# Building MCP Servers for Agentic Retrieval over Knowledge Graphs

Fassio Simone
EURECOM
Biot, France
Simone.Fassio@eurecom.fr

Gosmar Dario
EURECOM
Biot, France
Dario.Gosmar@eurecom.fr

## Abstract

*Modern AI agents require standardized access to external structured knowledge to perform complex reasoning tasks; however, enabling Large Language Models (LLMs) to reliably query Knowledge Graphs (KGs) remains a significant challenge due to the complexity of specific ontologies and the prevalence of hallucinations [7]. This project proposes the design and deployment of a Model Context Protocol (MCP) server dedicated to agentic retrieval over the DOREMUS KG. By adopting a Hybrid Agentic framework, the agent can dynamically perform "schema pruning" and "neighborhood retrieval" through the application of parameterized patterns, linking entities and properties to the user's specific intent step-by-step. The project evaluates this architecture by testing the ability of diverse agent clients to answer competency questions, orchestrate multi-step retrieval tasks, and generate explanations grounded in RDF results. Our code is available at https://github.com/SimoneFassio/DOREMUS_MCP*

## 1. Introduction

The integration of Large Language Models (LLMs) with structured Knowledge Graphs (KGs) represents a critical frontier in artificial intelligence, promising systems that combine the reasoning capabilities of generative models with the factual precision of graph databases. However, enabling LLMs to reliably translate Natural Language Questions (NLQs) into executable graph queries (e.g., SPARQL) remains a significant challenge. A primary obstacle is the model's lack of familiarity with specific, evolving ontologies; providing a full schema in the context window often overwhelms the model with irrelevant information, while zero-shot prompting frequently leads to hallucinations of non-existent properties or relationships.

Historically, approaches to this problem have relied on fine-tuning models on domain-specific datasets [7]. While effective, fine-tuning is architecturally "brittle"; it requires expensive retraining whenever the schema evolves and creates "data bottlenecks" due to the scarcity of high-quality training pairs. Consequently, the field has shifted toward **Schema-Retrieval-Augmented Generation (Schema-RAG)**. Unlike traditional RAG, which retrieves data instances, Schema-RAG retrieves a filtered subset of the graph's metadata—essentially providing the LLM with a dynamic "blueprint" required to construct a valid query.

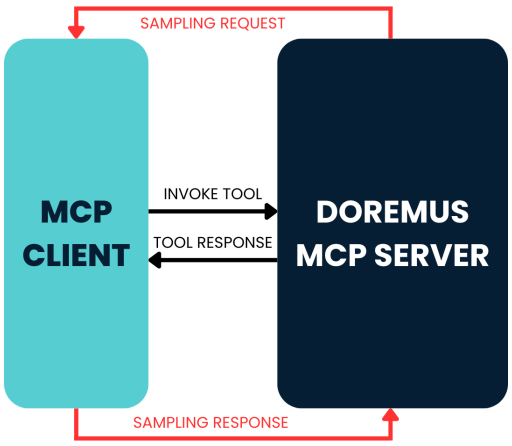


Figure 1. **MCP-based system for knowledge retrieval:** The DOREMUS MCP-based architecture, illustrating the bidirectional interaction loop: the MCP Client executes specialized tools for graph traversal, while the Server utilizes Sampling Requests to leverage the model's reasoning for intent-driven pathfinding.

While Schema-RAG offers a scalable alternative to fine-tuning, static retrieval methods remain vulnerable to ambiguity; if the initial retrieval step fails to identify the correct property path, the generation fails. To address this, this project implements a **Hybrid Agentic Framework**, combining low-granularity parameterized tools with LLM-driven reasoning, architected over the **Model Context Protocol (MCP)** [3]. By leveraging MCP's standardized primitives—specifically executable *Tools* for graph traversal and *Sampling* to resolve semantic ambiguities—the sys-

tem transforms the passive retrieval process into a dynamic workflow. This approach mirrors recent "iterative" methodologies (such as ARUQULA [1]), where the system does not simply retrieve a schema, but actively invokes low-granularity tools to explore the graph structure and validate assumptions step-by-step before query generation.

This project focuses on the design and deployment of MCP servers specifically optimized for agentic retrieval over the **DOREMUS** (music) Knowledge Graph [4]. Our architecture implements a "*Query Container*" mechanism that constructs SPARQL queries incrementally using modular tools. By embedding "neighborhood retrieval" inside iterative pattern-creating tools and performing "schema pruning" through execution-based validation, the system dynamically extracts only the subgraph relevant to the user's specific intent. Furthermore, we introduce an **execution feedback loop**, allowing the agent to validate query syntax and logic against the endpoint in real-time, thereby ensuring high reliability and the ability to self-correct hallucinations without human intervention [8].

## 2. Related Works

The challenge of translating Natural Language Questions (NLQs) into executable SPARQL queries has evolved from template-based and fine-tuning approaches toward Agentic Retrieval-Augmented Generation (RAG). In this section, we analyze two distinct categories of methodologies: established agentic frameworks for Text-to-SPARQL and the emerging landscape of Model Context Protocol (MCP) servers.

### 2.1. Agentic Frameworks for Text-to-SPARQL

Recent literature suggests that single-shot generation of SPARQL queries is insufficient for complex Knowledge Graphs (KGs) due to the "*Context Window*" constraint; providing an LLM with a complete schema often results in information overload and increased hallucination rates [7]. To address this, the field has converged on algorithmic strategies to isolate relevant structural elements: "**Schema Pruning**"—the selection of a minimal subset of the ontology to reduce noise [6]—and "**Neighborhood Retrieval**", which identifies "*seed*" entities in a user's request to retrieve only their immediate topological connections [6]. While static RAG methods attempt to pre-compute these steps, modern **Agentic Frameworks** transform this into an iterative, dynamic workflow, employing "*tools*" to explore the graph structure dynamically.

#### 2.1.1. Iterative Exploration and Enhancement

Recent research demonstrates that naive retrieval of schema elements often fails without a validation loop, necessitating a shift from single-shot generation to iterative refinement. Piao et al. [11] propose a framework for exploring KGs in a zero-shot setting, employing a cyclic workflow where a "*Query Enhancer*" validates extracted predicates against the schema and refines the generated SPARQL based on execution feedback [11]. Taking this agentic concept further, the mKGQAgent framework introduces a *human-inspired workflow* that decomposes the translation task into modular subtasks—such as Entity Linking and Query Refinement—rather than trying to achieve monolithic translation [10]. These methodologies collectively confirm that treating query generation as a multi-step, modular refinement process significantly outperforms static generation methods, particularly in reducing hallucinations and handling complex multilingual settings [10].

#### 2.1.2. ReAct and Graph Exploration Utilities

A more formalized agentic approach is presented in ARUQULA, which generalizes the SPINACH framework for RDF graphs [1]. Utilizing the ReAct (Reason and Act) paradigm, ARUQULA moves beyond simple retrieval by implementing a rigorous **feedback-driven validation loop**. The agent is equipped with exploration tools (e.g., `search_entity`, `get_property_examples`) to iteratively probe the schema, confirming that standard retrieval is insufficient for complex graphs. Crucially, ARUQULA identifies the need for **semantic grounding**—the mapping of ambiguous natural language terms to precise graph IRIs—as a distinct architectural phase. The framework implements a dedicated strategy to resolve ambiguities between schema elements (classes/properties) and data instances (named entities) before query construction.

