Natural language querying of RDF-based process models through an MCP interaction using Codex and GraphDB tools

Damaris – Naomi Dolha[1,*]

[1]Babeș-Bolyai University, 58–60 Teodor Mihali St., Cluj-Napoca 400591, România

Abstract

A process model repository annotated with execution information – either collected from past executions or stored for simulation purposes – can tell a rich story about how work unfolds, who carries it out, what it uses or produces, how long it may take and what it may cost. This paper reports a domain-specific evaluation of how OpenAI's agent (i.e., Codex), using gpt-5.6-Sol, works with GraphDB's tools through Model Context Protocol (MCP) to ground their answers in retrieved RDF representations of annotated business process models. Codex acts as the client for the built-in GraphDB MCP server containing RDF serializations of BPMN and Working Environment models from the Bee-Up modeling tool. Repeated, independent runs use a managerially phrased catalog of questions covering element identification and containment, activity order and possible routes, organizational information and modeled performance, document identification and use, participant communication and potential weaknesses. An automated evaluator compares the generated structured answers with a manually constructed gold standard using precision, recall and F_1. Beyond the scores, the analysis traces tool selection, repository querying and answer formulation to identify possible sources of errors and run-to-run variations.

Keywords

Business Process Management, Bee-Up, Knowledge Graph, Natural Language Querying, MCP

CEUR-WS.org/Vol-4313/kgmdse-paper4.pdf

1. Introduction

Interrelated activities, events and decision points are treated in Business Process Management (BPM) as elements to be discovered, analyzed, (re)designed, implemented, monitored and controlled across the BPM lifecycle [1]. Business Process Model and Notation[1] (BPMN), a widely used standard for business process modeling, balances accessibility with technical precision. As process model repositories grew, the challenge expanded from representing models to meaningfully accessing their knowledge [2]. Consequently, process querying became a distinct BPM research line [3].

What has changed more recently was not the need to query processes, but the interface through which process knowledge is accessed, particularly by users who are less comfortable with formal query languages. Earlier work had already recognized this accessibility problem [4] before the emergence of Large Language Models (LLMs). LLMs have since helped make natural language interfaces more practical through process-aware question answering [5] and tool use [6], where agents can invoke external functions. Further on, the Model Context Protocol[2] (MCP) standardizes connections between Artificial Intelligence (AI) agent clients and tool-providing servers. This progression creates a useful opening, although not an entirely untouched one. For instance, existing work evaluates tool coordination across a large multi-server MCP ecosystem [7] and applies MCP for natural language access to biological and biomedical knowledge graphs (KGs) represented in Resource Description Framework (RDF) [8].

The present study takes this direction into a different field, asking *how reliably an LLM-based agent*

Knowledge Graphs and Model-driven Systems Engineering (KGMDSE), in ISWC 2026 Workshops Joint Proceedings, October 25–26, 2026, Bari, Italy

*Corresponding author.

damaris.dolha@econ.ubbcluj.ro (D. -. N. Dolha)

0009-0001-7695-4383 (D. -. N. Dolha)

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

[1]https://www.bpmn.org

[2]https://www.anthropic.com/news/model-context-protocol

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

published 2026-11-05

can select among tools exposed by GraphDB[3] through a single MCP server, to answer a catalog spanning diverse categories of business process questions. The catalog adapts [9]'s business question frameworks and process querying constructs into managerially phrased questions for a static, design-time repository. It extends beyond the activity and direct flow presence checks in [10]'s quantitative evaluation, to multi-step relations and links crossing subprocess or model boundaries. This scope is also motivated by [11]'s case study, where two earlier LLMs correctly identified directly connected tasks, but were less reliable answering questions involving specific task orders or decision points. To evaluate business process querying in scenarios involving more complex process relations, this study uses OpenAI's Codex[4], under specific instructions and configured with gpt-5.6-Sol[5], as an MCP client connected to Ontotext GraphDB[6]. The graph repository combines RDF serializations of BPMN and Working Environment models, supporting six managerially oriented question families evaluated over repeated runs.

Section 2 surveys the relevant literature on business process querying, Section 3 introduces the query catalog posed on the model repository described in Section 4, Section 5 highlights the experimental setting, Section 6 presents representative query examples and discusses study-specific considerations before concluding the paper with Section 7, underlining limitations and future directions sketches.

2. From formal languages to LLM-mediated process querying

Growing business process model repositories made retrieving and reusing relevant models rather difficult [2], but encouraged the development of formal query languages such as BP-QL [3], APQL [12], BPMN-Q [13] and PQL [14], to name a few. Then the limits of structural querying further encouraged approaches that considered the model elements' meanings and connections to the domain knowledge through semantic matching [15], labeled property graphs [16] or semantic enrichment of enterprise models [17], while graph transformations made BPMN knowledge available outside the modeling environment [18, 19]. However, formal querying requires repository knowledge that business users may lack. LLMs offered a way to lower this barrier through natural language interaction across the BPM lifecycle [20], but the results remain uneven. For instance, studies using BPMN XML, JSON or image exports and earlier LLMs reported misinterpreted dependencies, omitted activities and confused BPMN elements [11, 21]. Other approaches introduced intermediate parsing or retrieval steps. Comparing rule-based, gpt-4-based and hybrid parsing, [5] found that combining rules with gpt-4's flexibility reduced failures, although the hybrid approach could not detect incomplete or incorrect rule matches. A framework retrieving BPMN fragments from a vector database quantitatively evaluated binary activity and direct flow questions; broader assessments were qualitative, with examples suggesting ordering and parallel path errors [10]. Outside BPMN, an adapted GraphRAG pipeline was evaluated against SPARQL[7] results on biomedical KGs using questions that verbalized triple patterns with exact entity and relationship labels. Retrieval sometimes omitted relationships and supplied incomplete context, particularly when finding subjects from known objects [22]. These findings motivate exploring other combinations of existing technologies, focusing on how agents can access knowledge using different functions. MCP provides a standard interface through which AI agents can access these tools and choose which to use – although multi-server experiments report difficulties with tool selection and multi-step planning [7]. [8] improved answer completeness and precision over an unaided LLM through MCP-based access to RDF databases, although questions requiring summaries across databases benefited less. Building on these approaches, this study evaluates whether a single AI agent can select among several GraphDB tools exposed through one MCP server to answer managerially phrased questions – including question types that challenged earlier LLM approaches – about linked RDF representations of static BPMN and Working Environment models.

