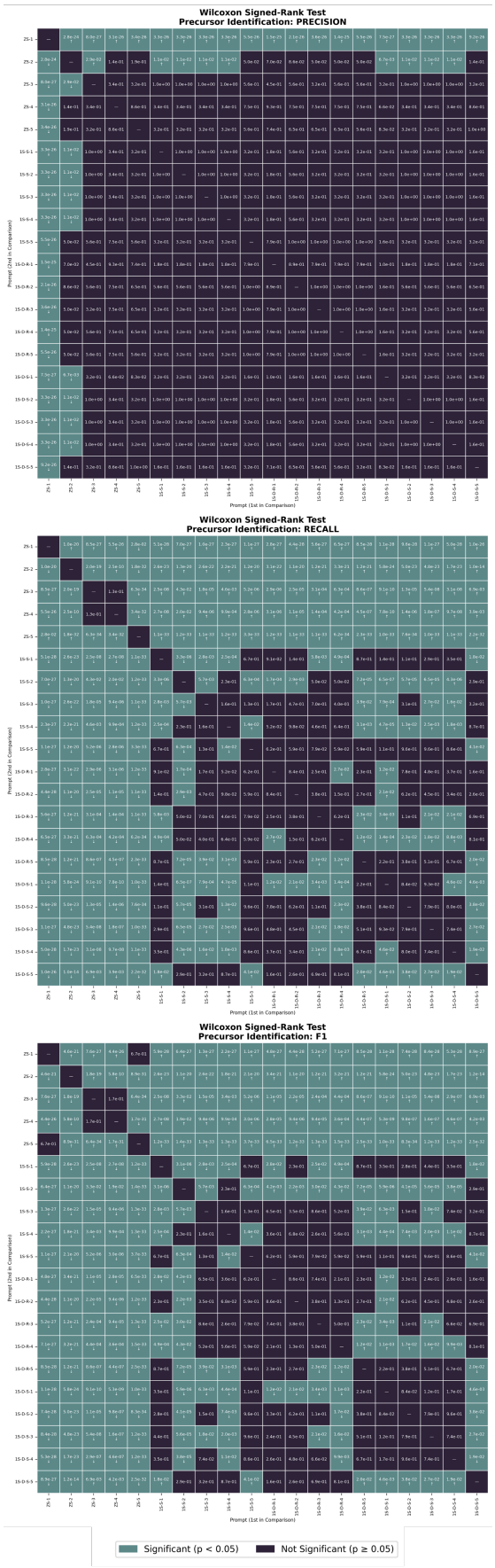


Fig. S7: Pairwise statistical significance tests for the *Precursor Identification* task. Each heatmap compares two prompt configurations (x-axis: first prompt, y-axis: second prompt). Cells show p-values from Wilcoxon Signed-Rank tests for Precision, Recall, and F1. Arrows within the cells indicate whether the first prompt (x-axis) significantly improves over or performs worse than the second prompt (y-axis).

