Explainability and Traceability of AI Systems through Provenance Tagging

Devin De Silva[1], Henrique Santos[1], John S. Erickson[1] and Deborah L. McGuinness[1]

[1]*Rensselaer Polytechnic Institute, Troy, NY 12180, USA*

Abstract

As artificial intelligence (AI) systems evolve into complex, multi-stage workflows orchestrating large language models (LLMs) and dynamic code execution, their inherent non-determinism and opacity pose significant barriers to trust, transparency, and accountability. Current monitoring tools are often framework-dependent and primarily focus on tracing individual agent actions rather than providing a holistic view of interactions across multiple components. To address this challenge, we present a novel hybrid framework for generating traceable, on-demand explanations for AI systems. Our approach leverages the ProvONE and Explanation Ontologies to capture execution traces as a knowledge graph, which is then used to generate natural-language explanations for user questions through a KG-based retrieval mechanism. Departing from traditional open-ended retrieval, we leverage the ontology to systematically generate a set of SPARQL queries that act as retrieval functions. Given a user question, the system iteratively selects and executes appropriate queries over the KG, using retrieved information to refine subsequent steps until a satisfactory answer is obtained. The system aggregates the final results into a structured chain of evidence to produce context-aware, verifiable explanations. We evaluate our approach through its implementation in ChatBS-NexGen, a dynamic LLM unit-testing system, and Biomni, a general-purpose biomedical AI agent. Our results on ChatBS-NexGen and Biomni show that the proposed approach outperforms all baselines on explanation-quality metrics and task-specific accuracy, achieves the highest final-step entity recall, and exceeds an 80% win rate in most pairwise explanation comparisons.

Keywords

Explainable AI (xAI), Provenance, AI Workflows, Knowledge graph-based xAI

1. Introduction

Modern artificial intelligence (AI) systems have moved beyond simple input-output models to become sophisticated orchestration engines, shifting from providing static responses to performing dynamic program synthesis and agentic workflows [1]. Such workflows often orchestrate multiple invocations of Large Language Models (LLMs), which are used not only for text generation but also for synthesizing executable code, defining intermediate functions on-the-fly to fulfill user intents, and interacting with external environments and tools [1, 2, 3]. In this work, an "execution trace" refers to the runtime record of such a workflow: the sequence of LLM calls, generated code or functions, tool invocations, input and output artifacts, intermediate data, and dependencies between workflow steps. While such architectures significantly elevate system capabilities, they inherit the non-determinism and opacity of their underlying models. This "black box" characteristic presents a significant barrier to adoption in critical domains [4], as it fundamentally undermines trustworthiness, accountability, traceability, and the ability to debug or verify system behavior. As AI-driven systems become more autonomous, the need for transparent, structured, and verifiable explanations grounded in such traces becomes a critical requirement for reliable system deployment.

Existing observability and tracing frameworks (e.g., OpenTelemetry [5]) primarily operate at the level of low-level service spans and function calls, while LLM-specific tracing tools (e.g., AgentOps [6]) are often framework-dependent. As a result, current approaches lack native support for capturing

XAI-KG@ISWC2026 - 2nd International Workshop on eXplainable AI and Knowledge Graphs, co-located with ISWC 2026, October 25-29, 2026, Bari, Italy

desild@rpi.edu (D. D. Silva); oliveh@rpi.edu (H. Santos); erickj4@rpi.edu (J. S. Erickson); dlm@cs.rpi.edu (D. L. McGuinness)

0000-0001-6678-6227 (D. D. Silva); 0000-0002-2110-6416 (H. Santos); 0000-0003-3078-4566 (J. S. Erickson); 0000-0001-7037-4567 (D. L. McGuinness)

© 2026 Copyright © 2026 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 2026-11-07

CEUR-WS.org/Vol-4318/paper7.pdf

semantically meaningful, cross-component provenance across multi-component AI systems, including agentic workflows. This operational opacity creates a dual challenge: how can execution traces be efficiently captured, and how can these traces be synthesized into coherent explanations?

On the information retrieval side, existing approaches such as Retrieval-Augmented Generation (RAG) [7, 8] and SPARQL generation strategies [9, 10] treat retrieval as an open-ended generation problem or an LLM-guided graph traversal problem. However, structured workflow traces require an expressive model that preserves the semantics and dependencies of execution. Recent work suggests that constraining the structured-query search space can improve reliability by reducing semantically invalid query paths and limiting the space of possible query constructions [11, 12]. This motivates a more constrained retrieval formulation in which valid operations are derived from the underlying schema rather than generated freely at inference time. Ignoring such structure may lead to inefficient retrieval, structurally or semantically invalid queries, increased hallucination risk, and reduced interpretability.