[3] https://graphdb.ontotext.com
[4] https://openai.com/codex/
[5] https://openai.com/index/previewing-gpt-5-6-sol/
[6] https://graphwise.ai/blog/the-power-of-model-context-protocol-using-natural-language-to-query-graphdb/#what-is-mcp-and-why-does-it-matter
[7] https://www.w3.org/TR/sparql11-query/

3. Query catalog: construction and scope

The work in [9] maps BPM professional profiles to recurring query interests and provides query templates, distinguishing performance- and root-cause-oriented interests from compliance questions concerning temporal constraints, activity occurrence and action order, among others. Adapting this framework was necessary, because this case study uses a static, design-time RDF snapshot rather than process execution data. Bee-Up[8] 1.7, an ADOxx[9]-based modeling environment, was used to create rather than enact or monitor the models. Nevertheless, BPMN models can represent planned or execution-derived information through links to Working Environment models that describe organizational structures and responsibilities, and through task-level time and cost attributes, all queryable after the RDF export. Accordingly, the catalog asks what exists, can occur, is assigned or is expected in terms of time and costs within the study's scope. The templates were instantiated with repository-specific entities and organized into six categories, with some representative studies cited where their query constructs or domain concepts informed particular questions (Table 1).

Table 1
Query catalog

Category	Representative questions	Querying approaches in prior work
Process element identification and containment	What tasks are involved in each (sub)process? How is each (sub)process initiated/concluded? Does document X exist?	Checks activity or event occurrence [23]. Retrieves models, process fragments and elements by structure, type, name or attributes [10, 24]. Finds activities nested within subprocesses [4].
Order of actions and possible routes	What must be completed right before task T can begin? Give me the possible outcomes of decision D. What other tasks can be performed in parallel with T?	Checks whether activities or events must occur, precede, follow, co-occur, exclude one another or run in parallel [14, 12, 23]. Retrieves direct connections, paths of any length and process fragments [4, 25, 26, 27], including paths between A to B without visiting C [13].
Resource allocation and modeled performance	Who or what is responsible for task T? Which tasks are assigned to employee R1 and which to employee R2? Which tasks are expected to take longer than n hour(s)?	Examines task assignment under role and capability constraints, considering workload, cost or duration [28, 29]. Connects process and organizational models to examine relationships between responsible individuals [30]. Execution-data queries additionally examine observed durations [9].
Document identification and use	What produces/uses document D? What documents are produced by task T? What documents are never used?	Traces data across process executions, identifying data objects affected by process runs during a selected period [9].
Communication between participants	What information must participant P1 provide to participant P2? At what points in the process does participant P1 interact with participant P3? What are the messages sent by participant P1 to participant P2 after task T?	Examines how participants exchange messages and whether an exchange is followed by a response [31]. Semantic querying also retrieves tasks of type `send` and participants involved in modeled activities [32].
Potential process weaknesses and improvement opportunities	Where does the process repeat the work? Which process outcomes depend on awaiting for participant P's instruction? Where does the process lack a fallback if another party does not respond?	Turns descriptions of parts of the process – that hinders its execution or that could be modeled differently – into structural patterns and retrieves matching model fragments [30].

[8] https://bee-up.omilab.org/activities/bee-up/
[9] https://www.omilab.org/adoxx/

4. Model repository

The evaluation used a repository of synthetic models depicting a simplified judicial collaboration among a client, law firm and court (Fig. 1). Created in Bee-Up, its main BPMN collaboration diagram and six referenced subprocess diagrams contain 52 tasks, 26 events, 13 gateways and six data objects. Two Working Environment models add organizational content, with modeled RACI assignments covering 50 tasks; all 52 tasks carry execution time and cost. The TriG export preserves one named graph per model and a metadata graph containing the ADOxx metamodel, as well as model types and names, allowing internal and cross-model relations to be queried together. After the import into the local GraphDB 11.4 repository, `bpmn_judicial`, a SPARQL delete operation removed selected Bee-Up export statements irrelevant to the experiments (e.g., non-analytical predicates or generic ADOxx types).