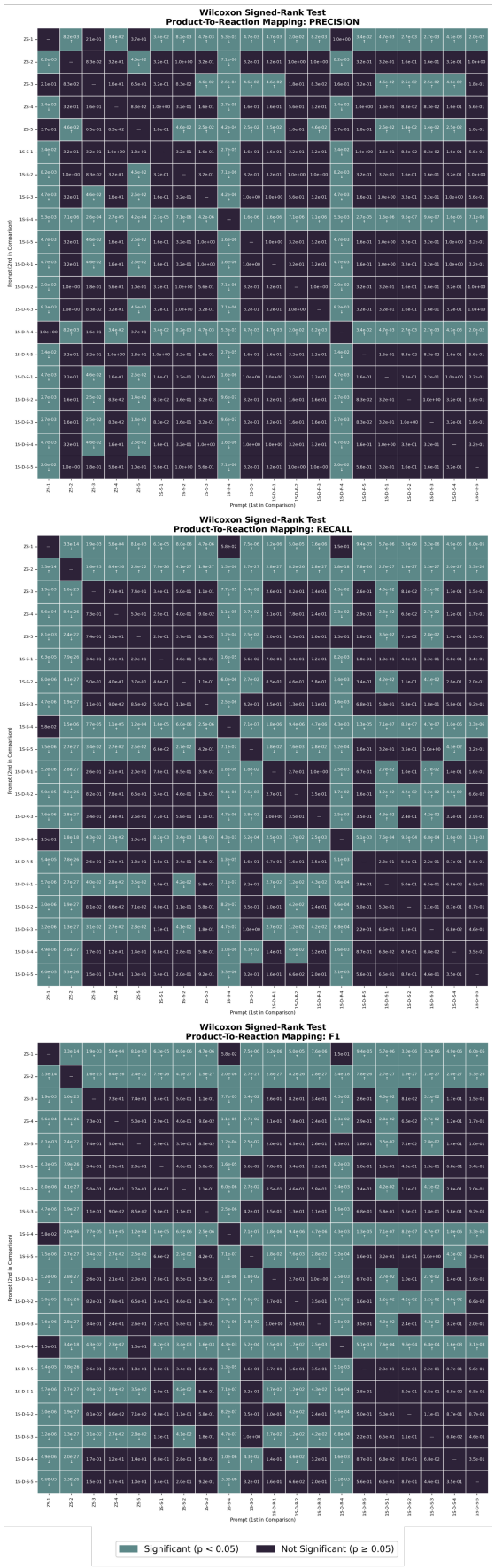


Fig. S8: Pairwise statistical significance tests for the *Product-to-Reaction Mapping* task. Each heatmap compares two prompt configurations (x-axis: first prompt, y-axis: second prompt). Cells show p-values from Wilcoxon Signed-Rank tests for Precision, Recall, and F1. Arrows within the cells indicate whether the first prompt (x-axis) significantly improves over or performs worse than the second prompt (y-axis).

