Structured Agentic Retrieval over Knowledge Graphs with MCP and Query Containers

Simone Fassio[1,†], Dario Gosmar[1,†], Fanfu Wei[1], Thibault Ehrhart[1], Pasquale Lisena[1] and Raphael Troncy[1]

[1]EURECOM, Sophia Antipolis, France

Abstract

Integrating Large Language Models (LLMs) with Knowledge Graphs (KGs) remains challenging due to the difficulty of translating natural language questions into valid and semantically grounded graph-based queries. Free-form SPARQL generation is particularly prone to hallucinations and errors when operating over complex and domain-specific schemas. In this paper, we propose a framework for structured agentic retrieval over KGs based on the Model Context Protocol (MCP). Our approach replaces unconstrained generation with a tool-driven workflow in which queries are incrementally constructed through low-granularity operations. Central to this design is the Query Container, a stateful abstraction that maintains a consistent query state and enforces structural validity throughout the process. We further introduce a hybrid decision mechanism combining constrained tool usage with model-guided selection via micro-sampling, coupled with execution-based validation. Experimental results show improved robustness and a reduction in hallucination-related errors compared to standard generation approaches.

Keywords

Model Context Protocol, Large Language Models, SPARQL, Knowledge Graphs

1. Introduction

The integration of Large Language Models (LLMs) with Knowledge Graphs (KGs) represents a critical frontier in Artificial Intelligence, combining the inference capabilities of generative models with the factuality of knowledge bases. However, enabling LLMs to reliably translate Natural Language Questions (NLQs) into executable graph queries (such as SPARQL queries) remains a significant challenge. A major 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.

In the literature, approaches to this problem have relied on fine-tuning models on domain-specific datasets [1]. While effective, fine-tuning requires expensive retraining whenever the schema evolves and may create data bottlenecks due to the scarcity of high-quality training pairs. Consequently, the field has shifted toward RAG-based approaches, such as *SchemaRAG* [2]. Unlike traditional RAG, which retrieves data instances, SchemaRAG retrieves a filtered subset of the database's metadata, essentially providing the LLM with a dynamic "blueprint" required to construct a valid query. Static SchemaRAG approaches remain vulnerable to ambiguity: if the initial retrieval step fails to identify the correct property path, the generation fails.

To address these limitations, we propose an 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, particularly Tools for graph traversal and Sampling for resolving semantic ambiguities, the system transforms passive retrieval into a dynamic workflow that iteratively invokes low-granularity tools to explore the graph structure and validate assumptions step-by-step before query generation. Our architecture implements a *Query Container* mechanism that

RAGE-KG 2026: The Third International Workshop on Retrieval-Augmented Generation Enabled by Knowledge Graphs, co-located with ISWC 2026, October 25–26, 2026, Bari, Italy

[†]These authors contributed equally.

simone.fassio@eurecom.fr (S. Fassio); dario.gosmar@eurecom.fr (D. Gosmar); fanfu.wei@eurecom.fr (F. Wei); thibault.ehrhart@eurecom.fr (T. Ehrhart); pasquale.lisena@eurecom.fr (P. Lisena); raphael.troncy@eurecom.fr (R. Troncy)



© 2026 Copyright for this paper by its authors. Use permitted under Creative Commons License Attribution 4.0 International (CC BY 4.0).

CEUR Workshop Proceedings ceur-ws.org ISSN 1613-0073

published 2026-11-04

CEUR-WS.org/Vol-4312/paper35.pdf

constructs SPARQL queries incrementally using modular tools. By embedding *neighborhood retrieval* inside iterative pattern-construction 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 improving robustness and allowing the agent to recover from invalid intermediate decisions [4]. Our main contribution is a MCP-based architecture that replaces free-form SPARQL generation with execution-validated query construction through interoperable MCP operations supporting schema exploration, constrained decision-making, and recovery from invalid updates. Our evaluation on the challenging use case of the DOREMUS KG for classical music [5] shows an improvement of 13.7 percentage points over the baseline.

The remainder of this paper is organized as follows. After an overview of the state of the art in Section 2, we detail our methodology in Section 3. The experimental setup and results are discussed in Section 4. We conclude and outline some future work in Section 5.

2. Related Work

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 [1]. To address this limitation, the field has converged on algorithmic strategies to isolate relevant structural elements: **Schema Pruning**, that is the selection of a minimal subset of the ontology to reduce noise, 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.

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. propose a framework for exploring KGs in a zero-shot setting, employing a cyclic workflow where a *Query Enhancer* validates extracted predicates against the ontology and refines the generated SPARQL based on execution feedback [7]. 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 [8].

A more formalized agentic approach is presented in ARUQULA [9]. Utilizing the ReAct (Reason and Act) paradigm, ARUQULA implements a rigorous feedback-driven validation loop. The agent is equipped with exploration tools 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.

These methodologies collectively confirm that multi-step approaches significantly outperform static generation methods, particularly in reducing hallucinations and handling complex multilingual settings.

2.2. Model Context Protocol (MCP) Based Pipelines

Implementations such as the RDF Explorer[1] 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`). More recently, an MCP interface has been integrated into the Comunica framework, enabling querying

[1] https://www.pulsemcp.com/servers/emekaokoye-rdf-explorer

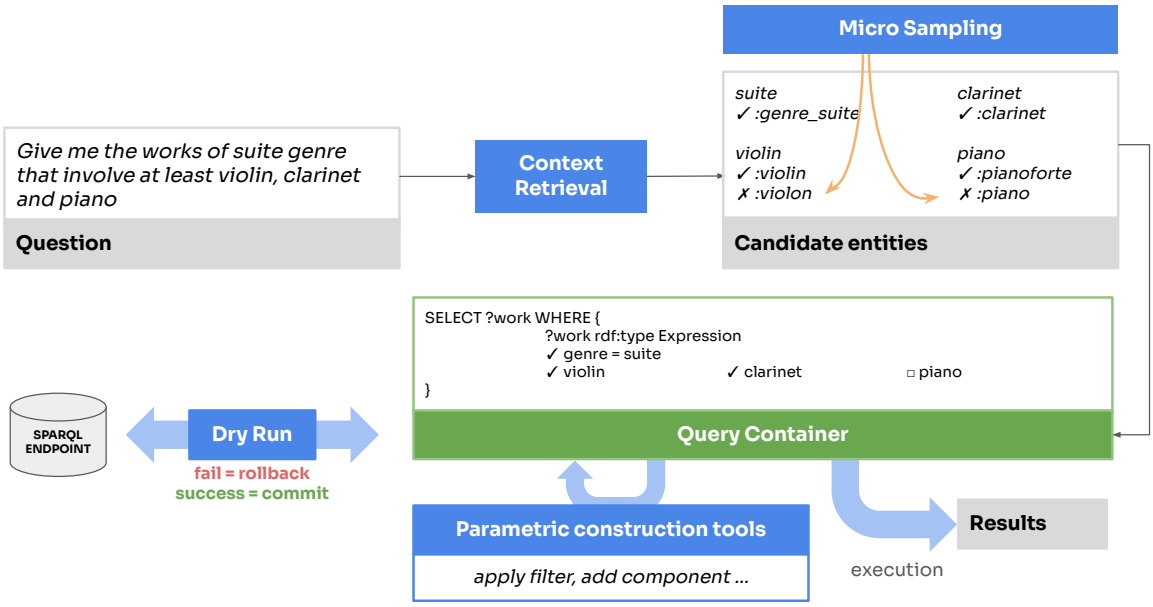


Figure 1: Running example illustrating the complete execution workflow.

different type of SPARQL endpoints, including RDF representation [10]. In [11], A SHACL description of the knowledge structure is used to guide agents in accessing the Knowledge Graph through the MCP protocol.