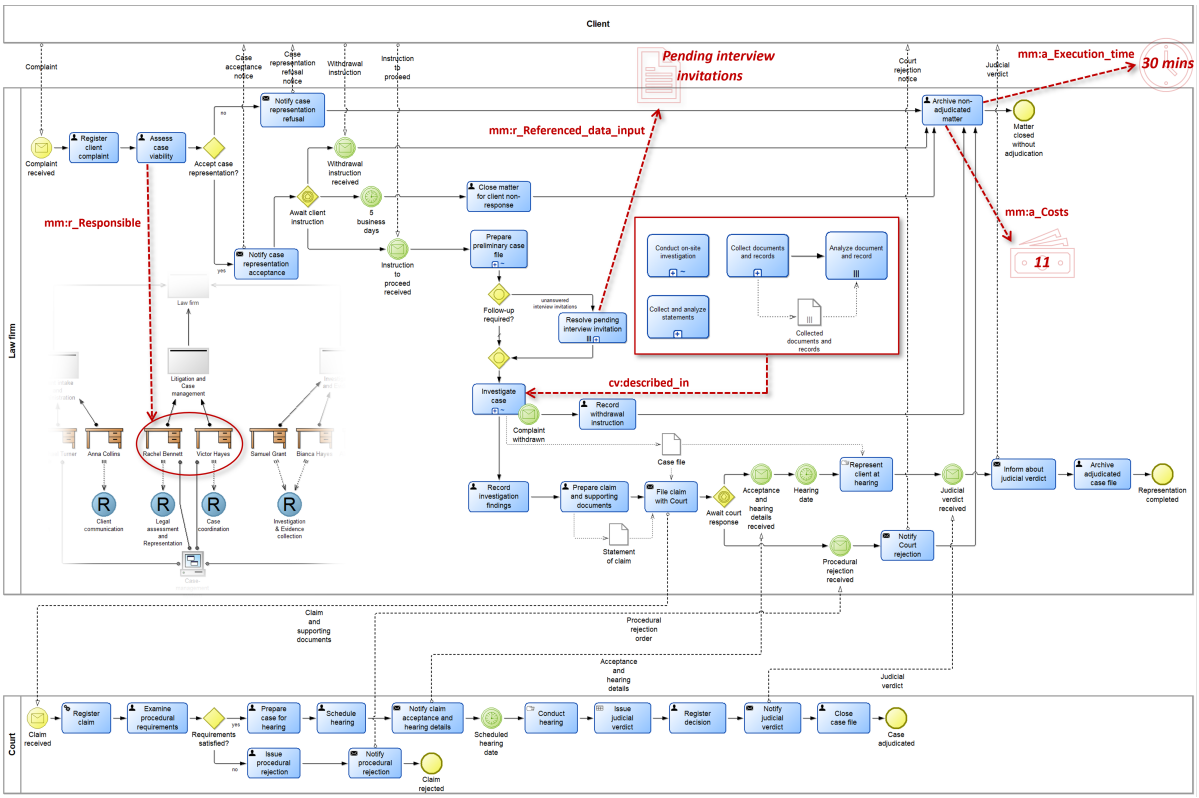


Figure 1: Main BPMN collaboration diagram in Bee-Up. Red annotations mark responsibility (left), containment (center), referenced data input (top center) and cost and duration (right). Other details are omitted for readability.

5. Experimental architecture and configuration

In order to query `bpmn_judicial` using natural language, Codex supplied gpt-5.6-Sol with the question, custom instructions and GraphDB's tool definitions. As the user interface and MCP client, it forwarded the LLM's tool requests to GraphDB, which retrieved evidence from the repository, its indexes or a dependent retrieval service. This cycle could continue until the LLM formulated the answer displayed by Codex. As Fig. 2 shows, Codex and GraphDB ran on one Windows workstation, with GraphDB exposing MCP tools through `Streamable HTTP` at `http://localhost:7200/mcp`. The ChatGPT Retrieval Plugin[10] API service and Weaviate[11] vector database ran locally in Docker containers[12], with Weaviate storing vector representations of indexed text.

[10] https://github.com/openai/chatgpt-retrieval-plugin
[11] https://weaviate.io
[12] https://www.docker.com/resources/what-container/

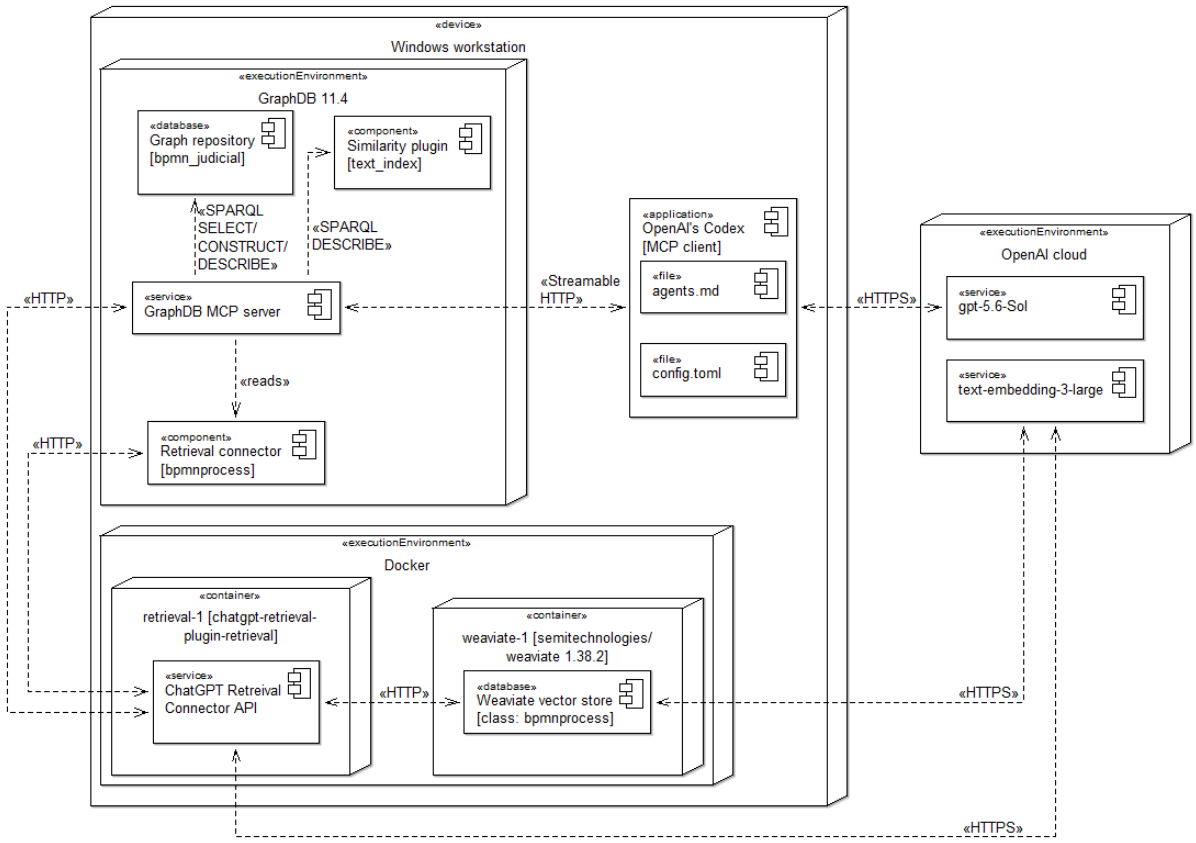


Figure 2: Simplified deployment architecture of the MCP-mediated Codex–GraphDB querying environment.