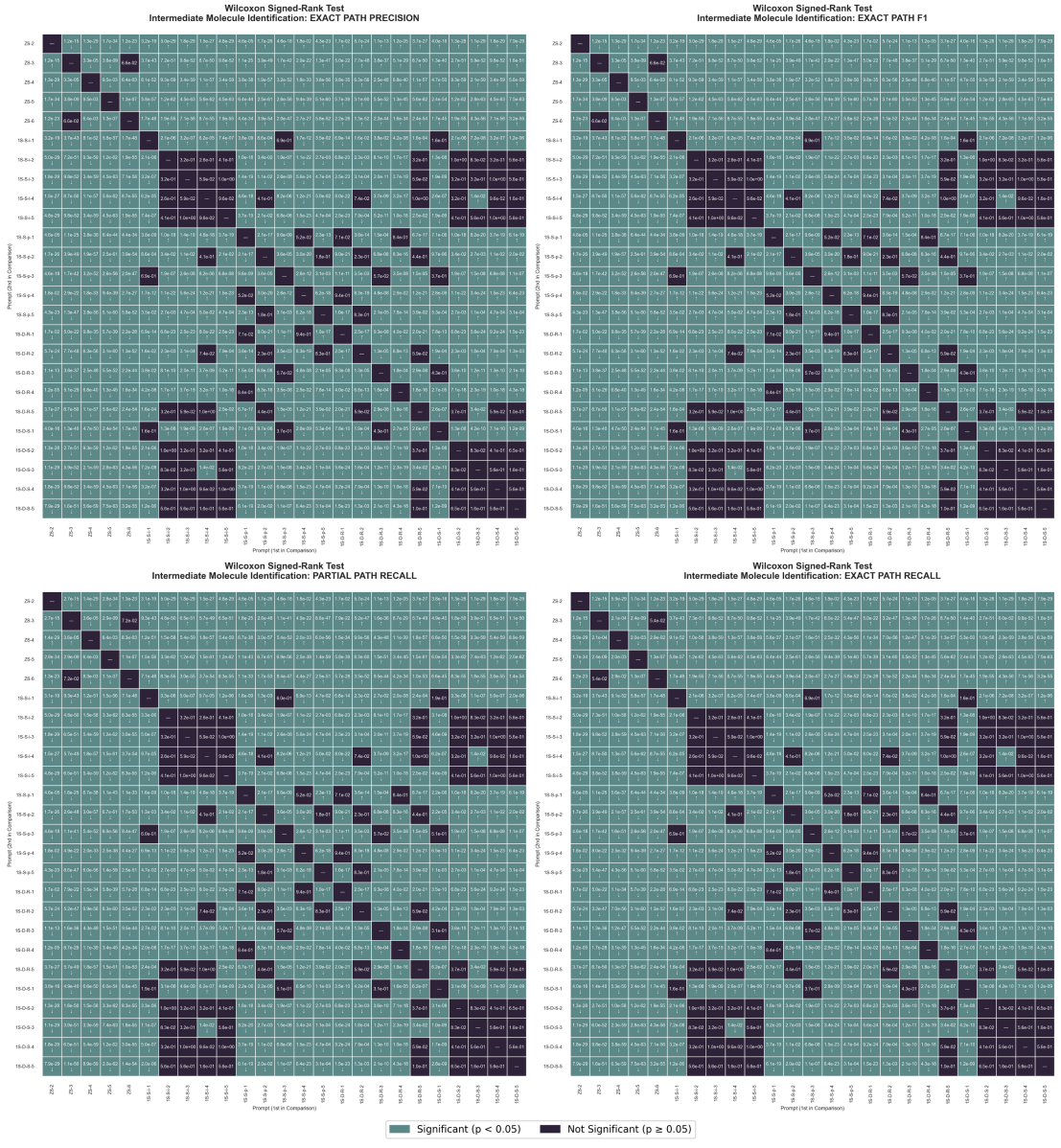


Fig. S9: Pairwise statistical significance tests for the *Intermediate Identification* task. Each heatmap compares two prompt configurations (x-axis: first prompt, y-axis: second prompt). Cells show p-values from Wilcoxon Signed-Rank tests for Exact Path Precision, Exact Path Recall, Partial Path Recall and Exact Path F1. Arrows within the cells indicate whether the first prompt (x-axis) significantly improves over or performs worse than the second prompt (y-axis).