The Wikidata MCP Server[2] and Proto-OKN Server [12] offer more specialized tools. For instance, the Wikidata server includes low-granularity tools such as `search_entity` and `search_property` to help the LLM find correct IDs before building the query. We define this approach as Explorative-Generative: the LLM iteratively alternates between graph exploration (retrieving schema elements, entities, or contextual metadata from an RDF graph) and query generation, where the retrieved context is used to synthesize executable SPARQL queries. This pattern underlies most existing MCP-based Knowledge Graph assistants, including the Wikidata MCP Server. However, the reliance on 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 such as [9] and [8]. Conversely, low-agency servers (parameterized tools) are often too rigid to handle complex, multi-hop questions, and are often monolithic applications. Our work aims to bridge the gap between these two categories, by adopting the Model Context Protocol design standard with the introduction of these sophisticated retrieval logic patterns within the MCP architecture.

3. Methodology

Our approach reformulates Text-to-SPARQL generation as an iterative agentic workflow rather than a single-shot translation task. Instead of directly generating SPARQL queries, the agent progressively constructs them through a sequence of low-granularity tool invocations operating on a persistent *Query*

[2] https://github.com/zzaebok/mcp-wikidata

Tool and description	MCP/API parameters	Example
find_candidate_entities Resolves a label to candidate KG entities.	name, entity_type	("violin", "vocabulary")
get_entity_properties Retrieves properties and values of an entity.	entity_uri	("ecrm:E21_Person")
build_query Initializes a Query Container from a template.	question, template	("Sonatas by Beethoven", "expression")
apply_filter Adds template-based constraints.	query_id, target_variable, schema_template, filters	{"composer_name":"Mozart"}
add_component_constraint Adds a component relation, optionally with cardinality.	query_id, source_variable, target_component, exact_count	("expression", "violin", 3)
filter_by_quantity Adds numerical or temporal constraints.	query_id, variable, property, operator, value(s)	duration > "PT15M"
groupBy_having Adds grouping and aggregate constraints.	query_id, group_by, aggregate, condition	COUNT(castingDetail) = 3
add_triplet Adds a schema-validated RDF triple as fallback.	query_id, subject, subject_class, property, object, object_class	expression mus:U4 pubEvent
select_aggregate_variable Selects or aggregates an output variable.	query_id, projection_variable, aggregator	COUNT(expression)
execute_query Executes the current Query Container.	query_id, limit, order_by, desc	limit=10

Table 1
Tools exposed by the MCP server.

Container. Each modification is validated before being committed, allowing the framework to recover from intermediate reasoning errors while preserving the structural consistency of the generated query.

Throughout this section, we illustrate the methodology using the following competency question:

Example 1. *Give me the works of suite genre that involve at least violin, clarinet and piano.*

Figure 1 summarizes the complete workflow. Starting from the natural language question, the framework first resolves ambiguous terms such as *suite* and *violin* into Knowledge Graph entities, initializes a Query Container for musical works, incrementally enriches it through parameterized construction tools, validates every modification through execution-based checks, and finally serializes the resulting state into an executable SPARQL query. The full list of tools is instead reported in Table 1.

3.1. Context Retrieval and Semantic Grounding

The first challenge is to determine which Knowledge Graph entities correspond to the terms *suite*, *violin*, *clarinet*, and *piano*. Rather than relying solely on the parametric knowledge of the LLM, the framework grounds the reasoning process by progressively retrieving schema and entity information from the Knowledge Graph. Directly querying the entire ontology for every user request would be inefficient and would provide the LLM with much more information than necessary.

Instead, the framework performs an **offline preprocessing step** that extracts a compact summary of the ontology. This summary is loaded into memory when the server starts and is subsequently used to guide the retrieval of relevant schema elements during query construction. The summary stores the schema topology together with simple statistics about class and relation frequencies.

At **runtime**, the framework first builds a compact schema summary by selecting only the most

representative classes and relations from the summary[3]. Then, it retrieves the entities mentioned in the user's question using the `find_candidate_entities` tool. For each term, the tool queries the SPARQL endpoint using label-based matching and returns a small set of candidate entities. In Example 1, independent searches are performed for the terms *suite*, *violin*, *clarinet*, and *piano*. When additional information is required to distinguish between multiple candidates, the framework invokes `get_entity_properties`. This tool retrieves the populated properties of an entity or explores its local schema neighborhood, allowing the LLM to better understand the role of the candidate within the ontology before constructing the query. At the end of this stage, the framework has identified a small set of candidate entities together with the local schema information required for the subsequent query construction phase.

This two-stage process ensures that the system remains resilient even with complex graphs, reducing the load on the retrieval part and addressing the hallucination issues inherent in zero-shot prompts [7].

3.2. Pattern-Based Query Construction

Once the relevant entities have been identified, the framework constructs the SPARQL query incrementally rather than generating it as free-form text. The process relies on a persistent **Query Container**, which maintains the current query state, including projected variables, graph patterns, filters, and aggregation clauses. A variable registry tracks the ontology type of each introduced variable, while duplicate patterns and aggregation syntax are handled automatically during serialization.

The container is initialized through `build_query`, using a template associated with the primary concept targeted by the question. A template is a lightweight annotated SPARQL file containing the mandatory graph patterns that characterize an ontological concept, together with reusable patterns for common attributes. For example, the *Expression* template defines the triples representing a musical work and its creation event, as well as optional patterns for attributes such as genre or composer. Entity values are not encoded in the template, but are selected dynamically during query construction. Templates therefore capture generic schema knowledge rather than complete question-specific queries and typically require less than 30 minutes to create for an ontology expert.

In Example 1, the agent invokes `build_query` with the *Expression* template, initializing a container whose main variable represents an instance of `efrbroo:F22_Self-Contained_Expression`. The `apply_filter` tool then activates the genre pattern and binds the selected URI corresponding to *suite*.

The instrumentation constraints require more complex graph paths. The agent therefore invokes `add_component_constraint` for *violin*, *clarinet*, and *piano*. Each invocation identifies a valid multi-hop path connecting the work variable to the selected instrument and proposes the corresponding graph fragment as an update to the Query Container. The framework provides a small set of construction tools covering the principal query operations: `build_query` initializes the container, `apply_filter` adds attribute constraints, `add_component_constraint` introduces multi-hop relations, `filter_by_quantity` handles numerical and temporal conditions, `groupBy_having` and `select_aggregate_variable` support aggregation, and `execute_query` serializes and executes the resulting SPARQL query.

Because every tool modifies the same persistent state, an unsuccessful operation affects only the latest proposed update. Previously validated patterns remain unchanged, allowing the agent to correct an intermediate decision without restarting the entire construction process.

3.3. Dry-Run and Micro-Sampling

The system is further reinforced by two mechanisms that complete the tool behavior and improve its robustness.

[3]In our implementation, the framework retains the 15 most instantiated classes and iterates over relations in descending order of frequency, selecting at most 20 schema triples whose endpoints belong to these classes. When this global summary is insufficient, the in-memory `GraphSchemaExplorer` retrieves a localized one- or two-hop schema neighborhood around the relevant class.