Each question was submitted in a fresh Codex chat to prevent retrieved facts or tool-use corrections (reasoning trail[13]) from influencing subsequent answers. Gpt-5.6-Sol[14] was used with *high*[15] reasoning effort as a controlled compromise between reasoning capacity and token use; *xhigh* and *max* target especially demanding reasoning, while *ultra* additionally supports automatic delegation, which lays outside our current single-agent design. We used a read-only sandbox and *Pragmatic*[16] communication style, with cross-chat memory, web search and multi-agent operation disabled. Custom instructions defined the repository, as well as the search facilities and required a JSON answer schema[17], leaving tool selection to the agent. The GraphDB MCP server exposed repository listing, ontology-schema extraction, SPARQL querying, full-text search, IRI discovery, retrieval search, similarity search and similarity-option discovery tools. Two of these tools are dependent on certain search structures, specifically:

- a **retrieval connector**[18] instance bridges RDF data and an external semantic retrieval system. Our instance, `bpmnprocess`, converted specified BPMN and Working Environment model content into textual facts and sent them to the local ChatGPT Retrieval Plugin API. Using `text-embedding-3-large`[19], the plugin embedded the indexed text and incoming queries, storing and searching the resulting vectors in Weaviate. For instance, following the `mm:r_Responsible` relation to the label of the responsible actor, produced text such as "-Register client complaint is carried out by Emily Carter." At query time, the `retrieval_search` tool

[13] https://learn.chatgpt.com/guides/best-practices
[14] https://learn.chatgpt.com/docs/models?surface=app
[15] https://learn.chatgpt.com/docs/agent-configuration/subagents?surface=app
[16] https://learn.chatgpt.com/docs/personalize
[17] https://developers.openai.com/api/docs/guides/structured-outputs#supported-schemas
[18] https://graphdb.ontotext.com/documentation/11.4/retrieval-graphdb-connector.html#retrieval-graphdb-connector-overview-of-connector-predicates
[19] https://developers.openai.com/api/docs/models/text-embedding-3-large

invoked this connector-plugin-Weaviate pipeline, returning textual evidence as the retrieval stage of a retrieval-augmented generation (RAG) process.

```
# prefix declarations omitted for compactness
{ "propertyChain": [ "mm:r_Responsible", "rdfs:label" ],
  "fieldTextPrefix": "is carried out by" }
```

- a **text similarity index**[20] provides an internal route to elements with similar descriptions. For our index, `text_index`, GraphDB's similarity plugin represented each process element as a *document* – here, meaning an indexed textual representation, not a separate file – via a data query pairing the the element's IRI (`?documentID`) with a SPARQL-generated description (`?documentText`) assembled from its name, types, containing model, cost, execution time and direct predecessor and successor labels. Lucene's `EnglishAnalyzer` tokenized the documents, while the `-termweight idf` parameter reduced the weight of recurring terms such as `Name`, `Element type`, `Cost`, `Incoming` etc. At query time, `similarity_search` vectorized the query, compared it with the indexed documents and ranked element IRIs by similarity; the SPARQL search query exposed these results as `?documentID` and `?score`.

```
# data query excerpt; prefixes omitted
SELECT ?documentID ?documentText WHERE {
  ?documentID a ?elementType; rdfs:label ?elementName .
  VALUES (?elementType ?elementTypeName) {
    (mm:o_Task_BPMN "task")
    (mm:o_Sub-Process_BPMN "subprocess") ... }
  ...
  BIND(CONCAT("Name: ", STR(?elementName), ". ",
    "Element type: ", STR(?elementTypeName), ".",
    ...) AS ?documentText) }

# search query
SELECT ?documentID ?score WHERE {
  ?search a ?index; ?searchType ?query;
    :searchParameters ?parameters; ?resultType ?result .
  ?result :value ?documentID; :score ?score . }
```

6. Experiments

This section evaluates structured and binary answers across the catalog's six categories, using a Python evaluator that automatically scores them against a manually constructed gold standard. Each subsection briefly introduces a category and examines the generated answers for a representative question across three runs. The complete query set, gold answers, raw conversations exported from Codex, reports and evaluator are available in the accompanying GitHub[21] repository.

6.1. Category 1: process element identification and containment

The 14 questions asked what processes and subprocesses we have, which elements they contain and how they start and end. With three runs per question, 23 of 27 structured answers recovered all facts required by their selected gold standards; 14 of 15 binary answers were correct. For the representative question in Table 2, all runs identified the seven requested (sub)processes, but combined what triggers them with how their internal works begin. The primary gold used the arrival of the *Client*'s *Complaint* message to represent the process' initiation[22]. All runs, instead, returned the *Complaint received* and *Claim received* start events. The *Investigate case*, *Prepare preliminary case file* and *Resolve pending*

[20] https://graphdb.ontotext.com/documentation/11.4/semantic-similarity-searches.html#text-based-similarity-searches
[21] https://github.com/DamarisDd/MCP-Codex-GraphDB-experiments.git
[22] https://www.omg.org/spec/BPMN/2.0.2/PDF

interview invitation subprocesses are reached through their respective incoming sequence flows, which the primary gold used to represent their initiation. The sequence flow/*subsequent* identifiers encode their source and target. Since the target subprocess was already identified in the answer, an alternative gold accepted the subprocesses' preceding elements, giving credit for the same modeled connection without requiring the flow identifier. Within the ad-hoc *Investigate case*, three subprocesses have no incoming sequence flows and are initially eligible for selection when their parent becomes active. The RDF specifies neither who selects them, nor on what basis. Nevertheless, the answers described this selection and additional internal beginnings. However, run 3 incorrectly listed *Analyze document and record* as a possible first activity, overlooking that *Collect documents and records* must finish first – a dependency not removed within the ad-hoc setting.