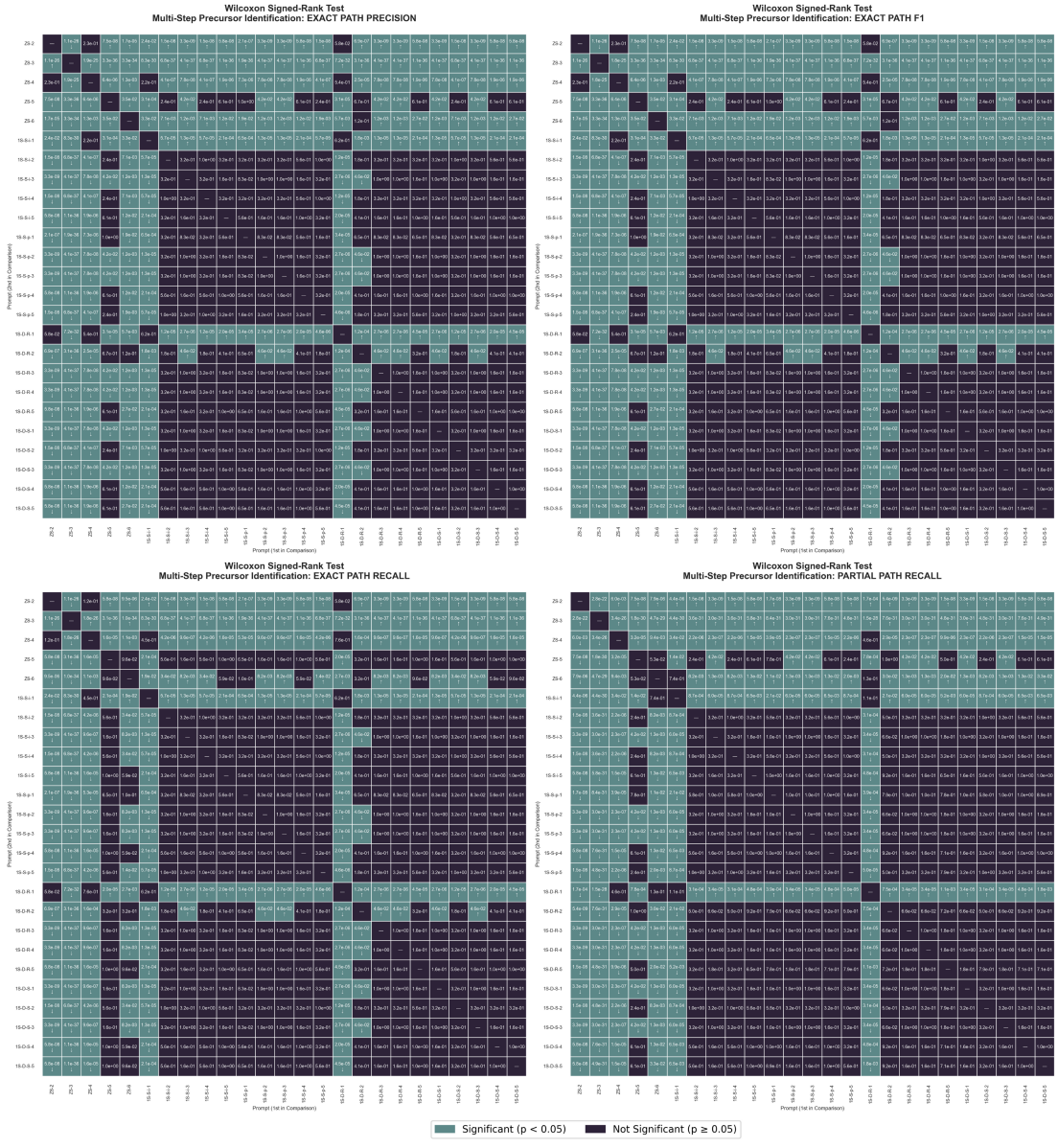


Fig. S10: Pairwise statistical significance tests for the *Multi-Step Precursor Identification* task. Each heatmap compares two prompt configurations (x-axis: first prompt, y-axis: second prompt). Cells show p-values from Wilcoxon Signed-Rank tests for Exact Path Precision, Exact Path Recall, Partial Path Recall and Exact Path F1. Arrows within the cells indicate whether the first prompt (x-axis) significantly improves over or performs worse than the second prompt (y-axis).