Dry-run executes each proposed graph fragment as a draft query before it is committed to the Query Container. The fragment is temporarily added to a copy of the current query and executed against the SPARQL endpoint. If the resulting query is invalid or returns no results, the modification is rejected and the previous container state is preserved; otherwise, the fragment is committed[4]. This mechanism does not guarantee that a non-empty query is semantically correct, but it prevents invalid or unpopulated graph paths from corrupting an otherwise valid query state.

In Example 1, the LLM initially selects an incorrect URI for *violin*. The corresponding invocation of `add_component_constraint` proposes a graph path that returns no results during Dry-Run. The update is therefore rejected, and the Query Container rolls back to the state containing only the previously validated genre constraint.

Micro-sampling resolves semantic ambiguities by asking the LLM to select from a constrained set of valid candidates rather than generating graph elements freely. Returning to Example 1, after the first *violin* candidate is rejected, the remaining candidate URIs are presented to the LLM through the MCP Sampling primitive. The selected alternative is again validated through Dry-Run and, once successful, committed to the container. The same process is subsequently applied to the *clarinet* and *piano* constraints.

4. Experiments

4.1. Dataset

To evaluate the performance of our approach, we present a challenging dataset of 64 Natural Language Questions (NLQs) and corresponding SPARQL queries, derived from the original Competency Questions (CQs) used to validate the DOREMUS ontology [5].

The DOREMUS Knowledge Graph was selected due to its particularly challenging characteristics. First, its ontology relies on and extends the CIDOC-CRM and FRBRoo ontologies [13, 14, 15] which provide a high level of abstraction. Hence, 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* 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, while optional patterns (`OPTIONAL`) were pruned to reduce noise, ensuring that the evaluation focuses exclusively on the agent's ability to retrieve the necessary structural constraints. Finally, the dataset has been stratified into four distinct complexity levels based on graph topology and aggregation logic.

- 30% Easy (Structural Retrieval): queries consisting of fewer than 5 triple patterns within the `WHERE` clause, requiring just simple attribute filtering.
- 30% Medium (Path Traversal): up to 13 triple patterns, requiring multi-hop reasoning and complex filter conditions.
- 30% 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 challenge standard templates.
- 10% 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.

[4]The non-empty-result requirement used by Dry-Run is appropriate for the present evaluation because every reference query is known to return at least one result. In an open deployment, this condition should be configurable so that structurally valid queries may legitimately return an empty answer.

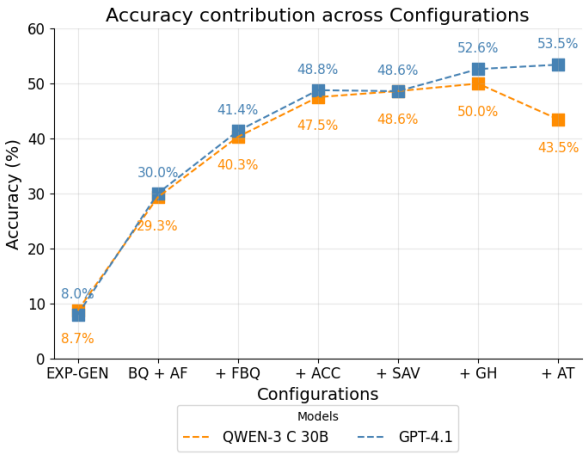


Figure 2: Incremental accuracy contribution of agentic tools, starting from the Explorative-Generative baseline (EXP-GEN) to the addition of Build Query (BQ), Add Filter (AF), Filter By Quantity (FBQ), Add Component Constraint (ACC), Select Aggregate Variable (SAV), Group By/Having (GH), and Add Triplet (AT)

4.2. Metrics and Protocols

Both the ground-truth (reference) query and the agent-generated query are executed against the live DOREMUS SPARQL endpoint[5]. 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. Being U_{gen} and U_{ref} the sets of URIs retrieved by the generated query and the reference query respectively, the accuracy is calculated as a precision-oriented overlap metric:

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

2. Scalar and Literal Fallback (LLM-Judge): If no URIs are detected in either result set (e.g. for COUNT aggregations or literal filtering), the pipeline falls back to an LLM-as-a-Judge mechanism that assigns a score of 1.0 (Correct), 0.5 (Partially Correct), or 0.0 (Incorrect) based on semantic equivalence.

4.3. Ablation Study

To quantify the impact of specific components within the agentic workflow, we conducted an evaluation using two distinct models: a generic large model using the thinking mode `gpt-4.1` [16] and a specialized model `qwen3-coder:30b` [17].

Figure 2 illustrates the accuracy trajectory as access to specific tools is systematically enabled, revealing critical insights into the architecture's dependencies. The baseline performance (where advanced tools are denied) underscores the insufficiency of unrestrained Explorative-Generative Workflows (marked as **EXP-GEN**), such as those in standard Wikidata MCP implementations.

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 hallucination. The other components significantly improve system accuracy, demonstrating the value of their combination, especially with GPT4.

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 full access to raw logic and graph traversal. Although these tools provide an additional accuracy gain of approximately

[5] https://data.doremus.org/sparql

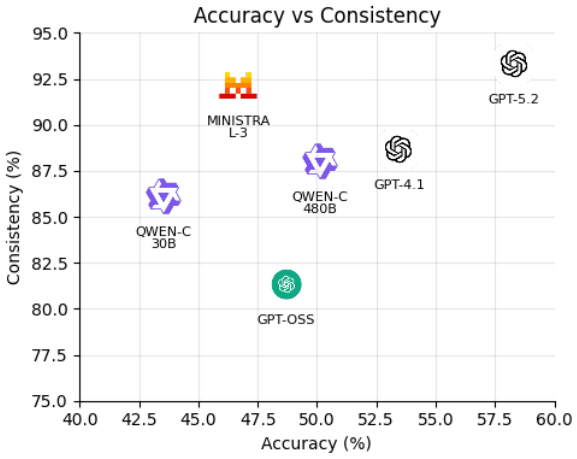


Figure 3: Consistency scores across models

five percentage points with `gpt-4.1`, they reduce performance with `qwen3-coder:30b`. 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.

4.4. Results

We evaluated the following architectures and models in order to cover a broad spectrum in terms of parameter sizes and licensing models:

- Big (1T to 5T parameters): `gpt-5.2-thinking` [18], `gpt-4.1` [16]
- Medium (100B-1T): `qwen3-coder:480b` [17], `gpt-oss:120b-thinking` [19]
- Small (10B-30B): `qwen3-coder:30b` [17], `ministral-3:14b` [20]

To derive a holistic assessment of model quality, we analyze performance across three dimensions: *Reliability*, *Efficiency* and *Reasoning Quality*.

4.4.1. Reliability and Consistency

We measure the **Consistency Score**, the likelihood of the agent converging on the same answer across identical prompts. To estimate run-to-run variability, the metric is calculated over three independent runs per question for each model in the benchmark.

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 3, all models exhibit a high consistency range (82%–93%). This validates the robustness of our architecture, where performance drops are attributable to reasoning limitations (inability to handle complexity) rather than stochastic tool hallucination or syntax errors.

4.4.2. Efficiency

We analyzed the computational cost of agentic retrieval by correlating the average number of tool calls with the total token consumption. High-reasoning generalist models, such as `gpt-4.1`, demonstrate superior efficiency, utilizing a minimal number of tool calls to isolate the correct schema path. However, so-called *thinking models* (e.g. `gpt-5.2`) reveal a distinct trade-off: these architectures prioritize reasoning depth over speed, trading token efficiency for higher accuracy, as shown in Figure 4. This is consistent with findings that reasoning performance scales with inference-time compute [21].