Table 2
Process and subprocess initiation across three runs

Question	How is each process or subprocess initiated?
Primary gold	*Main judicial process*: *Complaint* message flow from *Client* to *Complaint received*. *Prepare preliminary case file*: incoming sequence flow from *Instruction to proceed received*. *Resolve pending interview invitation*: incoming sequence flow from *Follow-up required?*, labeled (conditioned) *unanswered interview invitations*. *Investigate case*: incoming sequence flow from *Non-exclusive Gateway (BPMN)-59830*. *Collect documents and records*, *Conduct on-site investigation* and *Collect and analyze statements*: empty `related_elements` lists.
All runs	*Main judicial process*: *Complaint received* and *Claim received*. *Investigate case* and *Prepare preliminary case file*: ad-hoc, with incoming flows from *Non-exclusive Gateway (BPMN)-59830* and *Instruction to proceed received*, respectively. *Resolve pending interview invitation*: incoming flow from *Follow-up required?*; internal start event: *Initial interview request unanswered*. Within *Investigate case*, the answers described the selection of *Collect documents and records*, with the internal start event *Start document collection*; *Conduct on-site investigation*, itself ad-hoc; and *Collect and analyze statements*, with the internal start event *Scheduled date*.
Run 1	Called the message start events alternatives; no start event in the three ad-hoc subprocesses; described the gateway preceding *Investigate case* as inclusive; listed no possible first tasks.
Run 2	Described ad-hoc initiation as non-sequential; specified *unanswered interview invitations* as the incoming condition for *Resolve pending interview invitation*; listed eight possible first activities: - for *Investigate case*: *Collect documents and records*; *Conduct on-site investigation*; *Collect and analyze statements* - for *Prepare preliminary case file*: *Assign investigation team*; *Prepare list of potential witnesses*; *Identify unanswered interview invitations* - for *Conduct on-site investigation*: *Document observations*; *Collect physical evidence*.
Run 3	Stated that any task in the ad-hoc subprocesses could start first; listed run 2's activities plus *Analyze document and record* in *Investigate case* and *Send interview request* in *Prepare preliminary case file*.

Run	Retrieval search	SPARQL	Precision	Recall	F_1
1	1	15	0.27	0.75	0.40
2	1	12	0.16	0.75	0.26
3	1	12	0.14	0.75	0.24

6.2. Category 2: order of actions and possible routes

The nine questions examined what happens before or after an activity, how decisions branch, which activities can run in parallel and which routes connect two points in the process. Six required structured answers and three, yes/no answers. Thirteen of the 18 structured answers were exact and all nine binary answers were correct. The five non-exact answers concerned two questions about the steps

between specified elements. For the question in Table 3, a process *step* was interpreted as an activity represented by a task or subprocess. Gateways and events were excluded from the gold, as were the two boundary elements named in the question: *Register claim* and *Case adjudicated*, resulting in a gold of 11 activities across the Court and law firm's pools. Different orders across pools were accepted, provided that sequence flow order and message flow dependencies were preserved. Yet, all three runs returned the same Court sequence, adding a gateway and a timer event as steps, while omitting the law firm tasks. Run 3's final SPARQL query response even included the hearing and verdict message flow identifiers with their source and target elements. Tool use varied: runs 1-3 made eight, seven and seven calls, respectively, combining retrieval search with SPARQL; run 1 also used IRI discovery.

Table 3
Activities between two process steps across three runs

Question	What are all the steps that occur between registering a claim and adjudicating the case?					
Gold	*Examine procedural requirements*; *Prepare case for hearing*; *Schedule hearing*; *Notify claim acceptance and hearing details*; *Conduct hearing* and *Represent client at hearing*, in either order; *Issue judicial verdict*; *Register decision*; *Notify judicial verdict*; *Close case file* and *Inform about judicial verdict*, in either order.					
Runs	Returned answer					
1–3	All Court activities in the expected order. Law firm activities omitted: *Represent client at hearing* and *Inform about judicial verdict*. Two extras: *Requirements satisfied?* and *Scheduled hearing date*.					
Run	Retrieval search	SPARQL	IRI discovery	Precision	Recall	F_1
1	1	5	2	0.82	0.82	0.82
2	3	4	0	0.82	0.82	0.82
3	2	5	0	0.82	0.82	0.82

6.3. Category 3: resource allocation and modeled performance

The cross-pool scope issue from the previous category reappeared in the question asking for the longest complete process route and its duration (Table 4). This category's 23 questions asked who does what, how work is distributed and what expected time and cost it requires. Of the 69 structured answers, 60 were exact. The gold accepted a compact representation with 22 activities (19 tasks and three subprocesses) or 40 tasks after expanding the subprocesses. For either representation, the accepted duration was 42 h 40 min as the sum of task durations, or 39 h 10 min after accounting for the following assumed overlaps: (1) *Conduct hearing* and *Represent client at hearing* occur simultaneously – both take three hours and their preceding timer labels suggest the same appointment; and (2) the law firm receives the verdict as soon as the Court sends it. The Court's *Close case file* (30 minutes) and the law firm's *Inform about judicial verdict* (40 minutes) could start together, taking 40 minutes altogether. These interpretations draw on the model's labels, durations and dependencies; shared start times are not explicitly encoded. All activities still had to be listed, even when their durations overlapped, with sequence and message flow dependencies preserved; events and gateways were optional. Runs 1 and 2 returned compact routes of 34 hours (which is equal to the law firm's execution time subtotal) omitting the Court's activities, even if these and their durations had appeared in the tools' results. Once again, all runs combined retrieval search with SPARQL; run 3 additionally used `ontology_schema`, expanded the subprocesses and summed 37 task durations, obtaining the accepted 39 h 10 min. Interestingly, run 3 omitted *Represent client at hearing* (3 hours), *Close case file* (30 minutes) and *Interview involved parties* (no recorded duration). Removing the first two reduced the sum by exactly the time discounted for the previously mentioned possible overlaps; the third task did not change it. A gateway and three events were also absent, but their omission was not penalized under activity-only scoring.