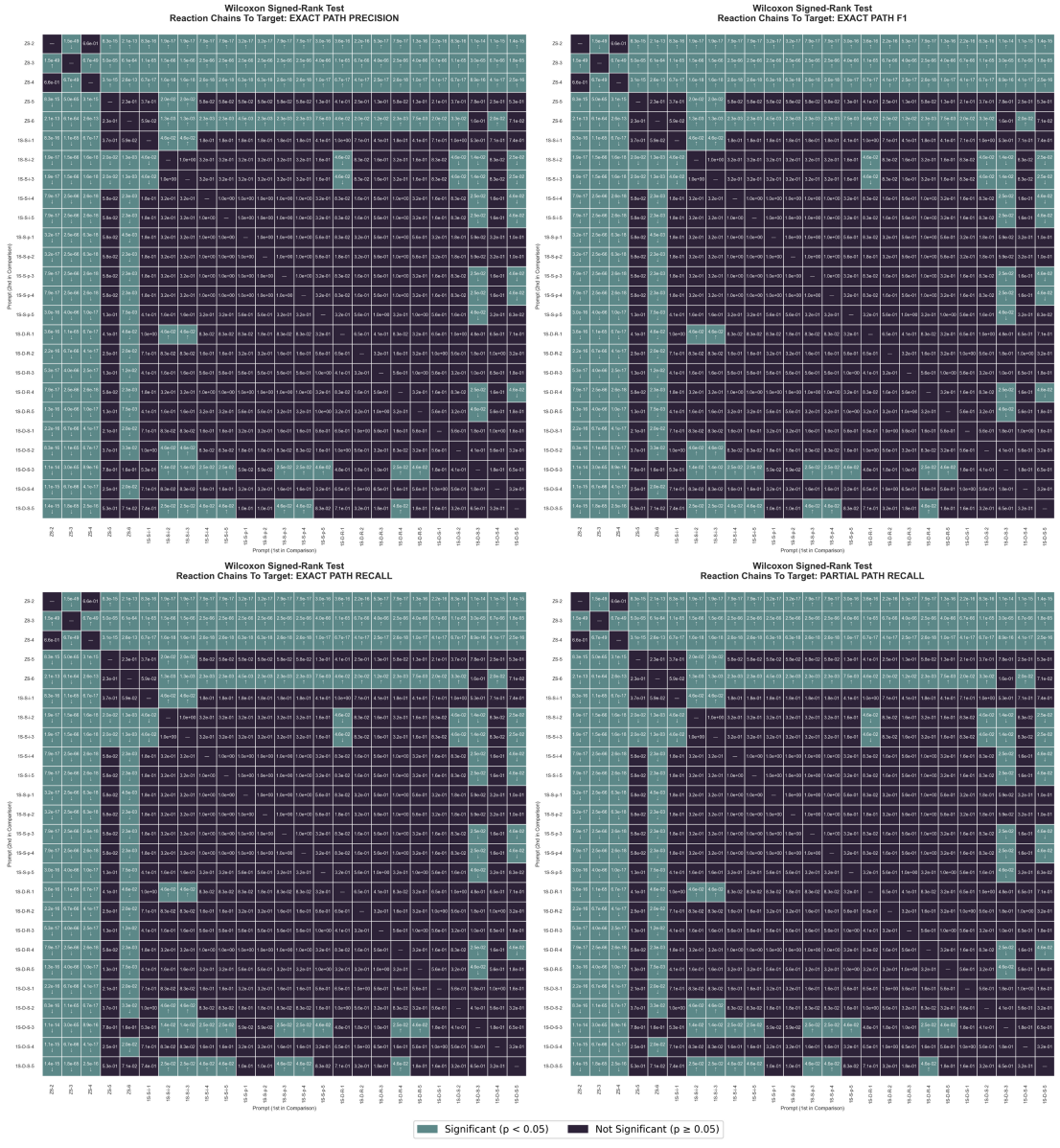


Fig. S11: Pairwise statistical significance tests for the *Reaction Chains Between Molecules* task. Each heatmap compares two prompt configurations (x-axis: first prompt, y-axis: second prompt). Cells show p-values from Wilcoxon Signed-Rank tests for Exact Path Precision, Exact Path Recall, Partial Path Recall and Exact Path F1. Arrows within the cells indicate whether the first prompt (x-axis) significantly improves over or performs worse than the second prompt (y-axis).