### 2.2. Model Context Protocol (MCP) Based Pipelines

While the academic research described above focuses on complex reasoning loops, the industrial landscape is coalescing around the Model Context Protocol (MCP) to standardize how LLMs connect to external data. A review of existing MCP implementations for RDF graphs reveals a dichotomy between "General Purpose" and "Specific-oriented" servers.

#### 2.2.1. General Purpose Servers

Implementations such as the "RDF Explorer" provide basic connectivity to SPARQL endpoints. These servers typically offer high-agency tools like `sparql_query`, which delegate the entire burden of query construction to the LLM. While flexible, they often lack safety mechanisms against hallucination and do not provide robust schema exploration capabilities beyond simple dumps (e.g., `schema://all`).

#### 2.2.2. Specific-Oriented Servers

The "Wikidata MCP Server" and "Proto-OKN Server" offer more specialized tools. For instance, the Wikidata server includes low-granularity tools like `search_entity` and

`search_property` to help the LLM find correct IDs before building the query. However, the reliance on a high-agency "Explorative-Generative workflows" fails to provide sufficient architectural barriers against the threat of schema hallucination. This approach exposes the system to significant stability risks in complex Text-to-SPARQL tasks, particularly when navigating domain-dense graphs where the lack of iterative validation loops renders standard zero-shot reasoning insufficient.

The analysis highlights a critical trade-off between "Agency" and "Reliability." High-agency servers (Text-to-SPARQL) are prone to hallucinations and syntax errors because they lack the validation layers found in frameworks like ARUQULA [1] and mKGQAgent [10]. Conversely, low-agency servers (parameterized tools) are often too rigid to handle complex, multi-hop questions.

### 2.3. Our Approach

Our project aims to bridge the gap between these two macro categories. The dynamic agentic frameworks [1] [11] [10] demonstrate that reliability requires iterative exploration, schema pruning, and execution feedback loops. However, these frameworks are often monolithic applications. By adopting the Model Context Protocol design standard, we aim to implement these sophisticated retrieval logic patterns—specifically the "schema pruning" and "neighborhood retrieval" strategies—within a standardized, modular MCP architecture.

## 3. Methodology

Our approach shifts the paradigm of Text-to-SPARQL generation from a single-shot translation task to an iterative, agentic workflow. While initial designs adhered to standard "Explorative-Generative Workflows" common in Model Context Protocol (MCP) implementations [3], we observed significant performance degradation when applied to specific, domain-dense graphs like **DOREMUS** [4]. To address the trade-off between the reliability of rigid parameterization and the reasoning capabilities of Large Language Models (LLMs), we propose a Hybrid Agentic Framework governed by a *Query Container* architecture.

### 3.1. Context Retrieval and Semantic Grounding

To ground the LLM's generation in the specific ontology of the target Knowledge Graph (KG), we move beyond static Schema-RAG approaches, which rely heavily on the quality of initial documentation indexing [6]. Instead, we adopt an iterative exploration strategy inspired by the ARUQULA framework [1]. As shown in Figure 2, the architecture provides exploratory tools such as `find_candidate_entities` and `get_entity_properties` that enable the agent to bridge the gap between vague natural language terms and precise Graph IRIs.

This module utilizes a hierarchical ontology representation to perform focused Named Entity Resolution (NER) and property traversal. By dynamically building a summarized ontology representation, the agent avoids traversing the entire RDF graph for every request. This reduces the search space for entity resolution and ensures that the retrieval process remains resilient to the "niche" structural characteristics of complex graphs like DOREMUS, addressing the hallucination issues inherent in zero-shot prompts [11].

### 3.2. Pattern-Based Query Construction

A core innovation of our methodology is the transition from "free-form generation" to "pattern-based construction." We reject the dichotomy of allowing the LLM to generate raw SPARQL (high hallucination risk) versus creating rigid, specific tools for every competency question (low scalability). Instead, we implement the *Query Container*—a stateful orchestrator within the MCP server that manages the incremental assembly of the SPARQL query. The *Query Container* replaces flat-string generation with a structural, object-oriented representation of SELECT, WHERE, and aggregation components. Its **Variable Registry** tracks the lifecycle of RDF variables and maps them to ontological classes, ensuring logical connectivity across iterative calls. To handle naming variations in musicological data, a **Pattern Expansion** mechanism expands singular URIs into `VALUES` blocks of equivalent identifiers. During serialization, the system applies Heuristic Projection to auto-manage aggregators or `SAMPLE()` functions for SPARQL compliance and uses **Dead-Code Elimination** to prune unused triples for optimized execution.

#### 3.2.1. Low-Granularity Tools

Rather than writing raw code, the LLM invokes atomic tools that abstract the complexities of the DOREMUS ontology through a template-driven architecture:

- **Base Generation:** `build_query` initializes the *QueryContainer* by consuming a template's *core definition*. This establishes the mandatory "ontological skeleton"—the specific triples required to define a primary entity (e.g., Expression, Performance) within the graph. It populates default `SELECT` variables and provides a dynamically generated *strategy guide* for subsequential retrival.
- **Iterative Refinement:** Parametric tools that incrementally inject specific graph patterns into the container:
  - `apply_filter`: Refines entities via attribute constraints or existence checks. For each template, the system defines specific relational paths that bridge the primary variable to target attributes, allowing the framework to be personalized to the specific topology of the