Table 4
The longest process route

Question	Of the available ways to complete the process, which ones, from start to finish, have the longest execution time and what are each of their durations?						
Gold	Compact route: 22 activities (19 tasks and three subprocesses). Expanded route: 40 tasks. Accepted duration for either representation: 42 h 40 min or 39 h 10 min.						
Run	Returned answer						
1–2	Compact route: 12/22 activities; 34 h.						
3	Expanded route: 37/40 tasks; 39 h 10 min.						
Run	Retrieval search	SPARQL	Repository listing	Ontology schema	Precision	Recall	F_1
1	1	6	1	0	0.92	0.52	0.67
2	1	8	0	0	0.92	0.52	0.67
3	2	7	0	1	1.00	0.93	0.96

6.4. Category 4: document identification and use

The eight questions concerned which activities produce or use documents, where they are needed and whether any remain unused. 20 the 24 structured answers were exact. Answering the question in Table 5 required finding documents along routes between *Prepare list of potential witnesses* and *Matter closed without adjudication*. The matter can close after the Court rejects a filed claim (along a route involving five documents) or following the complaint withdrawal during investigation. For consistency, we instructed the agent to assume that withdrawal interrupts investigation after exactly one internal task completes, before another internal or external element is selected. Under this assumption, the primary gold contained *Potential witnesses*, *Pending interview invitations* and *Collected statements*: the last could already exist if *Document statements* completed first before interruption. Runs 1-3 made 13, 18 and 23 calls, respectively, combining the already dominant tools: retrieval search and SPARQL; runs 2 and 3 also used full-text search and IRI discovery to locate candidate endpoints. All runs retained *Collected documents and records*, but under the said interpretation, its producing subprocess requires several internal tasks to complete, whereas the assumed interruption occurs after only one.

Table 5
Documents used or produced within a process interval

Question	Which documents are used or produced between preparing the witnesses list and closing the matter?							
Gold	Primary (three documents): *Potential witnesses*; *Pending interview invitations*; *Collected statements*. Accepted alternative (five documents): the same three, plus *Case file* and *Statement of claim*.							
Run	Returned answer							
1	All three documents in the primary gold, plus *Collected documents and records*.							
2–3	All five documents in the alternative gold, plus *Collected documents and records*.							
Run	Retrieval search	SPARQL	Full-text search	IRI discovery	IRI auto-complete	Precision	Recall	F_1
1	2	11	0	0	0	0.75	1.00	0.86
2	2	11	2	1	2	0.83	1.00	0.91
3	2	17	2	2	0	0.83	1.00	0.91

6.5. Category 5: communication between participants

The five questions examined who communicates with whom, what they exchange and when. All 15 structured answers were exact under the accepted scoring rules, although not always presenting information the same way, as illustrated in the following excerpts. Run 1 placed the interaction points among each message flow's elements, while run 2 returned them as the main results. However, this matter of placement did not affect their scores. The two runs combined retrieval search with SPARQL, making four and five calls, respectively.

Question. At what points in the process does the law firm interact with the client?

Run 1: message flow as root

```
"label": "Complaint",
"element_type": "message_flow_bpmn",
"related_elements": [
  {"relation": "from", "label": "Client"},
  {"relation": "to",
    "label": "Complaint received"}]
```

Run 2: interaction point as root

```
"label": "Complaint received",
"element_type": "start_event_bpmn",
"related_elements": [
  {"relation": "receives_message_from",
    "label": "Client"}]
```

6.6. Category 6: potential process weaknesses and improvement opportunities

Weakness-oriented questions appeared in earlier categories, covering outcome gaps, workload concentration, assignment anomalies, performance extremes, unused documents and message dependencies. The final four questions left for examination focus on where work repeats, whether it can return to earlier steps, which outcomes depend on a participant's communication and where a lack of response from a participant has no modeled fallback. Ten of the 12 structured answers were exact. The question in Table 6 revisits an earlier scope issue: the answer must include outcomes in one participant's process that depend on events in another's. The events following *Await client instruction* determine whether a claim reaches the Court, triggering its process and possible rejection or adjudication outcomes. Withdrawal or expiry of the five business day timer leads to closure without adjudication. All runs combined retrieval search with SPARQL and retrieved all four end events. Yet, run 3 retained only the two law firm outcomes after its final query checked the events' properties, including their pool assignments.

Table 6
Outcome dependencies across participant boundaries

Question	Which process outcomes from the main process depend on awaiting for the client's instruction?				
Gold	*Case adjudicated; Claim rejected; Matter closed without adjudication; Representation completed*				
Run	Returned answer				
1–2	All four outcomes.				
3	*Matter closed without adjudication*; *Representation completed.*				
Run	Retrieval search	SPARQL	Precision	Recall	F_1
1	2	8	1.00	1.00	1.00
2	2	5	1.00	1.00	1.00
3	1	4	1.00	0.50	0.67

6.7. Main findings

The agent answered most questions successfully, consistently handling process element inventories, questions about who performs particular tasks and comparisons of modeled task durations and costs. Its instructions governed repository use and answer structure, leaving tool selection and call sequences to the agent. Retrieval search appeared in most runs; SPARQL accounted for most calls. Difficulties

often concerned nested subprocesses or collaborating pools. Question wording may help explain some variation between runs (Figure 3). Explicit scope delimitation, such as *in each process or subprocess*, produced consistent answers; while less specific formulations, such as *part of* or *deal with* allowed different readings of nested membership. Similar ambiguities concerned whether *steps* included events and gateways, which alternatives fell *between* two points and whether *initiated* meant subprocess activation or the beginning of its internal work. Although instructed to seek clarification when uncertain, Codex always answered without asking. The transcripts reveal two sources of information loss: (1) some queries excluded nested tasks; (2) even if relevant facts appeared in the tool results, the final answer omitted them. For Codex running on gpt-5.6-Sol, following process routes was not uniformly difficult and additional tool calls did not necessarily produce more complete answers. Clear scope rules or documented gold alternatives, where applicable, would ease interpretation. Comparing what the tools retrieved with what the final answer included would help in locating information loss. Whether more detailed guidance improves answer quality, remains to be tested. Comparisons between categories remain descriptive, since question counts and difficulty were not controlled.