Building on these insights, we focus on data explanations and trace-based explanations [13], where explanations are grounded in the data, intermediate artifacts, and execution dependencies produced by the workflow. Accordingly, we seek to answer the following research question: "How can execution traces from multi-stage LLM-driven AI workflows be semantically represented and queried to generate faithful, traceable, auditable, and on-demand explanations of system behavior?" To address this question, we present a hybrid neuro-symbolic framework that leverages ontology-grounded knowledge graphs to provide traceable, on-demand explanations for complex AI workflows. Our approach transforms execution traces into a semantically structured resource and formulates explanation generation as a process of executing verifiable queries over this resource. To achieve this, we introduce a cohesive three-stage methodology: (1) a semantic annotation layer that captures workflow execution using ontologies such as ProvONE [14] and the Explanation Ontology (EO) [13], together with annotation libraries in Python and R to capture intermediate artifacts of AI systems; (2) a schema-driven generator that derives questions paired with executable SPARQL queries, forming a constrained function library; and (3) a neuro-symbolic explanation module that iteratively retrieves evidence using this function library.

By constraining retrieval to executable, ontology-aligned operations, the system ensures that generated explanations remain grounded in the underlying execution data. We evaluated our approach on ChatBS-NexGen [2] (an open-source platform for empirically validating LLM outputs against authoritative knowledge graphs) and Biomni [3] (a general-purpose biomedical AI agent), and results show that we outperform baselines across semantic quality and task-specific accuracy, with pairwise LLM-judged win rates exceeding 80% in most comparisons.

2. Related Work

Our work sits at the intersection of AI workflow representation and knowledge retrieval systems. Modeling AI workflows naturally favors knowledge graphs (KGs) and ontologies due to the inherently nested structure of workflow information and the interconnected use of data across workflow components. Furthermore, KG-based information retrieval provides an important paradigm for retrieving relevant information from these modeled AI workflows.

2.1. AI Workflow Modeling & Provenance

Provenance tracking is widely recognized as essential for ensuring transparency, accountability, and explainability in machine learning (ML) pipelines [15, 16]. A recent survey [15] argues that provenance is critical for trustworthy AI and notes that many current implementations remain ad-hoc. To address this, frameworks like MLflow2PROV [16] extract end-to-end provenance from industry-standard tools (e.g., Git, MLflow), validating the use of the W3C PROV ontology [17] as a robust backbone for serializing ML pipelines into Knowledge Graphs (KGs). Similarly, recent adaptations [18] have extended PROV-O to create the Safe Haven Provenance ontology, automating tracking within Trusted Research Environments.

Recent work increasingly connects provenance with Explainable AI (XAI). Data Explainability [19] argues that feature-attribution methods such as SHAP [20] and LIME [21] should be complemented by provenance information. Related approaches link provenance with model explanations or use it to quantify transparency and accountability [22], while the Data Journey Ontology (DJO) [23] abstracts Python notebook workflows into activity graphs. ProvAgent [24] further extends W3C PROV [17] with semantics for AI-agent interactions. In contrast, our work targets programmatic, compositional workflows containing both deterministic and non-deterministic components and explicitly represents how AI components are embedded within larger workflow executions.

2.2. Ontology-Guided KG-RAG & SPARQL generation

Recent advancements in KG question answering (KGQA) and KG-based RAG increasingly combine LLMs with structured retrieval. Systems such as BYOKG-RAG [7] use multi-strategy retrieval to ground answers in graph-derived entities, while SimGRAG [25] aligns user questions with graph patterns using Graph Semantic Distance. Self-supervised approaches such as BYOKG [8] further employ symbolic agents to explore KGs and synthesize query-program exemplars for zero-shot generalization. More recent structure-aware RAG methods, including HippoRAG [26], construct graph structures from text and use graph traversal mechanisms such as personalized PageRank to support multi-hop and associative retrieval. HyperGraphRAG [27] extends graph-based RAG by representing higher-order, n-ary relations as hyperedges and retrieving relevant entities and hyperedges through embedding-based similarity followed by neighborhood expansion. While effective, these approaches typically rely on induced graph or hypergraph structures, extracted relational facts, embeddings, and ranking heuristics, making retrieval dependent on the quality of knowledge extraction, representation construction, and semantic matching.

A closely related line of work focuses on translating natural language questions into executable SPARQL queries. LLM-based text-to-SPARQL methods enable zero-shot and few-shot query construction, with systems such as SPINACH [10] iteratively exploring Wikidata through specialized functions and GRASP [9] generalizing this strategy across RDF knowledge graphs. Other recent systems use retrieval-augmented prompting and schema-aware context injection to generate SPARQL over domain-specific or federated KGs [28, 29]. However, generated SPARQL can suffer from structural and semantic errors, including incorrect triple patterns and entity mismatches [30], and recent analyses suggest that some performance gains may reflect memorization rather than robust generalization [31].

Despite this progress, ontology-constrained retrieval remains under-explored. Competency questions provide a useful abstraction for expressing information that an ontology-backed KG should answer, and tools such as Q2Forge [32] and RevOnt [33] partially automate their generation. Related methods use ontologies for path pruning [34], while recent structured-query work shows that constraining the query space can improve reliability [11, 12], improving query-generation accuracy and query path validity. Motivated by these findings, we use the ontology schema to enumerate valid query paths and precompute question-SPARQL pairs as executable retrieval functions. This shifts inference-time retrieval from open-ended query generation to selection and composition over a validated query set.