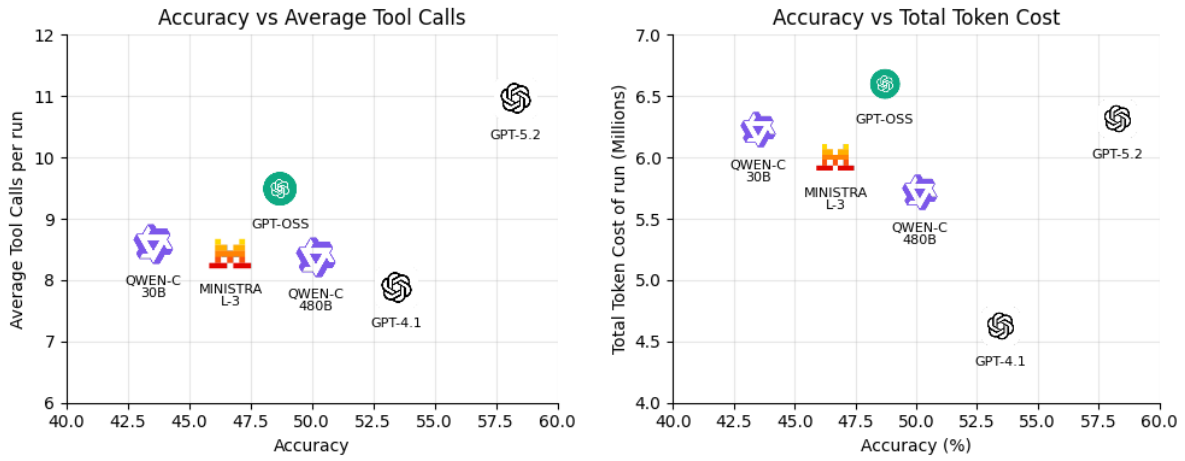


Figure 4: Operational Efficiency Metrics: average tool invocations per run and total token consumption.

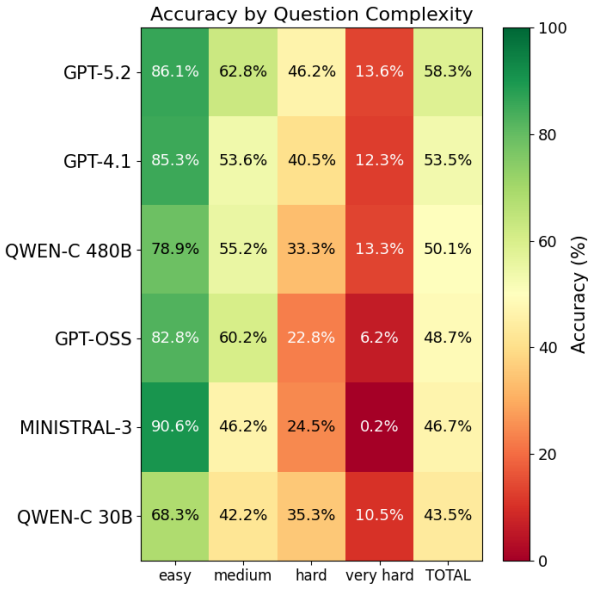


Figure 5: Accuracy across models and query-complexity levels.

Smaller models such as `ministral-3:14b` mitigate this latency through parallel tool invocation, exploring multiple schema hypotheses simultaneously. Conversely, specialized models like `qwen-coder:480b` 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*.

4.4.3. Reasoning Quality

Figure 5 reports performance across the four complexity levels introduced in Section 4.1. Although the best-performing model obtains an overall accuracy of 58.3%, its score decreases from 86.1% on *Easy* questions to 62.8% on *Medium*, 46.2% on *Hard*, and 13.6% on *Very Hard* questions.

Easy questions mainly require selecting a suitable template and adding direct attribute constraints, operations that are strongly supported by the Query Container. Medium questions introduce longer graph paths and interacting constraints, increasing the risk of selecting an incorrect relation or attaching a condition to the wrong variable.

Hard questions additionally require aggregation, conditional logic, or more open-ended graph con-

struction. Their resolution depends not only on valid SPARQL syntax, but also on correct variable scoping and tool sequencing. Very Hard questions involve nested queries or patterns that are not directly supported by the available tools; given the small size of this category, its results primarily indicate the current limitations of the framework.

Overall, differences between models become more pronounced as semantic planning requirements increase. Smaller coding-oriented models remain competitive when instantiating predefined patterns, whereas larger reasoning models more frequently manage interacting constraints and aggregation. For example, for *"Give me the works written for oboe and orchestra after 1900"*, stronger models better distinguish the paths for the solo instrument and ensemble while applying the temporal constraint to the correct variable. The resulting failure patterns are examined in the following section.

4.5. Taxonomy of Failures

We categorized failure modes to isolate the root causes of incorrect generation. Thanks to the architecture's execution-based validation, syntax errors were effectively eliminated. The remaining faults fall into three semantic categories:

1. **Schema Hallucination:** The model invents valid but non-pertinent schema elements, e.g. `foaf:name` instead of `rdfs:label`.
2. **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.
3. **Semantic Drift:** 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*.

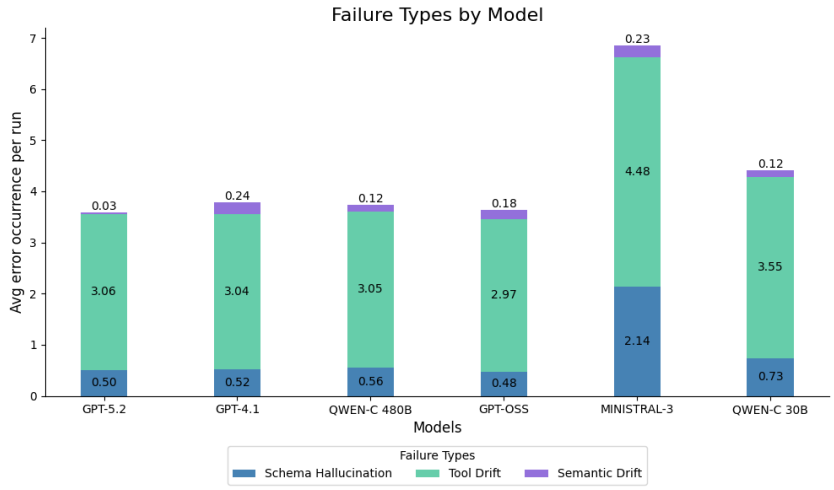


Figure 6: Failure types across different LLMs

As illustrated in Figure 6, the error distribution validates our architectural assumptions regarding stability and intent understanding. Schema hallucinations are minimal for reasoning-capable models. This confirms that the models have successfully internalized the tool definitions. The incidence of Semantic Drift is notably low across all LLMs, that possess a robust understanding of user intent. The few recorded Semantic Drift errors involved format divergence rather than logical failure, such as returning URIs instead of human-readable names.

Consequently, the majority of failures are due to Tool Drift. These 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. 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.

Configuration	Accuracy (%)	Total Cost	Avg 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 2
Ablation study of safety mechanisms on `gpt-4.1`, showing *accuracy*, *total token cost* and *average tool calls* in the different configurations

A distinct outlier in our analysis is `ministral-3:14b`, that exhibits a significantly higher rate of Schema Hallucination. Its *Parallel Tool Invocation* conducts more trials and explores the RDF graph more broadly in less time. However, the model generates a high volume of hypotheses but lacks the reasoning depth to verify them against the ontology, leading to hallucinated parameters despite the availability of correct tools.