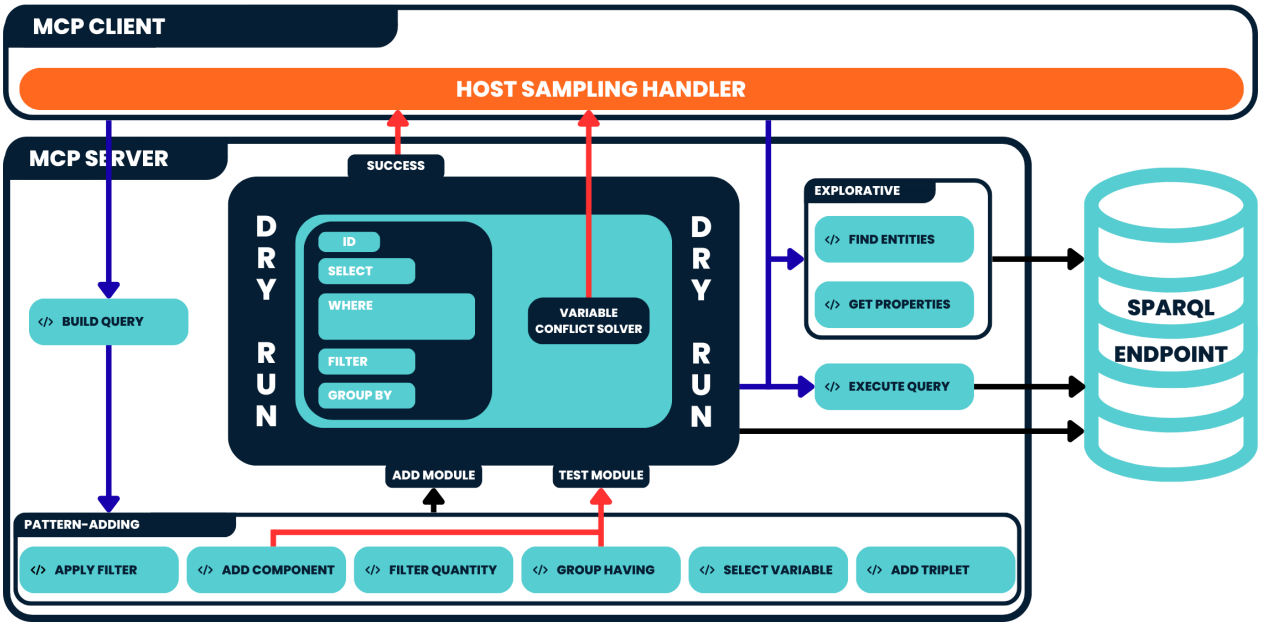


Figure 2. **Overview of the MCP server for Agentic Retrieval:** The architecture transitions from free-form generation to stateful assembly within a Query Container, utilizing, explorative tools for schema discovery, low-granularity tools for iterative refinement and a Sampling loop with Dry Run simulation to ensure syntactic and semantic validity before final SPARQL execution.

Knowledge Graph. It resolves natural language labels into precise URIs and supports `FILTER REGEX` patterns for string-based attributes.

- `add_component_constraint`: Resolves complex compositional logic through ontology traversal. It utilizes a pathfinding algorithm to link variables (e.g., works to instrumentation) and supports quantitative constraints like specific instrument counts (e.g., "for 3 violins").
- `filter_by_quantity`: Resolves numerical and temporal constraints, including durations, date ranges, and typed quantities.
- `groupBy_having`: Applies aggregation logic (e.g., `COUNT`, `AVG`) to segment results by calculated metrics. It supports comparative operators (`>`, `<`, `=`, `range`) within the `HAVING` clause for high-order queries.
- `select_aggregate_variable`: Updates the `SELECT` clause to project specific variables or apply aggregators (e.g., `COUNT`, `MIN`) for quantitative answers.
- `add_triplet`: Acts as a low-level fallback for manual graph traversal when standard patterns are insufficient; it includes a schema-validation layer to ensure property-class compatibility with the underlying ontology.

#### 3.2.2. Maintaining consistency in the Execution loop

This architecture facilitates an iterative **execution feedback loop** by providing the complete SPARQL state in every tool's output. Consequently, the agent can monitor the query's evolution in real-time [2], choosing to continue the build-up or restart the process via `build_query` if the semantic direction is flawed. The dry-run simulation acts as a formal *guarantor of consistency*: a module is only committed to the Query Container if the resulting state is syntactically valid and produces a non-empty result set. By ensuring the container always maintains a valid state, the system enables efficient error recovery. Rather than restarting the entire workflow upon a failure, the agent can simply refine the parameters of the last tool call to find a valid injection path. This methodology aligns with recent findings that reasoning capabilities scale with inference-time compute, leveraging iterative refinement methods akin to Tree of Thoughts" [12] and Self-Refine" [8] to guarantee query integrity before final execution.

### 3.3. Micro-Sampling for Decision Making

Although state-of-the-art agents such as ARUQULA employ complex dual-strategy architectures—combining Vector Stores for schema grounding and Lucene indices for entity resolution [1]—our approach aims at mitigating hallucinations and simplifying the disambiguation process by leveraging the *Sampling* primitive defined in the MCP specification [3].

Instead of maintaining external indices to resolve ambiguity, the server requests completions from the client via specific "*micro-sampling*" prompts. These prompts inject

the current user question, the state of the Query Container, and a *constrained set* of valid options (e.g., a list of potential property paths discovered via the exploratory tools). By reducing the generation task to a selection classification task—where the LLM simply selects an index rather than generating a string—we ensure that the agent's decisions are strictly grounded in the retrieved graph context. This allows the system to resolve semantic ambiguities (such as variable naming or path selection) using on-demand LLM reasoning rather than pre-computed vectors.

## 4. Experiments

In this section, we aim to empirically validate the hypothesis that an iterative, tool-driven approach significantly outperforms static generation methods for complex Knowledge Graph retrieval. Through a cross-analysis of tool usage patterns and model behaviors, we structure our evaluation around four primary questions:

- How does the proposed Agentic MCP framework compare to existing static and dynamic solutions in terms of retrieval accuracy and schema compliance?
- What is the marginal contribution of specific low-granularity tools (e.g., pattern-based refinement) to the overall system performance, and how does tool selection vary across architectures?
- How does the choice of the host LLM influence the consistency and computational efficiency of the workflow?
- To what extent do the proposed safety mechanisms—specifically *dry-run validation* and *micro-sampling*—mitigate hallucinations and ensure the semantic validity of the generated queries?

### 4.1. Evaluation Dataset

To evaluate the performance of agentic retrieval strategies over the DOREMUS Knowledge Graph, we present a specialized dataset of 64 Natural Language Questions (NLQs) and corresponding SPARQL queries. Focusing on the domain of music works and performances, this dataset is derived from the original Competency Questions (CQs) used to validate the DOREMUS ontology [4]. These baseline CQs were systematically expanded and refined to encompass a broader spectrum of linguistic and structural complexity.

#### 4.1.1. Ontological Challenges and Dataset Adaptation

The primary challenge in evaluating retrieval over DOREMUS lies in the high level of abstraction inherent to the CIDOC-CRM and FRBRoo ontologies [4]. The ontology employs complex abstract classes (e.g., `F22_Self-Contained_Expression`, `F28_Expression_Creation`) that do not align with natural language. Consequently, nearly all queries require complex inference and multi-hop traversal, rendering standard "Direct Mapping" classification buckets ineffective. To strictly evaluate reasoning capabilities rather than simple pattern matching, we applied a preprocessing protocol to the original CQs. Sorting clauses (`ORDER BY`) were eliminated to prioritize set-based accuracy metrics (Precision/Recall), while optional patterns (`OPTIONAL`) were pruned to reduce noise, ensuring the evaluation focuses exclusively on the agent's ability to retrieve necessary structural constraints. Finally, NLQs were rewritten to strictly align with the semantic intent of the SPARQL query. This ensures that the model is evaluated on its ability to derive logical structure from the question, mitigating discrepancies caused by incomplete ontology coverage or ambiguous intent.

#### 4.1.2. Complexity Stratification

To enable a layered analysis of agentic performance, the dataset is stratified into four distinct complexity levels based on graph topology and aggregation logic. The distribution is balanced, with Easy, Medium, and Hard classes each representing approximately 30% of the dataset, while the Very Hard class comprises less than 10%.