3. Methodology

Our methodology transforms AI workflow executions into queryable, provenance-grounded explanations. It consists of a setup phase (Secs. 3.1–3.3), where workflow executions are annotated and the SQ Library is generated, and a question-answering phase (Sec. 3.4), where the library is used to retrieve evidence for user questions. Developer-provided annotations are captured through Python/R libraries and serialized into a workflow graph conforming to the backing ontology.

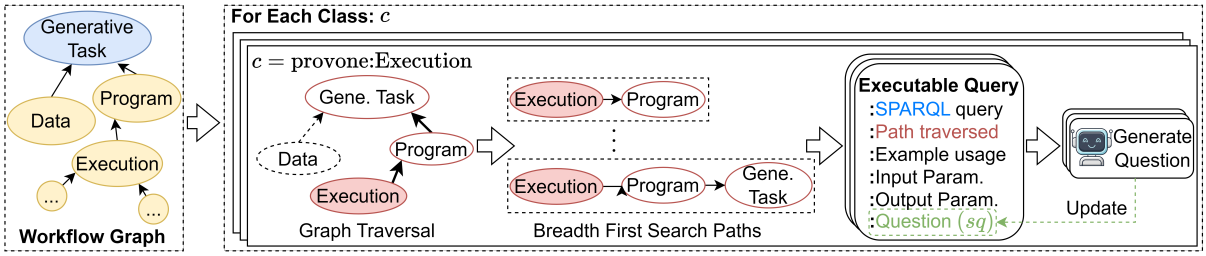


Figure 1: Automated generation of the Synthetic Question library. The process involves (1) traversing the source ontology schema to find paths, (2) formulating executable SPARQL queries and input parameters for each path, (3) generating synthetic natural language questions using an AI agent, and (4) modeling the resulting query-question pairs as a structured graph.

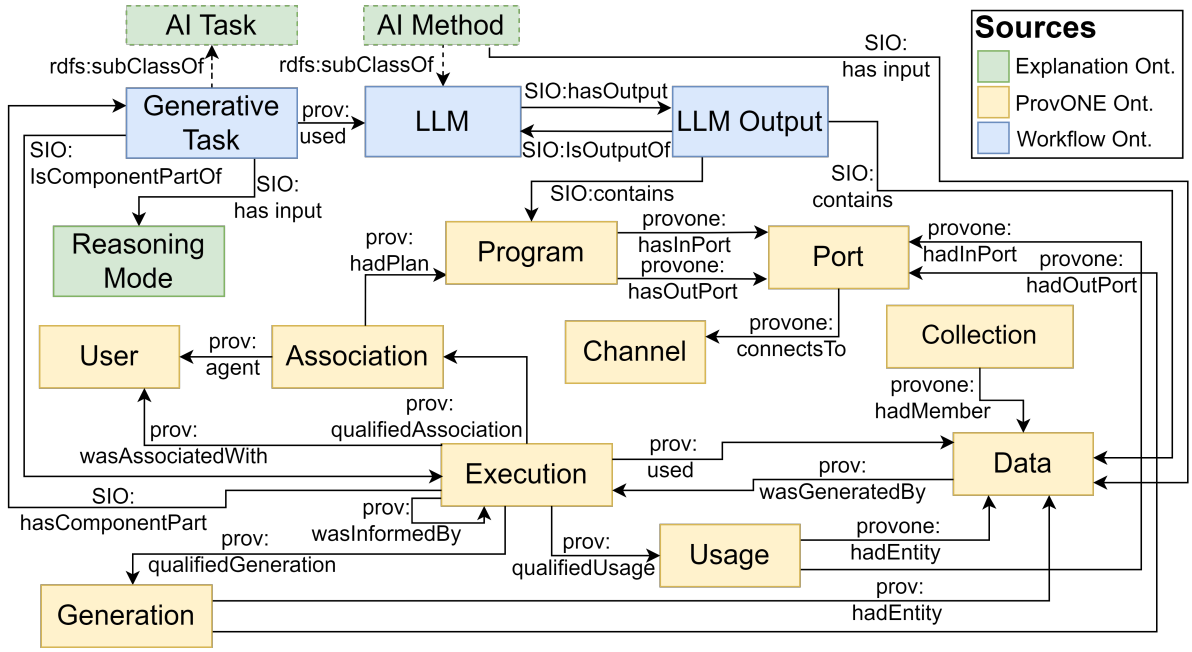


Figure 2: Backing ontology for the workflow graph

3.1. AI Workflow Representation and Workflow Graph Annotation

AI workflows are characterized by complex, nested, and interconnected processes, which makes modeling their execution a significant challenge. An AI component might produce an artifact or a function that is subsequently utilized in a different, separate workflow. To formally capture the elements of this execution, we reuse and align ProvONE [14] (provone:), the Explanation Ontology [13] (EO), the Semanticscience Integrated Ontology (SIO) [35], and W3C's Provenance Ontology (PROV-O) [17] by defining semantics to capture generative tasks and to capture how the AI is used within a system (denoted as Workflow Ontology) as depicted in Figure 2.