4.6. Synergy of Dry-run and Micro-Sampling

To isolate the contribution of our novel safety mechanisms, we conducted an ablation study using the `gpt-4.1` model. We measured performance across four configurations, toggling the *Micro-Sampling* (constrained decision-making) and *Dry-Run* (execution-based validation) features. The results are summarized in Table 2.

Enabling only the Dry-Run mechanism improves accuracy by +5.6% over the baseline, a behavior that mimics the *query correction layer* from FIRESPARQL [22], 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 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.

The fully enabled configuration (**Sampling ON + Dry Run ON**) achieves a peak accuracy of **53.45%**, a +13.7% absolute improvement over the baseline (EXP-GEN). 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, thanks to the complementary nature of the mechanisms: while sampling ensures that the agent makes semantically correct decisions early in the chain, 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 combined configuration uses fewer calls than the baseline and the Micro-Sampling-only configuration, averaging 7.89 calls and indicating that the combined constraints can reduce computational overhead.

4.7. Comparison with GRASP

GRASP [23] is a zero-shot SPARQL generation framework that combines LLM reasoning with dynamic graph exploration. It avoids benchmark-specific fine-tuning or full ontology prompts by using generic functions to search IRIs and literals, execute intermediate queries, and return a final answer or cancellation. This makes GRASP a strong baseline for our setting: it is portable across RDF KGs, interleaves reasoning with execution, and achieves state-of-the-art zero-shot results on Wikidata benchmarks.

From our experiments, GRASP struggles on complex KG such as DOREMUS, as shown in Table 3, primarily due to semantic grounding errors: it often selects incorrect ontology entities or relations, even when the generated SPARQL is syntactically valid. Many runs either fail to produce a final query

Output status	Count	Representative example
Null output	16 / 64 (25%)	*Which works have been written for string quartet?* produced no valid final output because generation timed out.
Empty result	22 / 64 (34.3%)	*Which works have been composed by Mozart?* produced a valid SPARQL query, but execution returned `Got no rows and 1 columns`. We also include cancellations, timeouts, and execution errors in this category, since they produce no usable answer.
Valid result	26 / 64 (40.6%)	*Give me performances between 2000 and 2001* produced a non-empty result with 20 performance–timespan pairs. However, none of the displayed examples fall within 2000–2001, since the query links to timespan nodes but does not filter the actual date values.

Table 3
GRASP output statistics on DOREMUS. *Null output* denotes cases with no valid final output object. *Empty result* denotes cases with no valid answer, including explicit empty executions, cancellations, timeouts, and execution errors. *Valid result* denotes non-empty execution, not necessarily correctness against the gold query.

or generate executable queries whose results are empty, reported as `Got no rows`[6]. For example, for *Which works have been composed by Mozart?*, GRASP searches for labels containing "Mozart" instead of following the composer–work relation. For instrumentation queries such as *Give me the works written for violin, clarinet and piano (strictly)*, it retrieves plausible instrument IRIs but connects them through `efrbroo:R25_performed`, whereas the gold query requires DOREMUS-specific casting and medium-of-performance structures. The valid cases are mostly label-based, such as *Find works with Moonlight' in the title* or *Find artists with Simone' in their name*. Structured questions remain difficult: for *Give me performances between 2000 and 2001*, GRASP links performances to time spans but fails to reach the correct date properties, returning generic performance–time-span pairs. Overall, GRASP retrieves plausible local fragments, whereas our MCP workflow constrains generation toward complete DOREMUS-specific query patterns.

These errors suggest that zero-shot graph exploration is brittle for domain-dense ontologies with indirect modeling conventions. Although GRASP is schema-agnostic, it often composes paths that are locally plausible but globally incorrect, which also makes the resulting queries difficult to explain in terms of the target ontology. By contrast, our MCP approach is less open-ended but more reliable: it avoids unconstrained SPARQL generation by using the Query Container to incrementally assemble queries, initializes query structure through KG-specific templates in `build_query`, and refines them through parametric tools with schema-validation and execution-preserving updates.

5. Conclusions and Future Work

In this work, we presented a methodology based on the Model Context Protocol (MCP) designed to facilitate structured agentic retrieval over Knowledge Graphs. By moving away from the standard *Explorative-Generative* paradigm, we proposed a hybrid agentic framework focused on controlled query construction and structural consistency. Our approach leverages a pattern-based query construction strategy, utilizing a suite of low-granularity modular tools to assemble SPARQL queries iteratively within a protected *Query Container*. We demonstrated that this architecture effectively balances the

[6]For this reason, we report GRASP outcomes by execution status. In fact, an answer-overlap score would have been undefined or uninformative. As a result, this analysis is complementary to, rather than directly comparable with, the answer-level evaluation of our framework.

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 Knowledge Graph, several avenues remain for extending its versatility and reasoning depth:

- First, we aim to validate the generalization capabilities of our pattern-based tools. While demonstrating its performance on the DOREMUS KG is particularly interesting because of its challenging ontological structure (Section 4.1), we aim to test the framework against other Knowledge Graphs in different domains. Although each template is lightweight, adapting the framework to a new Knowledge Graph still requires manual configuration. Future work will investigate automatic template generation.
- 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, treating a sub-query as a distinct artifact that can be generated, validated, and then injected into the parent query as a filter condition or a `MINUS` block.
- Our error analysis identified *Tool Drift* as the persistent bottleneck, where models resort to generic traversal tools (`add_triplet`) rather than specific patterns, as shown in Figure 6. Future work will focus on optimizing the semantic definitions within the MCP `Tools` schema.
- The current evaluation relies on a precision-oriented metric, which does not fully capture retrieval completeness. Future work will complement it with recall and F_1, and additionally include ontology-based variants such as type-aware or taxonomic-distance metrics. Finally, will calibrate the LLM-based evaluation of scalar and literal answers against human annotations or deterministic comparison methods.

Supplemental Material Statement: The dataset and source code needed to reproduce the results presented in this paper are available at https://github.com/SimoneFassio/DOREMUS_MCP

Acknowledgments

This work was supported by the French Public Investment Bank (Bpifrance) i-Demo program within the LettRAGraph project (Grant ID DOS0256163/00).

Author Contributions

Simone Fassio: Methodology, Software, Writing – review & editing; **Dario Gosmar:** Methodology, Software, Writing – review & editing; **Fanfu Wei:** Software, Writing – review & editing; **Thibault Ehrhart:** Supervision, Software; **Pasquale Lisena:** Supervision, Methodology, Writing – original draft; **Raphael Troncy:** Supervision, Writing – review & editing;

Declaration of use of Generative AI

During the preparation of this work, the authors used GPT5.6 in order to: grammar and spell check, paraphrase and reword. After using this tool, the authors reviewed and edited the content as needed and take full responsibility for the publication's content.

References

[1] C. Ma, Y. Chen, T. Wu, A. Khan, H. Wang, Large Language Models Meet Knowledge Graphs for Question Answering: Synthesis and Opportunities, in: Conference on Empirical Methods in

Natural Language Processing (EMNLP), Association for Computational Linguistics, Suzhou, China, 2025, pp. 24578–24597. doi:10.18653/v1/2025.emnlp-main.1249.