- **Easy (Structural Retrieval):** Queries consisting of fewer than 5 triple patterns within the `WHERE` clause. These require simple attribute filtering without complex traversals or aggregations.
- **Medium (Path Traversal):** Longer queries (up to 13 triple patterns) requiring multi-hop reasoning and complex filter conditions, but lacking aggregation functions. This tests the agent's ability to navigate deep graph paths, such as matching a composition to a performance via an intermediate expression.
- **Hard (Analytical and Open-Ended):** Queries involving aggregation functions (e.g., `COUNT`, `GROUP BY`) and conditional logic on aggregated data (e.g., `HAVING`). These questions frequently necessitate the use of the `add_triplet` tool for manual graph construction or to resolve patterns that defy standard templates, testing the agent's capacity for quantitative reasoning and flexible traversal.
- **Very Hard (Tool Infeasible):** Queries that are currently "impossible" to answer completely using the available toolset, typically involving nested subquery patterns or extremely complex filtering logic. While these cannot be fully resolved, they evaluate the agent's ability to provide high-quality **partial answers**. This class represents a minimal fraction of the dataset, serving primarily to identify the current "reasoning ceiling" of the framework.

### 4.2. Evaluation Metrics and Protocols

To rigorously assess the validity of the generated SPARQL queries, we employed an execution-based evaluation protocol rather than simple string matching. This approach accounts for the fact that syntactically different queries can

yield identical results on a Knowledge Graph and thus be considered equally correct.

#### 4.2.1. Evaluation Pipeline

Both the ground-truth (reference) query and the agent-generated query are executed against the live DOREMUS SPARQL endpoint. The system determines the accuracy score based on the data type of the returned results:

**1. Entity-Centric Queries (URI Extraction):** For the majority of queries returning graph entities, we extract all values beginning with `"http"` from the result sets, treating the results as unordered sets of URIs. This effectively neutralizes irrelevant ordering differences. The accuracy is calculated as a precision-oriented overlap metric:

$$Accuracy = \begin{cases} 1.0 & \text{if } |U_{gen}| = 0 \wedge |U_{ref}| = 0 \\ 0.0 & \text{if } |U_{gen}| = 0 \wedge |U_{ref}| > 0 \\ \frac{|U_{gen} \cap U_{ref}|}{|U_{gen}|} & \text{otherwise} \end{cases} \quad (1)$$

Where $U_{gen}$ is the set of URIs retrieved by the generated query and $U_{ref}$ is the set of URIs retrieved by the reference query.

**2. Scalar and Literal Fallback (LLM-Judge):** If no URIs are detected in either result set (e.g., for `COUNT` aggregations, Boolean `ASK` queries, or literal filtering), the pipeline falls back to an **LLM-as-a-Judge** mechanism. An evaluator LLM compares the result of the assistant's query against the reference result, assigning a discrete score of 1.0 (Correct), 0.5 (Partially Correct), or 0.0 (Incorrect) based on semantic equivalence.

### 4.3. Validating Tool Contributions

To quantify the impact of specific components within the agentic workflow, we conducted an evaluation on two distinct models: the high-reasoning `gpt-4.1` and the specialized `qwen3-coder 30b`. Figure 3 illustrates the accuracy trajectory as access to specific tools is systematically enabled, revealing critical insights into the architecture's dependencies.

#### 4.3.1. The Baseline vs. Dynamic Workflows

The baseline performance (where advanced tools are denied) underscores the insufficiency of unrestrained "*Explorative-Generative Workflows*", such as those observed in standard Wikidata MCP implementations. The data highlights that one-shot generation is inadequate for the DOREMUS ontology, necessitating a dynamic, multi-step agentic approach.

#### 4.3.2. Tool-Specific Contributions

As displayed in Figure 3, the most significant accuracy jumps are attributed to the `Build Query` (BQ) and `Add Filter` (AF) tools. This validates our hypothesis that modeling basic RDF patterns through templates reduces

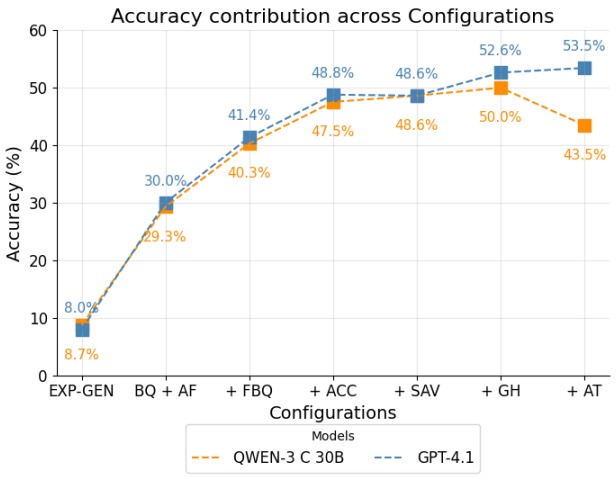


Figure 3. **Incremental accuracy contribution of agentic tools across model architectures.**: Here we represent a comparison based on the type of tools provided to the LLM client. EXP-GEN refers to "Explorative-Generative Workflows", where the LLM should explore the ontology and then generate the query. The tools are represented by their initials: BQ for Build Query, AF for Add Filter, FBQ for Filter By Quantity, ACC for Add Component Constraint, SAV for Select Aggregate Variable, GH for Groupby Having and AT for Add Triplet.

hallucination. Furthermore, the BQ tool's internal "*light classifier*" effectively directs the agent toward the correct template strategy (e.g., Artist vs. Expression) via few-shot prompting, providing a stable skeleton for the query. A further performance leap is observed with the introduction of the `Filter By Quantity` (FBQ) tool; this component contributes significantly to accuracy by abstracting complex **temporal and numerical patterns** (e.g., ISO-8601 durations or date ranges) that are frequent sources of syntax errors in standard generation. The `Add Component Constraint` (ACC) tool also serves a critical structural role. By employing recursive pathfinding logic, ACC effectively implements the **"Neighborhood Retrieval"** strategy defined in Section 2. It allows the agent to dynamically prune the schema and select relevant subgraphs (e.g., specific instrumentation logic) that would be too complex to infer in a single context window.

#### 4.3.3. The Reasoning Threshold

A divergence in model performance is observed when introducing advanced tools like `Groupby Having` (GH) and `Add Triplet` (AT). These tools essentially grant the agent "write access" to raw logic and graph traversal.

- **High-Reasoning Models:** For `gpt-4.1`, these tools provide the necessary flexibility to handle edge cases, yielding a final accuracy boost of approximately 5%. An example in Figure 4.

- **Model Degradation:** Conversely, `qwen3-coder 30b` exhibits a performance regression when these tools are enabled. This suggests a **"Reasoning Threshold"**: without sufficient reasoning depth, the model fails to manage the increased agency, misusing the open-ended tools and introducing semantic errors into the query container.