The ProvONE ontology provides the structural backbone for modeling the provenance and data flow of computational processes. We use its core classes to represent the "skeleton" and the runtime behavior of the workflow. For example:

- `provone:Program` represents the specification or definition of a particular workflow step.
- `provone:Execution` captures a specific runtime instance of that `Program`.
- `provone:Data`, `provone:Collection`, and `provone:Port` model the data artifacts a program consumes and produces during execution.
- `provone:User` and `prov:Association` link the agents (users or other systems) responsible for initiating or designing the workflow.

While ProvONE excels at providing the concepts for modeling the structure of the execution trace (i.e., "what ran and what data was used"), it does not inherently describe the purpose or nature of the AI tasks themselves coupled with their inputs and outputs. EO provides a set of classes that can be used to semantically enrich the ProvONE model, allowing us to describe the "what" and "why" of the AI-driven steps. As shown in Figure 2, we extend EO to create subclasses to represent generative tasks (`Generative Task`), LLM-based AI methods (`LLM`), and specific outputs (`LLM Output`, which can be `provone:Program` and `provone:Data`) that an LLM may generate.

- To model generated functions, we utilize `workflow:LLMOutput` $\xrightarrow{\texttt{sio:contains}}$ `provone:Program` connections.
- To represent the use of generative tasks within a function as a component, we utilize:
  - `provone:Execution` $\xrightarrow{\texttt{sio:hasComponentPart}}$ `workflow:Generative_Task`
  - `workflow:Generative_Task` $\xrightarrow{\texttt{sio:isComponentPartOf}}$ `provone:Execution`

This design captures structural provenance (using ProvONE and PROV-O) and task-level semantics (with EO and SIO), enabling execution traceability and explanation generation within a unified representation. The resulting schema defines the primitives and relations required to represent AI workflow executions, including programs, executions, ports, data artifacts, generated functions, and generative tasks. To operationalize this schema in real systems, we develop companion Python and R libraries that allow workflow code to be annotated and serialized into a workflow graph.

3.2. Python/R Library

The Python and R provenance libraries implement the annotation schema described above and support the construction of workflow graphs from AI system executions. The libraries provide high-level functions for declaring workflow programs, ports, channels, executions, inputs, outputs, and inter-step dependencies. Rather than requiring users to manually construct RDF triples, the API records each function or agent step as a `provone:Program` and each runtime invocation as a `provone:Execution`, linked to the artifacts it used or generated.

The libraries are designed to capture intermediate states that are often omitted from end-to-end workflow logs, including prompts, generated responses, extracted values, data-frame rows, lists, observations, and final outputs. These artifacts are represented as provenance data objects or collections and connected to the corresponding execution. For LLM-based workflows, the libraries also record model calls as generative tasks, including the model used, the inputs supplied, and the outputs produced. The captured annotations are then serialized as Turtle [36], producing the workflow graph used by the explanation system.

3.3. Synthetic Question Creation

Because the workflow graph conforms to a backing ontology (Fig. 2), we formulate retrieval as selection from a predefined set of executable SPARQL queries rather than generating queries directly at inference time using the ontology. Prior work suggests that constraining structured-query search spaces can improve reliability [11, 12]. We therefore traverse the ontology to identify viable semantic paths, defined as acyclic property sequences connecting a source and target class (e.g., `provone:Program` $\rightarrow$ `provone:hasOutPort` $\rightarrow$ `provone:Port`). Each path represents a retrievable unit of information.

We define a *synthetic question* SQ as a tuple $SQ = (id, sq, Q_{\text{sparql}}, U_{\text{sparql}}, \mathcal{I}, \mathcal{O})$, where id is a unique identifier, sq is the natural language question, Q_{sparql} is the corresponding executable SPARQL query, U_{sparql} is an example usage of Q_{sparql}, and $\mathcal{I}$ and $\mathcal{O}$ denote the input and output specifications, respectively. The generation of these programs proceeds in three phases (Fig. 1) using a schema-guided crawler:

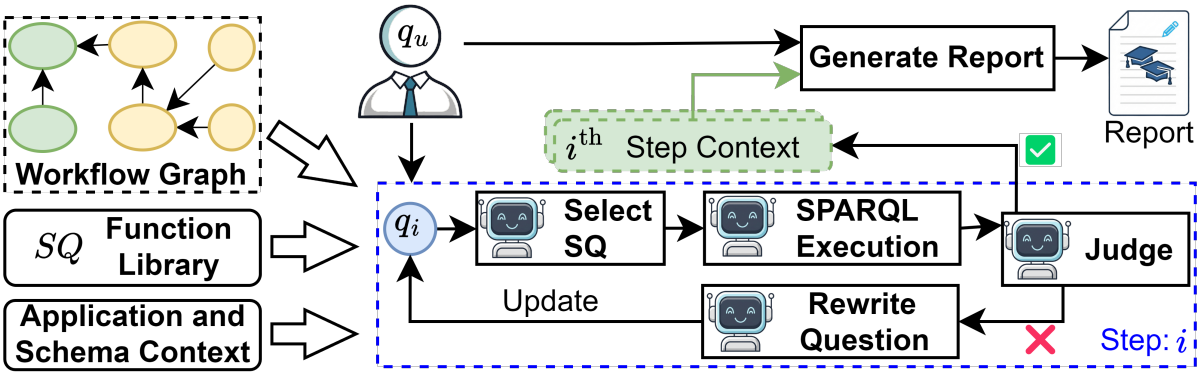


Figure 3: Overview of the retrieval pipeline. Given a user question, the system iteratively selects an SQ, executes its Q_{sparql}, and evaluates the result. If insufficient, the question is updated, and the process repeats. The accumulated context is used to generate the final report.