[2] S. Y. B. Ho, A. Coles, E. Larsson, E. Marshall, N. Bodenstab, P. Vozila, SchemaRAG: Dynamic Large Schema Reduction for LLM-driven Structured Information Extraction, in: The 64th Annual Meeting of the Association for Computational Linguistics – Industry Track, 2026. URL: https://openreview.net/forum?id=AAeD8y6ibk.

[3] X. Hou, Y. Zhao, S. Wang, H. Wang, Model Context Protocol (MCP): Landscape, Security Threats, and Future Research Directions, ACM Transactions on Software Engineering and Methodology (2026). doi:10.1145/3796519.

[4] A. Madaan, N. Tandon, P. Gupta, S. Hallinan, L. Gao, S. Wiegreffe, U. Alon, N. Dziri, S. Prabhumoye, Y. Yang, S. Gupta, B. P. Majumder, K. Hermann, S. Welleck, A. Yazdanbakhsh, P. Clark, SELF-REFINE: iterative refinement with self-feedback, 2023.

[5] M. Achichi, P. Lisena, K. Todorov, R. Troncy, J. Delahousse, DOREMUS: A Graph of Linked Musical Works, in: 17th International Semantic Web Conference (ISWC), Springer-Verlag, 2018, pp. 3––19. doi:10.1007/978-3-030-00668-6_1.

[6] C. Ma, S. Chakrabarti, A. Khan, B. Molnár, Knowledge Graph-based Retrieval-Augmented Generation for Schema Matching, 2025. URL: https://arxiv.org/abs/2501.08686. arXiv:2501.08686.

[7] G. Piao, M. Mountantonakis, P. Papadakos, P. Sonawane, A. O'Mahony, Toward Exploring Knowledge Graphs with LLMs, in: International Conference on Semantic Systems (SEMANTiCS), Amsterdam, Netherlands, 2024. URL: https://ceur-ws.org/Vol-3759/paper1.pdf.

[8] A. Perevalov, A. Both, Text-to-SPARQL Goes Beyond English: Multilingual Question Answering Over Knowledge Graphs through Human-Inspired Reasoning, in: First International TEXT2SPARQL Challenge at Text2KG, Portorož, Slovenia, 2025. URL: https://ceur-ws.org/Vol-4094/paper6.pdf.

[9] F. Brei, L. Bühmann, J. Frey, D. Gerber, L.-P. Meyer, C. Stadler, K. Bulert, ARUQULA – An LLM based Text2SPARQL Approach using ReAct and Knowledge Graph Exploration Utilities, in: First International TEXT2SPARQL Challenge at Text2KG, Portorož, Slovenia, 2025. URL: https://ceur-ws.org/Vol-4094/paper4.pdf.

[10] R. Taelman, J. De Smet, Comunica MCP SPARQL: Improving the Accuracy of AI Agents using SPARQL-based Access to Decentralized Knowledge Graphs, in: Extended Semantic Web Conference, Posters and Demos track, Dubrovnik, Croatia, 2026.

[11] T. Francart, MCP protocol over SPARQL endpoints, https://www.sparna.fr/en/posts/mcp-protocol-over-sparql-endpoints/, 2026. Accessed: 2026-05-07.

[12] P. W. Rose, B. M. Good, C. A. Nelson, A. M. Saravia-Butler, Y. Shi, A. I. Su, S. E. Baranzini, MCP Server Proto-OKN, https://github.com/sbl-sdsc/mcp-proto-okn, 2025. Software.

[13] M. Doerr, C.-E. Ore, S. Stead, The CIDOC conceptual reference model: a new standard for knowledge sharing, in: Tutorials, Posters, Panels and Industrial Contributions at the 26th International Conference on Conceptual Modeling - Volume 83, ER '07, Australian Computer Society, Inc., AUS, 2007, p. 51–56.

[14] M. Doerr, C. Bekiari, P. LeBoeuf, FRBRoo, a conceptual model for performing arts, in: 2008 Annual Conference of CIDOC, Athens, 2008, pp. 15–18.

[15] P. Lisena, R. Troncy, Representing Complex Knowledge for Exploration and Recommendation: The Case of Classical Music Information, in: Applications and Practices in Ontology Design, Extraction, and Reasoning, volume 49 of *Studies on the Semantic Web Series (SSWS)*, IOSPress, 2020, pp. 107–123. doi:10.3233/SSW200038.

[16] OpenAI, J. Achiam, S. Adler, S. Agarwal, L. Ahmad, I. Akkaya, F. L. Aleman, D. Almeida, J. Altenschmidt, S. Altman, S. Anadkat, R. Avila, I. Babuschkin, S. Balaji, V. Balcom, P. Baltescu, H. Bao, M. Bavarian, J. Belgum, I. Bello, J. Berdine, G. Bernadett-Shapiro, C. Berner, L. Bogdonoff, O. Boiko, M. Boyd, A.-L. Brakman, G. Brockman, T. Brooks, M. Brundage, K. Button, T. Cai, R. Campbell, A. Cann, B. Carey, C. Carlson, R. Carmichael, B. Chan, C. Chang, F. Chantzis, D. Chen, S. Chen, R. Chen, J. Chen, M. Chen, B. Chess, C. Cho, C. Chu, H. W. Chung, D. Cummings, J. Currier, Y. Dai, C. Decareaux, T. Degry, N. Deutsch, D. Deville, A. Dhar, D. Dohan, S. Dowling, S. Dunning,