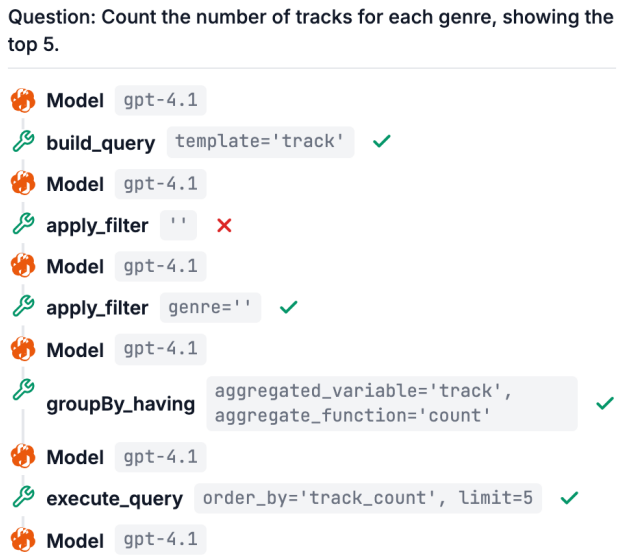


Figure 4. **Reasoning for aggregate SPARQL queries.** This execution trace from `gpt-4.1` illustrates the framework's capacity to handle complex analytical queries involving multi-stage aggregation. The model correctly identifies the `track` template, applies a non-restrictive genre filter to initialize the graph pattern, and successfully leverages the `groupBy_having` tool for statistical count operations. In contrast, smaller architectures like `ministral-3:14b` often fail to recognize the necessity of the grouping logic.

## 4.4. Comparative Analysis of LLM Hosts

Standard accuracy metrics alone are insufficient to evaluate agentic pipelines, as they fail to capture the nuances of the iterative refinement process. To derive a holistic assessment of model quality, we analyze performance across three dimensions: *Reliability* (consistency across runs), *Efficiency* (token/tool economy), and *Reasoning Quality* (performance by complexity stratum).

We benchmarked the following architectures to cover a spectrum of parameter sizes and licensing models:

- **Big models (1T to 5T parameters):** `openai/gpt-5.2-thinking`, `openai/gpt-4.1`
- **Medium models (100b to 1T parameters):** `qwen/qwen3-coder:480b`, `openai/gpt-oss:120b-thinking`
- **Small models (10b to 100b parameters):** `qwen/qwen-coder:30b`, `mistral/ministral-3:14b`

### 4.4.1. Reliability and Consistency

We evaluated the agent's stability by measuring the **Consistency Score**—the likelihood of the agent converging on the same answer across identical prompts. To ensure statistical significance, the metric is calculated in three independent experiments (three runs per question) for each model in the benchmark.

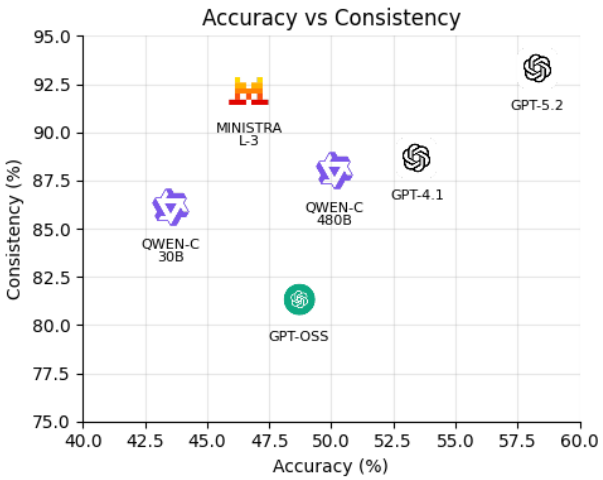


Figure 5. **Consistency scores across model architectures:** The high baseline (higher than 80%) indicates that the framework's constraints effectively mitigate syntactic stochasticity.

This metric is defined as a clipped inverse of the standard deviation:

$$Consistency = \text{clip}(1.0 - 2.0 \times \sigma_{question}, 0.0, 1.0)$$

As illustrated in Figure 5, all models exhibit a high consistency range (82%–93%). This validates the robustness of the **Query Container** architecture: by enforcing syntax constraints and performing "dry runs" within the tools, the framework offloads structural validity from the LLM. Consequently, performance drops are attributable to reasoning limitations (inability to handle complexity) rather than stochastic tool hallucination or syntax errors.

### 4.4.2. Efficiency: The Iteration Trade-off

We analyzed the computational cost of agentic retrieval by correlating the average number of tool calls with total token consumption. High-reasoning generalist models, such as `openai:gpt-4.1`, demonstrate superior efficiency, utilizing a minimal number of tool calls to isolate the correct schema path. However, the introduction of "*Thinking Models*" (e.g., `openai:gpt-5.2`) reveals a distinct trade-off: these architectures prioritize reasoning depth over speed, trading token efficiency for higher accuracy. As shown in

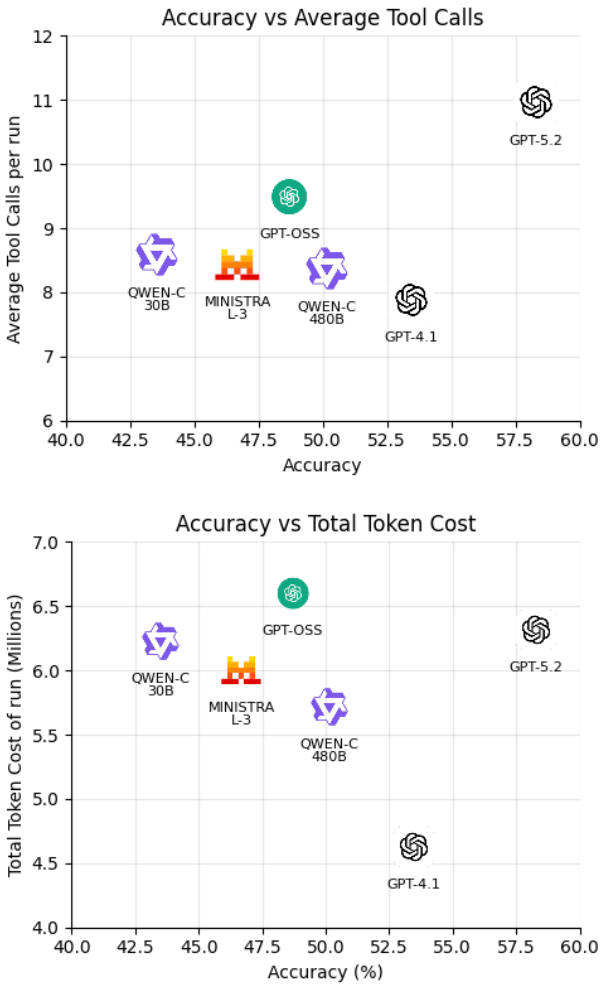


Figure 6. **Operational Efficiency Metrics:** Comparative analysis of average tool invocations per run and total token consumption. The data highlights the variance in computational cost required by different architectures to resolve the query.

Figure 6, this results in increased latency and execution costs , consistent with findings that reasoning performance scales with inference-time compute [12].

A notable emergent behavior is observed in smaller, agile models like `ministral-3:14b`, which mitigate this latency by leveraging **parallel tool invocation** (batch calling). This capability allows the agent to explore multiple schema hypotheses simultaneously rather than sequentially.

Conversely, specialized models like `ollama:qwen-coder480b` exhibit high resource consumption, generating significant token overhead. Despite possessing fewer parameters than the largest proprietary models, they achieve comparable accuracy by relying heavily on the **iterative feedback loop**.