1. **Schema-Guided Traversal:** We employ a Breadth-First Search (BFS) over the ontology schema (rather than instance data). Starting from a class C_{start} (e.g., `provone:Execution`), the algorithm traverses relationships r to identify valid semantic paths $p = \{C_{start} \xrightarrow{r_1} C_1 \cdots \xrightarrow{r_n} C_n\}$. The traversal depth is bounded by a parameter LP_{max} (set to 5 in all experiments). This process is repeated for each class in the schema, and the set of unique paths is collected.
2. **Query Formulation:**
   For each identified path p, the system constructs parameterized SPARQL queries Q_{sparql} using templating. For every path, two queries are generated corresponding to forward and reverse traversal, enabling retrieval in both directions (i.e., from source to target and vice versa). In parallel, an LLM (using a 1-shot strategy) generates the corresponding natural language question sq by interpreting the path semantics (e.g., "What activity generated this entity?").
3. **Filtering:** To ensure that generated SQs correspond to actual data, we perform a filtering step. Each candidate query is executed on the workflow graph using a sample instantiation U_{sparql}. Queries that return empty results are discarded, ensuring that only SQs that contain entities (useful in the context of retrieval) in the workflow graph are retained.

In addition, we construct SQs for common retrieval operations, including retrieving all objects of a given class, enumerating properties of a given object, and filtering objects of a class based on property values. All generated SQs form a library of functions (the SQ library), where each SQ represents an answerable question over the AI system.

3.4. Explanation Question Answering System

To effectively utilize the SQ library generated in Section 3.3, we develop a neuro-symbolic question-answering system that bridges the gap between unstructured natural-language queries and structured SPARQL-based retrieval over the workflow graph. As illustrated in Figure 3, the system leverages the SQ library as a constrained set of executable retrieval functions. The pipeline also includes an LLM-based *judge* component that evaluates whether the evidence accumulated during retrieval is sufficient to answer the original user question. When the available evidence is insufficient, the judge identifies the missing information and provides feedback that guides the next retrieval step. Given a user question q_u, the system performs a sequence of retrieval steps. At step i, the system maintains a step question q_i, which represents the current information need to be resolved. The process begins with $q_0 = q_u$, and each subsequent step updates q_i using the information retrieved so far and feedback from the judge.

The retrieval process proceeds as an iterative loop:

- **SQ Selection:** Given q_i, the system grounds the information need to relevant ontology classes, relations, and entities to restrict the SQ library to compatible functions. An LLM then selects

one SQ_i from this candidate set. In the initial step, candidates are limited to retrieving all objects of a given class or filtering objects by property values; if this retrieval produces no usable results, an embedding-based entity-linking fallback is used. The LLM receives the question, schema and application context, candidate function identifiers, and entity mentions in the initial step. Subsequent steps additionally incorporate accumulated retrieval context and judge feedback.

- **SPARQL Execution:** The parameterized SPARQL query (Q_{sparql}) in the selected SQ is instantiated using $\mathcal{I}$ and q_i and executed over the workflow graph to retrieve relevant information/entities (e_i).
- **Answer Evaluation:** A summarization component utilizes the retrieved information from the previous and current steps to answer q_i, producing an intermediate answer a_i. The judge then evaluates a_i together with the retrieved evidence to determine whether the accumulated context is sufficient to answer the original user question q_u. If the evidence is insufficient, the judge identifies the missing information required for further retrieval. Otherwise, the current grounded answer is accepted as a_{final} and the loop terminates.
- **Question Update:** If the answer is insufficient, the system updates the step question using an LLM for the next step (q_{i+1}) by incorporating the retrieved information, the judge feedback, and other loop context information. The question focuses on the information still required to answer the original user question.

This loop is executed for a fixed number of steps (set to 3 in our experiments) or until a satisfactory answer is obtained. If the iterative loop still cannot produce a sufficient answer, the system uses a separate class-level fallback that retrieves objects from the available ontology classes. In the final phase, the system aggregates the results obtained for all steps to construct an explanation for the original query q_u. Let $\mathcal{C} = \{(q_i, SQ_i, a_i, e_i)\}_{i=1}^{n}$ denote the set of step records: step-question, executed SQ, intermediate answer, and evidence entity triplets. $\mathcal{C}$ is structured as a report that chains the stepwise retrieval and answers, with a_{final} presented as the final explanation. This design ensures that each component of the final explanation is grounded in verifiable data retrieved from the workflow graph, reducing hallucination and improving interpretability.