A. Ecoffet, A. Eleti, T. Eloundou, D. Farhi, L. Fedus, N. Felix, S. P. Fishman, J. Forte, I. Fulford, L. Gao, E. Georges, C. Gibson, V. Goel, T. Gogineni, G. Goh, R. Gontijo-Lopes, J. Gordon, M. Grafstein, S. Gray, R. Greene, J. Gross, S. S. Gu, Y. Guo, C. Hallacy, J. Han, J. Harris, Y. He, M. Heaton, J. Heidecke, C. Hesse, A. Hickey, W. Hickey, P. Hoeschele, B. Houghton, K. Hsu, S. Hu, X. Hu, J. Huizinga, S. Jain, S. Jain, J. Jang, A. Jiang, R. Jiang, H. Jin, D. Jin, S. Jomoto, B. Jonn, H. Jun, T. Kaftan, Łukasz Kaiser, A. Kamali, I. Kanitscheider, N. S. Keskar, T. Khan, L. Kilpatrick, J. W. Kim, C. Kim, Y. Kim, J. H. Kirchner, J. Kiros, M. Knight, D. Kokotajlo, Łukasz Kondraciuk, A. Kondrich, A. Konstantinidis, K. Kosic, G. Krueger, V. Kuo, M. Lampe, I. Lan, T. Lee, J. Leike, J. Leung, D. Levy, C. M. Li, R. Lim, M. Lin, S. Lin, M. Litwin, T. Lopez, R. Lowe, P. Lue, A. Makanju, K. Malfacini, S. Manning, T. Markov, Y. Markovski, B. Martin, K. Mayer, A. Mayne, B. McGrew, S. M. McKinney, C. McLeavey, P. McMillan, J. McNeil, D. Medina, A. Mehta, J. Menick, L. Metz, A. Mishchenko, P. Mishkin, V. Monaco, E. Morikawa, D. Mossing, T. Mu, M. Murati, O. Murk, D. Mély, A. Nair, R. Nakano, R. Nayak, A. Neelakantan, R. Ngo, H. Noh, L. Ouyang, C. O'Keefe, J. Pachocki, A. Paino, J. Palermo, A. Pantuliano, G. Parascandolo, J. Parish, E. Parparita, A. Passos, M. Pavlov, A. Peng, A. Perelman, F. de Avila Belbute Peres, M. Petrov, H. P. de Oliveira Pinto, Michael, Pokorny, M. Pokrass, V. H. Pong, T. Powell, A. Power, B. Power, E. Proehl, R. Puri, A. Radford, J. Rae, A. Ramesh, C. Raymond, F. Real, K. Rimbach, C. Ross, B. Rotsted, H. Roussez, N. Ryder, M. Saltarelli, T. Sanders, S. Santurkar, G. Sastry, H. Schmidt, D. Schnurr, J. Schulman, D. Selsam, K. Sheppard, T. Sherbakov, J. Shieh, S. Shoker, P. Shyam, S. Sidor, E. Sigler, M. Simens, J. Sitkin, K. Slama, I. Sohl, B. Sokolowsky, Y. Song, N. Staudacher, F. P. Such, N. Summers, I. Sutskever, J. Tang, N. Tezak, M. B. Thompson, P. Tillet, A. Tootoonchian, E. Tseng, P. Tuggle, N. Turley, J. Tworek, J. F. C. Uribe, A. Vallone, A. Vijayvergiya, C. Voss, C. Wainwright, J. J. Wang, A. Wang, B. Wang, J. Ward, J. Wei, C. Weinmann, A. Welihinda, P. Welinder, J. Weng, L. Weng, M. Wiethoff, D. Willner, C. Winter, S. Wolrich, H. Wong, L. Workman, S. Wu, J. Wu, M. Wu, K. Xiao, T. Xu, S. Yoo, K. Yu, Q. Yuan, W. Zaremba, R. Zellers, C. Zhang, M. Zhang, S. Zhao, T. Zheng, J. Zhuang, W. Zhuk, B. Zoph, GPT-4 Technical Report, 2024. URL: https://arxiv.org/abs/2303.08774. arXiv:2303.08774.

[17] R. Cao, M. Chen, J. Chen, Z. Cui, Y. Feng, B. Hui, Y. Jing, K. Li, M. Li, J. Lin, Z. Ma, K. Shum, X. Wang, J. Wei, J. Yang, J. Zhang, L. Zhang, Z. Zhang, W. Zhao, F. Zhou, Qwen3-Coder-Next Technical Report, arXiv preprint arXiv:2603.00729 (2026).

[18] A. Singh, A. Fry, A. Perelman, A. Tart, A. Ganesh, A. El-Kishky, A. McLaughlin, A. Low, A. Ostrow, A. Ananthram, A. Nathan, A. Luo, A. Helyar, A. Madry, A. Efremov, A. Spyra, A. Baker-Whitcomb, A. Beutel, A. Karpenko, A. Makelov, A. Neitz, A. Wei, A. Barr, A. Kirchmeyer, A. Ivanov, A. Christakis, A. Gillespie, A. Tam, A. Bennett, A. Wan, A. Huang, A. M. Sandjideh, A. Yang, A. Kumar, A. Saraiva, A. Vallone, A. Gheorghe, A. G. Garcia, A. Braunstein, A. Liu, A. Schmidt, A. Mereskin, A. Mishchenko, A. Applebaum, A. Rogerson, A. Rajan, A. Wei, A. Kotha, A. Srivastava, A. Agrawal, A. Vijayvergiya, A. Tyra, A. Nair, A. Nayak, B. Eggers, B. Ji, B. Hoover, B. Chen, B. Chen, B. Barak, B. Minaiev, B. Hao, B. Baker, B. Lightcap, B. McKinzie, B. Wang, B. Quinn, B. Fioca, B. Hsu, B. Yang, B. Yu, B. Zhang, B. Brenner, C. R. Zetino, C. Raymond, C. Lugaresi, C. Paz, C. Hudson, C. Whitney, C. Li, C. Chen, C. Cole, C. Voss, C. Ding, C. Shen, C. Huang, C. Colby, C. Hallacy, C. Koch, C. Lu, C. Kaplan, C. Kim, C. Minott-Henriques, C. Frey, C. Yu, C. Czarnecki, C. Reid, C. Wei, C. Decareaux, C. Scheau, C. Zhang, C. Forbes, D. Tang, D. Goldberg, D. Roberts, D. Palmie, D. Kappler, D. Levine, D. Wright, D. Leo, D. Lin, D. Robinson, D. Grabb, D. Chen, D. Lim, D. Salama, D. Bhattacharjee, D. Tsipras, D. Li, D. Yu, D. Strouse, D. Williams, D. Hunn, E. Bayes, E. Arbus, E. Akyurek, E. Y. Le, E. Widmann, E. Yani, E. Proehl, E. Sert, E. Cheung, E. Schwartz, E. Han, E. Jiang, E. Mitchell, E. Sigler, E. Wallace, E. Ritter, E. Kavanaugh, E. Mays, E. Nikishin, F. Li, F. P. Such, F. de Avila Belbute Peres, F. Raso, F. Bekerman, F. Tsimpourlas, F. Chantzis, F. Song, F. Zhang, G. Raila, G. McGrath, G. Briggs, G. Yang, G. Parascandolo, G. Chabot, G. Kim, G. Zhao, G. Valiant, G. Leclerc, H. Salman, H. Wang, H. Sheng, H. Jiang, H. Wang, H. Jin, H. Sikchi, H. Schmidt, H. Aspegren, H. Chen, H. Qiu, H. Lightman, I. Covert, I. Kivlichan, I. Silber, I. Sohl, I. Hammoud, I. Clavera, I. Lan, I. Akkaya, I. Kostrikov, I. Kofman, I. Etinger, I. Singal, J. Hehir, J. Huh, J. Pan, J. Wilczynski, J. Pachocki, J. Lee, J. Quinn, J. Kiros, J. Kalra, J. Samaroo, J. Wang, J. Wolfe, J. Chen, J. Wang, J. Harb, J. Han, J. Wang, J. Zhao, J. Chen, J. Yang, J. Tworek, J. Chand, J. Landon, J. Liang, J. Lin, J. Liu, J. Wang, J. Tang,