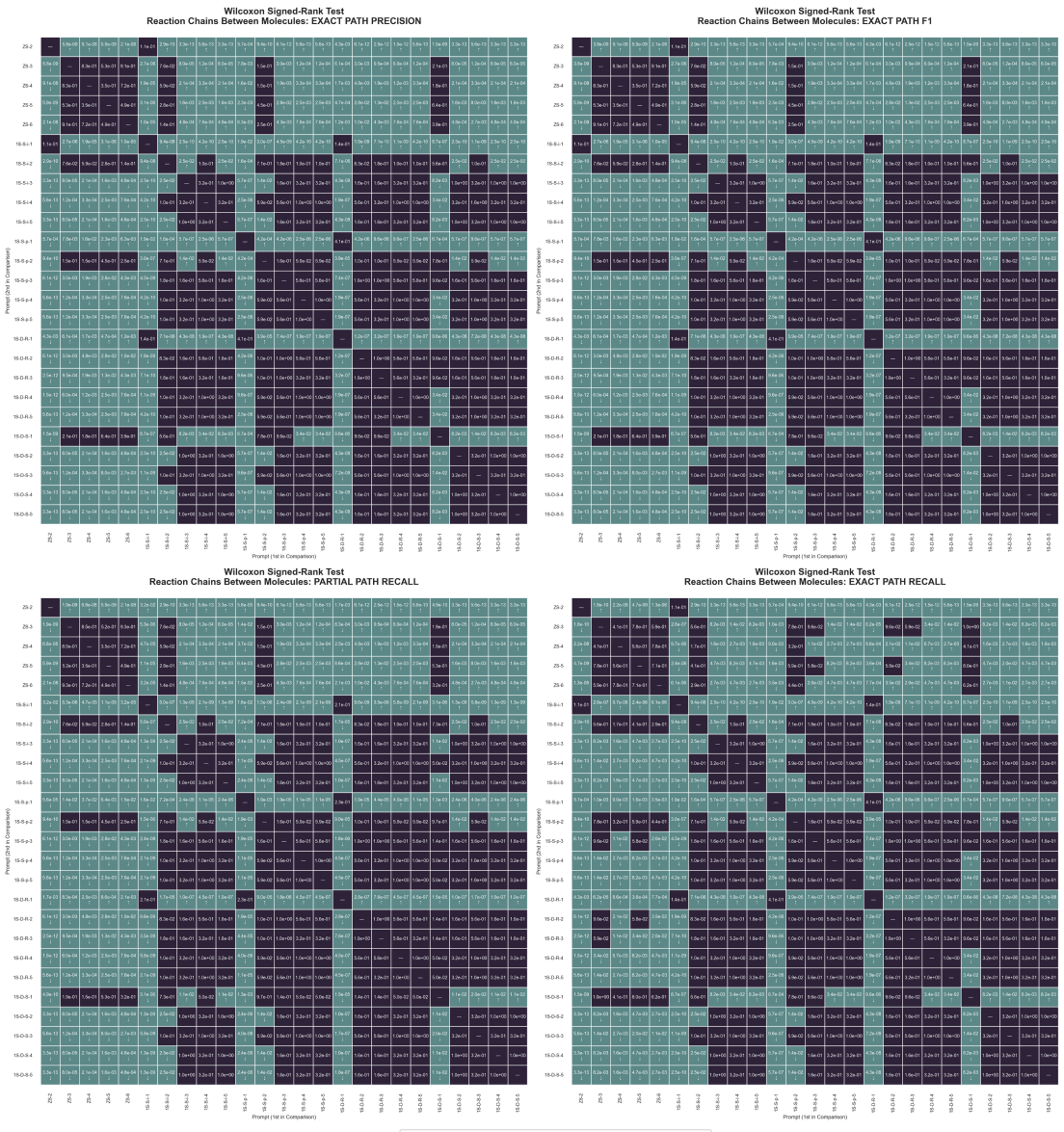


Fig. S12: Pairwise statistical significance tests for the *Reaction Chains To Target* task. Each heatmap compares two prompt configurations (x-axis: first prompt, y-axis: second prompt). Cells show p-values from Wilcoxon Signed-Rank tests for Exact Path Precision, Exact Path Recall, Partial Path Recall and Exact Path F1. Arrows within the cells indicate whether the first prompt (x-axis) significantly improves over or performs worse than the second prompt (y-axis).