4. Evaluation

The primary goal of our evaluation is to assess the end-to-end ability of the framework to generate explanations that accurately address user questions about AI workflow executions. We construct independent evaluation benchmarks for ChatBS-NexGen [2] and Biomni [3], consisting of natural-language user questions and corresponding ground-truth answers. These evaluation questions are distinct from the generated synthetic questions (SQs); rather than evaluating the SQs directly, we measure how effectively the system selects and composes them to retrieve evidence and answer unseen user questions.

We evaluate both the quality of the final explanations, in terms of completeness, reference agreement, and relevance, and the underlying retrieval process through entity recall and exact-answer accuracy for boolean and numeric questions.

4.1. Implementation

The SQ creation and question-answering pipelines are implemented in Python using the DSPy framework [37] for programmatic prompting. Large language models are self-hosted on an NVIDIA DGX server and accessed via LM Studio. Across all methods, we use `gpt-oss-20b` for LLM calls and `text-embedding-bge-large-en-v1.5` for embedding creation.

4.2. Baselines

We evaluate our framework against baselines representing distinct paradigms for querying KGs.

GroundedWorkflowBaseline (GWB). A symbolic, grounding-first approach that answers queries using deterministic heuristics over the workflow graph. It classifies questions (e.g., model, input, lineage) and retrieves evidence using lexical matching and explicit relations (e.g., `prov:used`, `prov:wasGeneratedBy`). While efficient and interpretable, it is limited to predefined patterns and lacks multi-hop reasoning.

VectorSimilarityBaseline (VSB). A retrieval-augmented baseline that embeds KG-derived object descriptions (constructed from RDF subjects, labels, types, and outgoing properties) into a vector space. For each query, the top-k similar objects are retrieved and passed to an LLM for answer generation. While effective for semantic matching, it treats objects independently and does not preserve multi-hop structure, ontology semantics, or predicate directionality.

GRASP. A zero-shot KGQA method that performs iterative SPARQL generation via LLM-guided function calls, refining queries using intermediate feedback [9]. While expressive, it requires dynamic query construction at inference time, introducing latency and sensitivity to entity and query formulation errors.

HippoRAG-KG (HippoRAG). A graph-based RAG baseline adapted from HippoRAG [26]. The KG is transformed into object-level documents and OpenIE-style triples, enabling graph-based retrieval without LLM-based extraction. While this preserves the relational structure, RDF semantics, such as datatype constraints and predicate identity, are flattened into textual representations.

HyperGraphRAG-KG (HyperGRAG). A hypergraph-based RAG baseline adapted from HyperGraphRAG [27]. The KG is converted into entities and hyperedge-like fragments, enabling associative retrieval over structured data. However, this transformation loses RDF-specific semantics such as directionality, datatypes, and ontology constraints.

These baselines span symbolic retrieval, vector-based retrieval, and iterative query synthesis. In contrast, our approach precomputes ontology-grounded SQs, eliminating query generation at inference time and enabling structured, traceable retrieval over the KG. Our GitHub repository provides the full adaptation protocol, including the exact representation exposed to each baseline.

4.3. Evaluation Benchmark Construction

Table 1
A few evaluation question examples categorized by question type.

Question	Category
In what places do we utilize AI in this workflow?	*Entity*
What are the inputs used by the large language model to create the “Extract entities” function?	*Entity*
What is the number of outputs for the “Output port for kg_info” in the execution <http://testwebsite/testProgram#id_20260420105727_300>?	*Numeric*
Did the execution <http://testwebsite/testProgram#id_20260420105710_61> use data that was created by a generative task?	*True/False*
Do any executions of the “Generates a system prompt” function include a generative task as a component?	*True/False*
What are the outputs of the generative task that was a component in executions of the “Chat with the LLM" function?	*Entity*

For each use case, we construct an independent benchmark of natural-language questions representing information that a user may request about the recorded workflow execution. We construct these benchmark questions independently of the SQ library and use them solely as inputs to evaluate the end-to-end explanation system. The curated benchmark datasets consist of manually annotated

natural-language queries. Each dataset entry is formally defined as a tuple $T = (Q, E_{gt}, Q_{\text{sparql}}, S_{ent})$, where:

- Q is the natural language question derived from templated patterns to ensure linguistic consistency while maximizing coverage.
- E_{gt} is the ground truth textual explanation that represents the ideal response expected from the system.
- Q_{sparql} is the corresponding SPARQL query that retrieves the information required to construct E_{gt}.
- S_{ent} is the set of critical knowledge graph entities (URIs and literals) that must be present in a well-grounded explanation. These entities serve as a grounding constraint, ensuring that generated responses are anchored to the correct graph nodes.

We categorize the evaluation questions into three types, as shown in Table 1. *Entity* questions require retrieving specific entities or sets of entities associated with a given concept or execution. *True/False* questions evaluate the system's ability to verify the presence or absence of specific conditions within the workflow. *Numeric* questions assess the system's completeness in retrieval by requiring aggregation over all relevant instances, thereby testing whether the system captures the full extent of the underlying data rather than partial results.