J. Yin, J. Jang, J. Morris, J. Flynn, J. Ferstad, J. Heidecke, J. Fishbein, J. Hallman, J. Grant, J. Chien, J. Gordon, J. Park, J. Liss, J. Kraaijeveld, J. Guay, J. Mo, J. Lawson, J. McGrath, J. Vendrow, J. Jiao, J. Lee, J. Steele, J. Wang, J. Mao, K. Chen, K. Hayashi, K. Xiao, K. Salahi, K. Wu, K. Sekhri, K. Sharma, K. Singhal, K. Li, K. Nguyen, K. Gu-Lemberg, K. King, K. Liu, K. Stone, K. Yu, K. Ying, K. Georgiev, K. Lim, K. Tirumala, K. Miller, L. Ahmad, L. Lv, L. Clare, L. Fauconnet, L. Itow, L. Yang, L. Romaniuk, L. Anise, L. Byron, L. Pathak, L. Maksin, L. Lo, L. Ho, L. Jing, L. Wu, L. Xiong, L. Mamitsuka, L. Yang, L. McCallum, L. Held, L. Bourgeois, L. Engstrom, L. Kuhn, L. Feuvrier, L. Zhang, L. Switzer, L. Kondraciuk, L. Kaiser, M. Joglekar, M. Singh, M. Shah, M. Stratta, M. Williams, M. Chen, M. Sun, M. Cayton, M. Li, M. Zhang, M. Aljubeh, M. Nichols, M. Haines, M. Schwarzer, M. Gupta, M. Shah, M. Y. Guan, M. Huang, M. Dong, M. Wang, M. Glaese, M. Carroll, M. Lampe, M. Malek, M. Sharman, M. Zhang, M. Wang, M. Pokrass, M. Florian, M. Pavlov, M. Wang, M. Chen, M. Wang, M. Feng, M. Bavarian, M. Lin, M. Abdool, M. Rohaninejad, N. Soto, N. Staudacher, N. LaFontaine, N. Marwell, N. Liu, N. Preston, N. Turley, N. Ansman, N. Blades, N. Pancha, N. Mikhaylin, N. Felix, N. Handa, N. Rai, N. Keskar, N. Brown, O. Nachum, O. Boiko, O. Murk, O. Watkins, O. Gleeson, P. Mishkin, P. Lesiewicz, P. Baltescu, P. Belov, P. Zhokhov, P. Pronin, P. Guo, P. Thacker, Q. Liu, Q. Yuan, Q. Liu, R. Dias, R. Puckett, R. Arora, R. T. Mullapudi, R. Gaon, R. Miyara, R. Song, R. Aggarwal, R. Marsan, R. Yemiru, R. Xiong, R. Kshirsagar, R. Nuttall, R. Tsiupa, R. Eldan, R. Wang, R. James, R. Ziv, R. Shu, R. Nigmatullin, S. Jain, S. Talaie, S. Altman, S. Arnesen, S. Toizer, S. Toyer, S. Miserendino, S. Agarwal, S. Yoo, S. Heon, S. Ethersmith, S. Grove, S. Taylor, S. Bubeck, S. Banesiu, S. Amdo, S. Zhao, S. Wu, S. Santurkar, S. Zhao, S. R. Chaudhuri, S. Krishnaswamy, Shuaiqi, Xia, S. Cheng, S. Anadkat, S. P. Fishman, S. Tobin, S. Fu, S. Jain, S. Mei, S. Egoian, S. Kim, S. Golden, S. Mah, S. Lin, S. Imm, S. Sharpe, S. Yadlowsky, S. Choudhry, S. Eum, S. Sanjeev, T. Khan, T. Stramer, T. Wang, T. Xin, T. Gogineni, T. Christianson, T. Sanders, T. Patwardhan, T. Degry, T. Shadwell, T. Fu, T. Gao, T. Garipov, T. Sriskandarajah, T. Sherbakov, T. Korbak, T. Kaftan, T. Hiratsuka, T. Wang, T. Song, T. Zhao, T. Peterson, V. Kharitonov, V. Chernova, V. Kosaraju, V. Kuo, V. Pong, V. Verma, V. Petrov, W. Jiang, W. Zhang, W. Zhou, W. Xie, W. Zhan, W. McCabe, W. DePue, W. Ellsworth, W. Bain, W. Thompson, X. Chen, X. Qi, X. Xiang, X. Shi, Y. Dubois, Y. Yu, Y. Khakbaz, Y. Wu, Y. Qian, Y. T. Lee, Y. Chen, Y. Zhang, Y. Xiong, Y. Tian, Y. Cha, Y. Bai, Y. Yang, Y. Yuan, Y. Li, Y. Zhang, Y. Yang, Y. Jin, Y. Jiang, Y. Wang, Y. Wang, Y. Liu, Z. Stubenvoll, Z. Dou, Z. Wu, Z. Wang, OpenAI GPT-5 System Card, 2026. URL: https://arxiv.org/abs/2601.03267. arXiv:2601.03267.

[19] OpenAI, gpt-oss-120b & gpt-oss-20b Model Card, 2025. URL: https://arxiv.org/abs/2508.10925. arXiv:2508.10925.

[20] A. H. Liu, K. Khandelwal, S. Subramanian, V. Jouault, A. Rastogi, A. Sadé, A. Jeffares, A. Jiang, A. Cahill, A. Gavaudan, A. Sablayrolles, A. Héliou, A. You, A. Ehrenberg, A. Lo, A. Eliseev, A. Calvi, A. Sooriyarachchi, B. Bout, B. Rozière, B. De Monicault, C. Lanfranchi, C. Barreau, C. Courtot, D. Grattarola, D. Dabert, D. de las Casas, E. Chane-Sane, F. Ahmed, G. Berrada, G. Ecrepont, G. Guinet, G. Novikov, G. Kunsch, G. Lample, G. Martin, G. Gupta, J. Ludziejewski, J. Rute, J. Studnia, J. Amar, J. Delas, J. Somerville Roberts, K. Yadav, K. Chandu, K. Jain, L. Aitchison, L. Fainsin, L. Blier, L. Zhao, L. Martin, L. Saulnier, L. Gao, M. Buyl, M. Jennings, M. Pellat, M. Prins, M. Poirée, M. Guillaumin, M. Dinot, M. Futeral, M. Darrin, M. Augustin, M. Chiquier, M. Schimpf, N. Grinsztajn, N. Gupta, N. Raghuraman, O. Bousquet, O. Duchenne, P. Wang, P. von Platen, P. Jacob, P. Wambergue, P. Kurylowicz, P. R. Muddireddy, P. Chagniot, P. Stock, P. Agrawal, Q. Torroba, R. Sauvestre, R. Soletskyi, R. Menneer, S. Vaze, S. Barry, S. Gandhi, S. Waghjale, S. Gandhi, S. Ghosh, S. Mishra, S. Aithal, S. Antoniak, T. Le Scao, T. Cachet, T. S. Sorg, T. Lavril, T. Nait Saada, T. Chabal, T. Foubert, T. Robert, T. Wang, T. Lawson, T. Bewley, T. Bewley, T. Edwards, U. Jamil, U. Tomasini, V. Nemychnikova, V. Phung, V. Maladière, V. Richard, W. Bouaziz, W.-D. Li, W. Marshall, X. Li, X. Yang, Y. El Ouahidi, Y. Wang, Y. Tang, Z. Ramzi, Ministral 3, arXiv e-prints (2026). doi:10.48550/arXiv.2601.08584. arXiv:2601.08584.

[21] S. Yao, D. Yu, J. Zhao, I. Shafran, T. L. Griffiths, Y. Cao, K. Narasimhan, Tree of thoughts: deliberate problem solving with large language models, in: 37th International Conference on Neural Information Processing Systems (NeurIPS), Curran Associates Inc., Red Hook, NY, USA, 2023.

[22] X. Pan, V. de Boer, J. van Ossenbruggen, FIRESPARQL: A LLM-Based Framework for SPARQL

Query Generation over Scholarly Knowledge Graphs, in: Proceedings of the 17th International Joint Conference on Knowledge Discovery, Knowledge Engineering and Knowledge Management, International Joint Conference on Knowledge Discovery, Knowledge Engineering and Knowledge Management, IC3K - Proceedings, Science and Technology Publications, Lda, 2025, pp. 123–134. doi:10.5220/0013774000004000.

[23] S. Walter, H. Bast, GRASP: Generic Reasoning and SPARQL Generation Across Knowledge Graphs, in: 24th International Semantic Web Conference (ISWC), Springer, Nara, Japan, 2025, pp. 271–289. doi:10.1007/978-3-032-09527-5_15.