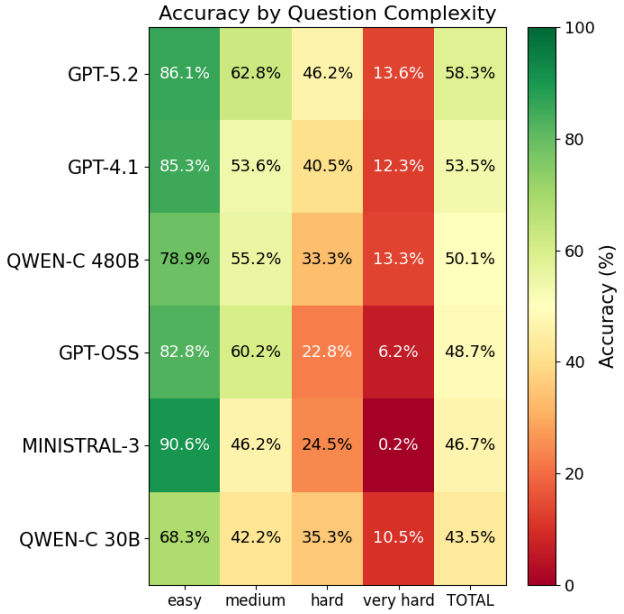


Figure 7. **Analysis of the accuracy metric across different complexity classes:** The top-performing models define the interpretability limit of the agentic framework (approx. 85% on Easy), while the vertical drop-off in the *Hard/Very Hard* columns demonstrates the necessity of high inference-time compute for resolving aggregation and multi-hop traversal.

### 4.4.3. Reasoning Quality by Complexity Stratum

By mapping model accuracy against the dataset complexity levels, we observe a clear divergence between specificity and scale: The heatmap in Figure 7 depicts how accuracy varies across the previously defined complexity strata.

First, the performance of state-of-the-art models (e.g., `gpt-5.2`) effectively establishes the **interpretability limit** of the current agentic framework. Since these models possess the necessary reasoning capabilities to solve the task, the remaining error margin represents the *intrinsic limitations of the framework itself*—such as missing tools, ambiguous tool definitions, or insufficient feedback granularity—rather than model hallucination. Consequently, a strong positive correlation is observed between the number of parameters and final accuracy, particularly within the *Hard* (Analytical) and *Very Hard* (Generative) strata.

Furthermore, we observe a nuanced relationship between parameter count and model specialization. While smaller models like `ministral-3:14b` remain competitive on *Easy* structural retrieval tasks—leveraging their coding specialization to fill templates—their performance degrades as the need for long-context reasoning increases, confirming that semantic planning requires the reasoning depth typically associated with larger parameter counts [7]. However, an interesting divergence appears when analyzing the

”Coder” (C) model family: despite the disparity in size, coding specialists exhibit a more uniform performance distribution across complexity classes compared to generalist models. This phenomenon suggests that queries classified as ”Hard” due to semantic aggregations (e.g., `GROUP BY`) often rely on syntactic patterns that are trivial for code-trained models, effectively decoupling semantic reasoning difficulty from code generation difficulty. Nonetheless, the sheer scale remains a deciding factor within this category, as the larger `qwen-coder480b` significantly outperforms its smaller counterpart in tool-use precision, proving that while specialization aids syntax, parameter density is still required to navigate the agentic decision tree effectively.

A distinct behavioral pattern emerges when analyzing ”*Thinking Models*” (e.g., `gpt-5.2` and `gpt-oss:120b`), which employ extended inference-time compute. Qualitative analysis reveals that these architectures excel in resolving *Variable Conflicts* and *Aggregation Scoping*, areas where standard models frequently falter. In complex constraint scenarios, such as ”*Give me the works written for oboe and orchestra after 1900*”, the ”Thinking” models successfully disentangle the variable paths for the solo instrument versus the ensemble. Conversely, non-thinking models often conflate the `medium_of_performance` path, resulting in empty or incorrect casting filters. In strict quantification queries like ”*Give me works written for violin, clarinet and piano (strictly)*”, the `gpt-5.2` model correctly identifies that the `groupBy_having` tool must be applied to the `casting` entity. In contrast, standard models frequently suffer from semantic drift, grouping by the `expression` instead, which invalidates the ”*strictly*” constraint.

However, this reasoning strategy entails significant risk when applied to smaller architectures. As evidenced by Figure 5, the smaller thinking model `gpt-oss:120b` exhibits degraded reliability compared to its larger counterparts. This failure mode can be attributed to limited context window and attention capacity: once a small model commits to an incorrect strategic plan in the early thought steps, it lacks the semantic flexibility to adjust direction. [12].

#### 4.4.4. Taxonomy of Failure

Finally, we categorized failure modes to isolate the root causes of incorrect generation. Thanks to the architecture’s execution-based validation (dry runs), syntax errors were effectively eliminated. The remaining faults fall into three distinct semantic categories:

- **Type I (Schema Hallucination):** The model invents valid-looking but non-existent schema elements (e.g., using `foaf name` instead of `rdfs label`) despite the availability of exploratory tools. This indicates a failure to verify assumptions against the retrieved vocabulary.
- **Type II (Tool Drift):** The model correctly identifies the intent but selects a sub-optimal tool strategy (e.g., forcing an `add_triplet` traversal when a safer `filter_by_quantity` logic was applicable).
- **Type III (Semantic Drift):** A ”silent failure” where the model produces a syntactically valid and executable query that answers a *different* question than the one asked (e.g., retrieving ”Composers” instead of ”Performers”). This represents the hardest challenge for automated evaluation, necessitating the use of LLM-as-a-Judge metrics.


Figure 8. **Iterative error recovery and multi-tool parallelization.** This trace from `ministral-3:14b` demonstrates the framework’s resilience against initial parameter hallucinations. The process begins with the `find_candidate_entities` tool, which retrieves the top 10 most similar entity URIs based on a provided label. In this instance, the initial `add_component_constraint` calls fail due to incorrect URI selection from the candidates; however, the subsequent execution feedback allows the model to rectify its internal state and re-evaluate the retrieved list. Ultimately, the agent resolves the correct instrument identifiers (`URI_1`), converting failed attempts into informative reasoning steps that converge on a valid SPARQL state.