4.4. Evaluation Metrics

To evaluate system performance, we employ an automated LLM-as-a-judge framework that enables scalable assessment against ground-truth annotations. Given a user question Q, ground truth explanation E_{gt}, and model-generated response E_{pred}, an evaluator LLM assigns normalized scores in the range $[0, 1]$ across three dimensions:

1. **Completeness (Comp.):** The extent to which E_{pred} covers the full information content of E_{gt}. A response is considered complete only if all key semantic components present in the ground truth are captured.
2. **Reference Agreement (RA):** The degree to which E_{pred} remains faithful to E_{gt}, penalizing hallucinations, unsupported claims, or contradictions. This metric replaces traditional accuracy by explicitly measuring grounding to the reference explanation.
3. **Relevance (Rel.):** The degree to which the response directly addresses the user question Q, penalizing inclusion of extraneous or off-topic information.

We acknowledge that LLM-as-a-judge evaluation may introduce evaluator bias and may not fully capture whether a response is grounded in the correct KG entities. To address this, we complement semantic scoring with entity-recall metrics and exact-answer accuracy for question types with definitive answers. For entity grounding, we compute:

$$R_{final} = \frac{|S_{ent} \cap S_{ret_final}|}{|S_{ent}|}, \; R_{total} = \frac{|S_{ent} \cap S_{ret_total}|}{|S_{ent}|}$$

where S_{ret_final} denotes entities retrieved in the final step of the execution plan, and S_{ret_total} denotes entities retrieved across all steps. For *True/False* and *Numeric* questions, we compute exact-answer accuracy by comparing them with the ground truth, as they have definitive answers. We denote the accuracies as Acc_{bool} and Acc_{numeric}.

Inspired by prior use of LLM-as-a-judge for evaluating agent-workflow outputs [38], we additionally assess generated explanations through pairwise comparison. Using an automated LLM-as-a-judge framework, the evaluator is presented with two candidate explanations and selects the more understandable and faithful response with respect to the ground truth. We compute win rates as the proportion of pairwise comparisons in which a given method is preferred over competing baselines, with ties counted

as half wins. To reduce evaluator-specific bias, both absolute and pairwise evaluations are repeated using three judges: `Gemma 4 31B`, `Qwen 3.6 35B A3B`, and `GPT-5.4 mini`. We report the equal-weight mean across judges. For completeness, reference agreement, and relevance, we additionally measure item-level inter-judge agreement using interval Krippendorff's α, which accounts for the magnitude of disagreement between continuous $[0, 1]$ scores.

4.5. ChatBS-NexGen Explanation Module

We evaluate the framework on ChatBS-NexGen [2], a four-step LLM experimentation pipeline for validating generated outputs against authoritative knowledge graphs. Three steps use LLM-generated task-oriented functions, making this use case representative of a structured pipeline in which LLMs act as dynamic function generators rather than free-form agents. We focus on its dietary recommendation validation task, where personalized prompts are generated from patient attributes (e.g., diagnosis and health indicators) and LLM-generated food recommendations are validated against DBpedia [39] to examine whether recommended foods were appropriate for diabetic patients by linking food mentions to glycemic index information. We instrument the pipeline using our R library to capture generated functions, LLM calls, intermediate artifacts, and downstream dependencies.

We construct a benchmark comprising 103 questions utilizing 13 multi-variable string templates, each paired with a manually authored SPARQL query to retrieve the corresponding ground-truth answers from the ChatBS-NexGen workflow graph. The dataset is distributed across the following categories: 38 True/False, 49 Entity, and 16 Numeric questions. These questions were used to generate explanation responses from each system under evaluation, and the aggregated results are reported in Table 2.

Table 2
Evaluation results for ChatBS-NexGen [2]. Completeness (Comp.), Reference Agreement (RA), and relevance (Rel.) report the mean ± sample standard deviation across GPT-5.4 mini, Qwen 3.6 35B A3B, and Gemma 4 31B judges, using all question types. Missing ratings are excluded. Final and total entity recall (R_{final}, R_{total}) use entity questions only; boolean and numeric accuracy use their respective question types. All scores are percentages. Best means are in bold.

Method	Comp.	RA	Rel.	R_{final}	R_{total}	Acc_{bool}	Acc_{numeric}
GWB	12.70 ± 0.65	18.38 ± 5.27	26.07 ± 5.68	29.90	56.84	0.00	12.50
VSB	21.54 ± 0.61	27.42 ± 5.21	59.16 ± 17.68	10.59	18.93	34.21	6.25
GRASP	18.05 ± 3.52	22.48 ± 1.82	36.20 ± 6.96	0.00	0.00	28.95	12.50
HippoRAG	27.23 ± 3.72	31.28 ± 2.27	69.09 ± 11.27	34.55	**64.79**	52.63	25.00
HyperGRAG	38.68 ± 3.75	48.20 ± 6.38	61.23 ± 7.62	0.00	0.00	42.11	25.00
Ours	$\mathbf{69.24} \pm 5.15$	$\mathbf{65.44} \pm 8.49$	$\mathbf{91.97} \pm 8.68$	**38.37**	49.42	**71.05**	**43.75**