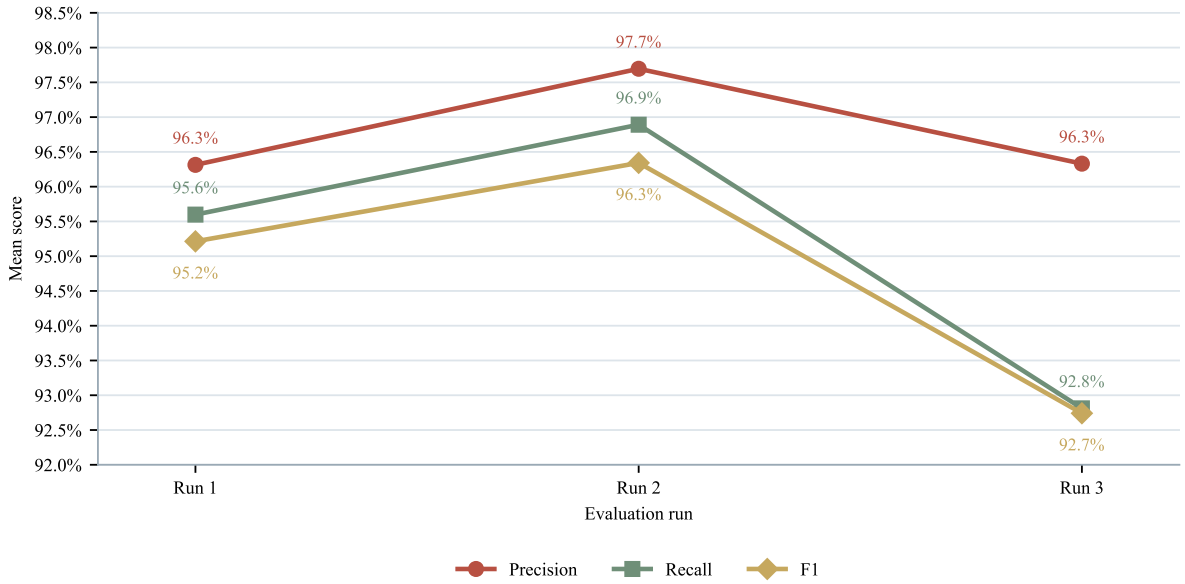


Figure 3: Macro-averaged precision, recall and F_1 for structured answers by run.

7. Conclusions

In this integration of established technologies used for business process querying, GraphDB stored Bee-Up's RDF export, while Codex, using gpt-5.6-Sol, selected GraphDB's MCP tools and constructed answers. Most questions across six categories of managerial concerns were answered successfully. Remaining errors highlighted the importance of selecting the right scope, while respecting modeled dependencies and preserving relevant facts. Additional tool calls did not ensure answer completeness.

These experiments remain bounded by one synthetic repository, one agent and three repetitions per question, limiting conclusions about larger repositories or other domains. Differences between runs are consistent with stochastic variation in tool use and answer generation; they do not seem to demonstrate learning or convergence. Perhaps more questions and runs with controlled difficulty could help assess this variability. Our next priority is to develop a reusable benchmark for natural language querying of BPMN models in RDF. We plan to extend the repository around recurring BPMN patterns and investigate public collections available in the literature (e.g., Camunda's BPMN for research[23] or

[23] https://github.com/camunda/bpmn-for-research

the BPM Academic Initiative [33]). Given that such collections tend to be XML standard-compliant and that only a few experimental RDF representations are currently available outside Bee-Up, their reuse would require extensive redesigns for RDF serialization [18]. Future re-deployments of the experimental protocol could diversity the tool ecosystem, while maintaining the focus on BPMN and RDF.

Declaration on Generative AI

During the preparation of this work, the author used gpt-5.6-Sol to improve the readability and English grammar. After using this service, the author reviewed and edited the content as needed and takes full responsibility for the publication's content.

References

[1] M. Dumas, M. La Rosa, J. Mendling, H. A. Reijers, Introduction to Business Process Management, in: Fundamentals of Business Process Management, 2 ed., Springer, 2018, pp. 1–33. doi:10.1007/978-3-662-56509-4_1.

[2] R. Dijkman, M. Dumas, B. van Dongen, R. Käärik, J. Mendling, Similarity of Business Process Models: Metrics and Evaluation, Inf. Syst. 36 (2011) 498–516. doi:10.1016/j.is.2010.09.006.

[3] C. Beeri, A. Eyal, S. Kamenkovich, T. Milo, Querying Business Processes with BP-QL, Inf. Syst. 33 (2008) 477–507. doi:10.1016/j.is.2008.02.005.

[4] C. Di Francescomarino, P. Tonella, The BPMN Visual Query Language and Process Querying Framework, in: Process Querying Methods, Springer, 2022, pp. 181–218. doi:10.1007/978-3-030-92875-9_7.

[5] L. Barbieri, K. Stroeh, E. R. M. Madeira, W. M. P. van der Aalst, An LLM-Based Q&A Natural Language Interface to Process Mining, in: ICPM Workshops, volume 533 of *LNBIP*, Springer, 2025, pp. 5–17. doi:10.1007/978-3-031-82225-4_1.

[6] T. Schick, J. Dwivedi-Yu, R. Dessi, et al., Toolformer: Language Models Can Teach Themselves to Use Tools, in: Adv. Neural Inf. Process. Syst. 36, Neural Information Processing Systems Foundation, 2023, pp. 68539–68551. doi:10.52202/075280-2997.

[7] J.-K. Dong, I.-W. Huang, C.-T. Wu, Y.-T. Tsai, ETOM: A Five-Level Benchmark for Evaluating Tool Orchestration within the MCP Ecosystem, in: Findings of ACL: EACL 2026, Association for Computational Linguistics, 2026, pp. 1453–1488. doi:10.18653/v1/2026.findings-eacl.75.

[8] A. R. Kinjo, Y. Yamamoto, S. Bustamante-Larriet, J.-E. Labra-Gayo, T. Fujisawa, TogoMCP: Natural Language Querying of Life-Science Knowledge Graphs via Schema-Guided LLMs and the Model Context Protocol, Database 2026 (2026). doi:10.1093/database/baag042.

[9] E. G. L. de Murillas, H. A. Reijers, W. M. P. van der Aalst, Data-Aware Process Oriented Query Language, in: Process Querying Methods, Springer, 2022, pp. 49–83. doi:10.1007/978-3-030-92875-9_3.

[10] M. L. Bernardi, A. Casciani, M. Cimitile, A. Marrella, Conversing with Business Process-Aware Large Language Models: The BPLLM Framework, J. Intell. Inf. Syst. 62 (2024) 1607–1629. doi:10.1007/s10844-024-00898-1.

[11] L. F. Lins, N. Nascimento, P. Alencar, T. Oliveira, D. Cowan, Comparing Generative Chatbots Based on Process Requirements: A Case Study, in: IEEE BigData, IEEE, 2023, pp. 4664–4673. doi:10.1109/BigData59044.2023.10386251.