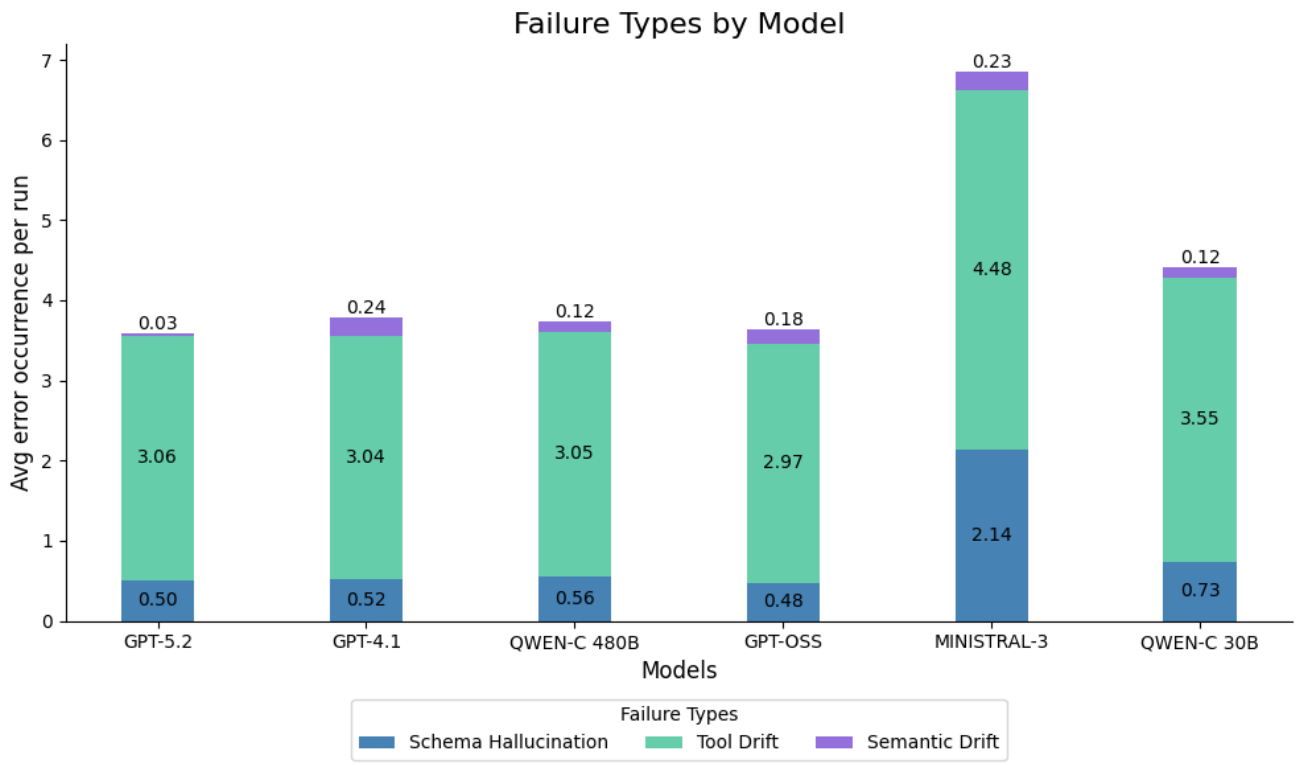


Figure 9. **Failure types across different LLMs:** Type II (Tool Drift) dominates the error landscape, while the framework's validation logic effectively suppresses Type I and Type III errors in high-reasoning models.

**Analysis of Error Distribution** As illustrated in Figure 9, the error distribution validates our architectural assumptions regarding stability and intent understanding:

- **Suppression of Type I Errors:** Consistent with the high stability scores observed previously, Type I errors are minimal for reasoning-capable models (e.g., `gpt-5.2`, `qwen-480b`). This confirms that the models have successfully internalized the tool definitions.
- **Minimal Semantic Drift (Type III):** The incidence of Type III errors is notably low across all architectures. This indicates that modern LLMs possess a robust understanding of user intent; they rarely misunderstand *what* is being asked. Qualitatively, we observed that the few recorded Type III errors were often cases of **format divergence** rather than logical failure—such as returning URIs instead of human-readable names, or returning a query definition instead of the executed answer—rather than a fundamental misalignment with the user's question.
- **The Dominance of Tool Drift:** Consequently, the majority of failures are concentrated in Type II (Tool Drift), represented by the dominant green bars in Figure 9. This reveals that while models understand the *goal* (low Type III) and the *vocabulary* (low Type I), they struggle to select the most efficient *strategy*. We observed that *dry-run* failures frequently stem from **ontological noise**—a condition where multiple distinct entities share equivalent labels in the vocabulary. This ambiguity makes it difficult for the model to distinguish the correct candidate, leading to a valid tool call that nonetheless executes an incorrect semantic pattern. Ultimately, this constitutes a strategic failure rather than a schema error: the model does not necessarily invoke the "wrong" tool, but the resulting pattern is flawed, effectively rendering the tool's application a Type II error. The tendency to fallback to complex tools (like `add_triplet`) instead of utilizing specific pattern-based tools remains the primary obstacle to achieving higher accuracy.
- **Ministral: Parallelism vs. Precision:** A distinct outlier in our analysis is `ministral-3:14b`. As observed in Figure 9, this model exhibits a significantly higher rate of Type I (Schema Hallucination) errors compared to its peers. This behavior stems from a fundamental trade-off: while its limited parameter count restricts the reasoning capacity required to strictly adhere to schema constraints, the model compensates by leveraging **Parallel Tool Invocation**. Unlike sequential reasoners, `ministral-3:14b` frequently utilizes the MCP proto-

col to execute multiple tools simultaneously—effectively "batching" its exploration. As observed in Figure 8, this allows it to conduct more trials and explore the RDF graph more broadly in less time. However, this aggressive exploration often lacks semantic precision: the model generates a high volume of hypotheses (tool calls) but lacks the reasoning depth to verify them against the ontology, leading to hallucinated parameters despite the availability of correct tools.

### 4.5. Synergy of Dry-run and Micro-Sampling

To isolate the contribution of our novel safety mechanisms, we conducted an ablation study on the `gpt-4.1` host. We measured performance across four configurations, toggling the *Micro-Sampling* (constrained decision-making) and *Dry-Run* (execution-based validation) features. The results, summarized in Table 1, reveal a distinct synergistic effect when both mechanisms are active.

Enabling only the Dry-Run mechanism improves accuracy by **+5.6%** over the baseline. This aligns with the "Query Correction Layer" concept proposed in FIRES-PARQL [9], where syntax errors are caught before finalization. However, this comes at the highest token cost (4.87M). While the validation logic prunes dead-end paths (reducing average tool calls to 7.61), the overhead of re-prompting the LLM to fix syntax errors inflates the token count without necessarily solving semantic ambiguities. Enabling only Micro-Sampling provides a significant accuracy boost of **+8.5%**. By converting open-ended generation into a constrained selection task, this mechanism effectively mitigates Type II errors (Tool Drift). However, this increases the average tool calls (8.61) and token usage. Without the Dry-Run safety net, the agent may confidently select semantically plausible but syntactically invalid paths, leading to wasted retrieval cycles.

#### 4.5.1. The Synergistic Optimization

The critical finding of this study is observed in the fully enabled configuration (**Sampling ON + Dry Run ON**). This setup achieves a peak accuracy of **53.45%**, representing a **13.7% absolute improvement** over the baseline.

More importantly, this configuration exhibits a **synergistic efficiency**. While one might expect the costs of Sampling and Dry-Run to add up, the combined token cost (4.62M) is actually *lower* than either individual configuration. This phenomenon can be explained by the complementary nature of the mechanisms. While sampling ensures that the agent makes semantically correct decisions early in the chain—exemplified in Figure 10—the Dry-Run ensures that those decisions are syntactically executable. Together, they minimize the number of "expensive failures"—long, hallucinated tool chains that eventually time out or error out. The agent reaches the correct answer in fewer valid steps (7.89 calls), proving that rigorous constraints actually reduce computational overhead.

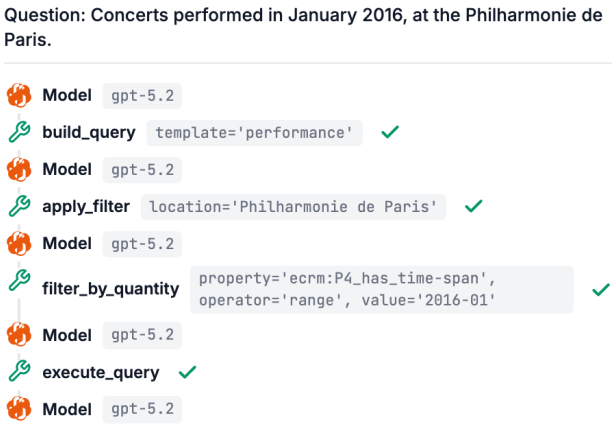


Figure 10. **Internalized Micro-Sampling for entity retrieval.** This trace from `gpt-5.2` highlights the efficiency of delegating entity resolution to the tool layer. Rather than forcing the LLM to manually explore the KG for the "Philharmonie de Paris" URI, the `apply_filter` tool internally resolves the label through a sampling-based mechanism.

## 5. Conclusions

In this work, we presented an implementation of the Model Context Protocol (MCP) designed to facilitate reliable Agentic Retrieval over Knowledge Graphs. By moving away from the standard "Intended-workflow" paradigm—which often suffers from high hallucination rates in complex ontologies—we proposed a **Hybrid Agentic Framework** focused on reliability and structural correctness.

Our approach leverages a **Pattern-Based Query Construction** strategy, utilizing a suite of low-granularity modular tools (e.g., `build_query`, `add_component_constraint`) to assemble SPARQL queries iteratively within a protected **Query Container** 3. We demonstrated that this architecture effectively balances the trade-off between LLM agency and structural constraints and we achieved a framework that is both effective in tackling the Text-to-SPARQL task and efficient in terms of token consumption. While the current framework establishes a strong baseline for agentic retrieval over the DOREMUS graph, several avenues remain for extending its versatility and reasoning depth.

- **Cross-Domain Generalizability (ODEUROPA):** A primary objective for future research is to validate the universality of our pattern-based tools. While demonstrated on the musical domain (DOREMUS), we aim to deploy the framework on the **ODEUROPA** Knowledge Graph

| Configuration | Accuracy (%) | Total Token Cost | Avg Tool Calls |
|---|---|---|---|
| Sampling OFF + Dry Run OFF (Baseline) | 39.75 | 4,553,907 | 8.36 |
| Sampling OFF + **Dry Run ON** | 45.36 | 4,872,659 | 7.61 |
| **Sampling ON** + Dry Run OFF | 48.21 | 4,789,270 | 8.61 |
| **Sampling ON + Dry Run ON** | **53.45** | 4,627,187 | 7.89 |

Table 1. Ablation study of safety mechanisms on `gpt-4.1`. The combination of Sampling and Dry-Run yields the highest accuracy while optimizing token efficiency compared to partial configurations.

[5], which models olfactory history and sensory data.

- **Recursive Subquerying Dynamics:** Currently, the Query Container operates primarily on linear logical chains. To address highly complex questions requiring nested logic (e.g., "Find composers who wrote works during the time they lived in cities where a specific event occurred"), future iterations will exploit **Subquerying Dynamics**. This involves enabling the agent to instantiate nested Query Containers—effectively treating a subquery as a distinct artifact that can be generated, validated, and then injected into the parent query as a filter condition or a `MINUS` block.
- **Mitigating Tool Drift via Definition Refinement:** Our error analysis identified **Type II (Tool Drift)** as the persistent bottleneck, where models resort to generic traversal tools (`add_triplet`) rather than specific patterns, as shown in Figure 9. Future work will focus on optimizing the semantic definitions within the MCP `Tools` schema.

## References


[1] Felix Brei, Lorenz Bühmann, Johannes Frey, Daniel Gerber, Lars-Peter Meyer, Claus Stadler, and Kirill Bulert. Aruqula – an llm based text2sparql approach using react and knowledge graph exploration utilities, 2025. 2, 3, 4

[2] Aleksandr Gashkov, Aleksandr Perevalov, Maria Eltsova, and Andreas Both. Sparql query generation with llms: Measuring the impact of training data memorization and knowledge injection, 2025. 4

[3] Xinyi Hou, Yanjie Zhao, Shenao Wang, and Haoyu Wang. Model context protocol (mcp): Landscape, security threats, and future research directions, 2025. 1, 3, 4

[4] Pasquale Lisena, Manel Achichi, Pierre Choffé, Cécile Cecconi, Konstantin Todorov, Bernard Jacquemin, and Raphaël Troncy. Improving (re-) usability of musical datasets: An overview of the doremus project. *Bibliothek Forschung und Praxis*, 42(2):194–205, 2018. 2, 3, 5

[5] Pasquale Lisena, Daniel Schwabe, Marieke van Erp, Raphaël Troncy, William Tullett, Inger Leemans, Lizzie Marx, editor="Groth Paul Ehrich, Sofia Colette", Maria-Esther Vidal, Fabian Suchanek, Pedro Szekley, Pavan Kapanipathi, Catia Pesquita, Hala Skaf-Molli, and Minna Tamper. Capturing the semantics of smell: The odeuropa data model for olfactory heritage information. https://doi.org/10.1007/978-3-031-06981-9_23, 2022. 12

[6] Chuangtao Ma, Sriom Chakrabarti, Arijit Khan, and Bálint Molnár. Knowledge graph-based retrieval-augmented generation for schema matching, 2025. 2, 3

[7] Chuangtao Ma, Yongrui Chen, Tianxing Wu, Arijit Khan, and Haofen Wang. Large language models meet knowledge graphs for question answering: Synthesis and opportunities, 2025. 1, 2, 8

[8] Aman Madaan, Niket Tandon, Prakhar Gupta, Skyler Hallinan, Luyu Gao, Sarah Wiegreffe, Uri Alon, Nouha Dziri, Shrimai Prabhumoye, Yiming Yang, Shashank Gupta, Bodhisattwa Prasad Majumder, Katherine Hermann, Sean Welleck, Amir Yazdanbakhsh, and Peter Clark. Self-refine: Iterative refinement with self-feedback, 2023. 2, 4

[9] Xueli Pan, Victor de Boer, and Jacco van Ossenbruggen. Firesparql: A llm-based framework for sparql query generation over scholarly knowledge graphs, 2025. 11

[10] Aleksandr Perevalov and Andreas Both. Text-to-sparql goes beyond english: Multilingual question answering over knowledge graphs through human-inspired reasoning, 2025. 2, 3

[11] Guangyuan Piao, Michalis Mountantonakis, Panagiotis Papadakos, Pournima Sonawane, and Aidan O'mahony. Toward exploring knowledge graphs with llms. In *International Conference on Semantic Systems*, 2024. 2, 3

[12] Shunyu Yao, Dian Yu, Jeffrey Zhao, Izhak Shafran, Thomas L. Griffiths, Yuan Cao, and Karthik Narasimhan. Tree of thoughts: Deliberate problem solving with large language models, 2023. 4, 8, 9