[12] A. H. M. ter Hofstede, C. Ouyang, M. La Rosa, L. Song, J. Wang, A. Polyvyanyy, APQL: A Process-Model Query Language, in: AP-BPM, volume 159 of *LNBIP*, Springer, 2013, pp. 23–38. doi:10.1007/978-3-319-02922-1_2.

[13] A. Awad, S. Sakr, On Efficient Processing of BPMN-Q Queries, Comput. Ind. 63 (2012) 867–881. doi:10.1016/j.compind.2012.06.002.

[14] A. Polyvyanyy, Process Query Language, in: Process Querying Methods, Springer, 2022, pp. 313–341. doi:10.1007/978-3-030-92875-9_11.

[15] A. Awad, A. Polyvyanyy, M. Weske, Semantic Querying of Business Process Models, in: IEEE EDOC, IEEE, 2008, pp. 85–94. doi:10.1109/EDOC.2008.11.

[16] Ș. Uifălean, A.-M. Ghiran, R. A. Buchmann, Employing Graph Databases for Business Process Management and Representation, in: Adv. Inf. Syst. Dev., volume 63 of *LNISO*, Springer, 2023, pp. 73–92. doi:10.1007/978-3-031-32418-5_5.

[17] R. A. Buchmann, D. Karagiannis, Pattern-Based Transformation of Diagrammatic Conceptual Models for Semantic Enrichment in the Web of Data, Procedia Comput. Sci. 60 (2015) 150–159. doi:10.1016/j.procs.2015.08.114.

[18] S. Bachhofner, E. Kiesling, K. Revoredo, P. Waibel, A. Polleres, Automated Process Knowledge Graph Construction from BPMN Models, in: DEXA, volume 13426 of *LNCS*, Springer, 2022, pp. 32–47. doi:10.1007/978-3-031-12423-5_3.

[19] R. A. Buchmann, M. Ussenbayeva, W. Utz, D. Karagiannis, Leveraging RDF Graphs, Similarity Metrics and Network Analysis for Business Process Management, in: AAAI-MAKE Spring Symp., volume 3433 of *CEUR Workshop Proc.*, CEUR-WS.org, 2023. URL: https://ceur-ws.org/Vol-3433/paper14.pdf.

[20] M. Vidgof, S. Bachhofner, J. Mendling, Large Language Models for Business Process Management: Opportunities and Challenges, in: BPM Forum, volume 490 of *LNBIP*, Springer, 2023, pp. 107–123. doi:10.1007/978-3-031-41623-1_7.

[21] H. Kourani, A. Berti, J. Hennrich, et al., Leveraging Large Language Models for Enhanced Process Model Comprehension, Decis. Support Syst. 200 (2026). doi:10.1016/j.dss.2025.114563.

[22] T. C. Nemțoc, R. A. Buchmann, G. C. Silaghi, Comparative Experiments on Natural Language Querying of Knowledge Graphs Using a Graph RAG Approach, in: ER, volume 16189 of *LNCS*, Springer, 2025, pp. 302–319. doi:10.1007/978-3-032-08623-5_16.

[23] A. Elgammal, O. Turetken, CRL and the Design-Time Compliance Management Framework, in: Process Querying Methods, Springer, 2022, pp. 285–311. doi:10.1007/978-3-030-92875-9_10.

[24] K. Kammerer, J. Kolb, M. Reichert, PQL: A Descriptive Language for Querying, Abstracting and Changing Process Models, in: BPMDS/EMMSAD, volume 214 of *LNBIP*, Springer, 2015, pp. 135–150. doi:10.1007/978-3-319-19237-6_9.

[25] M. Fellmann, O. Thomas, B. Busch, A Query-Driven Approach for Checking the Semantic Correctness of Ontology-Based Process Representations, in: BIS, volume 87 of *LNBIP*, Springer, 2011, pp. 62–73. doi:10.1007/978-3-642-21863-7_6.

[26] V. Acretoaie, H. Störrle, Efficient Model Querying with VMQL, in: CMSEBA, volume 1340 of *CEUR Workshop Proc.*, CEUR-WS.org, 2014, pp. 7–16. URL: https://ceur-ws.org/Vol-1340/paper2.pdf.

[27] M. Missikoff, M. Proietti, F. Smith, Querying Semantically Enriched Business Processes, in: DEXA, volume 6861 of *LNCS*, Springer, 2011, pp. 294–302. doi:10.1007/978-3-642-23091-2_25.

[28] L. Pufahl, F. Stiehle, S. Ihde, M. Weske, I. Weber, Resource Allocation in Business Process Executions – A Systematic Literature Study, Inf. Syst. 132 (2025). doi:10.1016/j.is.2025.102541.

[29] N. Russell, W. M. P. van der Aalst, A. H. M. ter Hofstede, D. Edmond, Workflow Resource Patterns: Identification, Representation and Tool Support, in: CAiSE, volume 3520 of *LNCS*, Springer, 2005, pp. 216–232. doi:10.1007/11431855_16.

[30] P. Delfmann, D. M. Riehle, S. Höhenberger, C. Corea, C. Drodt, The Diagramed Model Query Language 2.0: Design, Implementation, and Evaluation, in: Process Querying Methods, Springer, 2022, pp. 115–148. doi:10.1007/978-3-030-92875-9_5.

[31] F. Corradini, F. Fornari, A. Polini, B. Re, F. Tiezzi, A. Vandin, A Formal Approach for the Analysis of BPMN Collaboration Models, J. Syst. Softw. 180 (2021) 111007. doi:10.1016/j.jss.2021.111007.

[32] E.-M. Kalogeraki, T. Panayiotopoulos, D. Apostolou, Semantic Queries in BPMN 2.0: A Contemporary Method for Information Retrieval, in: IISA, IEEE, 2015, pp. 1–6. doi:10.1109/iisa.2015.7388061.

[33] M. Weske, G. Decker, M. Dumas, M. La Rosa, J. Mendling, H. A. Reijers, Model Collection of the Business Process Management Academic Initiative, 2020. URL: https://fundamentals-of-bpm.org/process-model-collections/.