4.6. Biomni Explanation Module

Biomni [3] provides a contrasting free-form agentic use case, combining LLM-based planning, tool selection, code execution, and a critic loop producing complex execution traces across biomedical resources. We evaluate our explanation module on a 12-step execution for the user request: "Predict ADMET properties for this compound: CC(C)CC1=CC=C(C=C1)C(C)C(=O)O." with access to a BioConda-based environment containing biomedical Python, R, and command-line tools, including chemistry and drug-discovery libraries such as RDKit, DeepPurpose, OpenBabel, and PyTDC. We instrument the execution using our Python library to capture LLM calls, tool invocations, generated code, intermediate artifacts, and outputs. This produces a more dynamic execution trace than the fixed ChatBS-NexGen pipeline.

We construct a benchmark comprising 106 questions utilizing 22 multi-variable string templates, each paired with a manually authored SPARQL query to retrieve the corresponding ground-truth answers from the Biomni workflow graph. The dataset is distributed across the following categories: 39

True/False, 46 Entity, and 21 Numeric questions. These questions were used to generate explanation responses from each system under evaluation, and the aggregated results are reported in Table 3.

Table 3
Evaluation results for Biomni. Completeness (Comp.), Reference Agreement (RA), and relevance (Rel.) report the mean ± sample standard deviation across GPT-5.4 mini, Qwen 3.6 35B A3B, and Gemma 4 31B judges, using all question types. Missing ratings are excluded. Final and total entity recall (R_{final}, R_{total}) use entity questions only; boolean and numeric accuracy use their respective question types. All scores are percentages. Best means are bolded.

Method	Comp.	RA	Rel.	R_{final}	R_{total}	Acc_{bool}	Acc_{numeric}
GWB	7.64 ± 3.06	9.89 ± 4.33	26.66 ± 9.18	33.60	36.73	0.0	14.29
VSB	22.98 ± 2.46	30.03 ± 4.17	63.96 ± 13.89	5.64	15.58	43.59	0.0
GRASP	20.70 ± 3.16	22.34 ± 3.51	48.10 ± 8.08	2.27	2.27	38.46	19.05
HippoRAG	28.78 ± 3.27	32.39 ± 1.93	73.96 ± 6.85	39.14	**66.58**	48.72	33.33
HyperGRAG	50.22 ± 5.57	50.14 ± 3.26	79.74 ± 7.02	-	-	53.85	28.57
Ours	$\mathbf{68.45} \pm 6.10$	$\mathbf{65.47} \pm 8.49$	$\mathbf{93.75} \pm 8.12$	**46.81**	55.68	**76.92**	**52.38**

4.7. Comparative Study

Across both benchmarks, our approach achieves the highest completeness, reference agreement, relevance, final-step entity recall, and boolean/numeric accuracy. Although HippoRAG achieves higher total entity recall, this does not translate into stronger explanation quality. Our method instead retrieves a larger proportion of the required entities in the final step while producing explanations that more closely align with the ground truth. Relevance reaches 91.97% on ChatBS-NexGen and 93.75% on Biomni, indicating that the final explanations remain highly focused on the user question. The stepwise examples in our repository (docs/report_example.md and docs/example_stepwise_execution.md) illustrate how iterative retrieval narrows the information used in the final explanation. Numeric accuracy is comparatively lower on both benchmarks, though it remains the best among all methods. Error analysis identified two recurring causes: incorrect ontology-class grounding for the requested quantity and incorrect LLM counting even when the relevant entities were retrieved. The post-loop class-level fallback was used in 17/103 ChatBS-NexGen questions (16.5%) and 29/106 Biomni questions (27.4%).

Inter-judge agreement. Item-level agreement among the LLM judges was measured using interval Krippendorff's α. For ChatBS-NexGen, agreement was 0.856 for completeness, 0.672 for reference agreement, and 0.656 for relevance. For Biomni, the corresponding values were 0.841, 0.719, and 0.771. Agreement was therefore strongest for completeness across both benchmarks, while reference agreement and relevance showed comparatively greater variation among judges.

4.8. Win Rate Analysis

As shown in Fig. 4, our method outperforms every baseline on both benchmarks. On ChatBS-NexGen, win rates range from 65.4% against HyperGRAG to 92.8% against GWB, while on Biomni they range from 64.3% against HyperGRAG to 91.1% against GWB. Eight of the ten comparisons exceed an 80% win rate, and all remain above the 50% parity threshold. HyperGRAG provides the closest comparison on both benchmarks, but our approach remains preferred overall.

Overall, these results demonstrate that our approach consistently produces preferable explanations compared to all baselines, with particularly strong gains over symbolic and query-generation methods and robust improvements across both datasets.

4.9. Latency and Call Count Analysis

Table 4 summarizes average latency and call counts. VSB and graph-RAG baselines require fewer LLM calls and achieve lower latency, while GRASP incurs additional overhead from iterative SPARQL

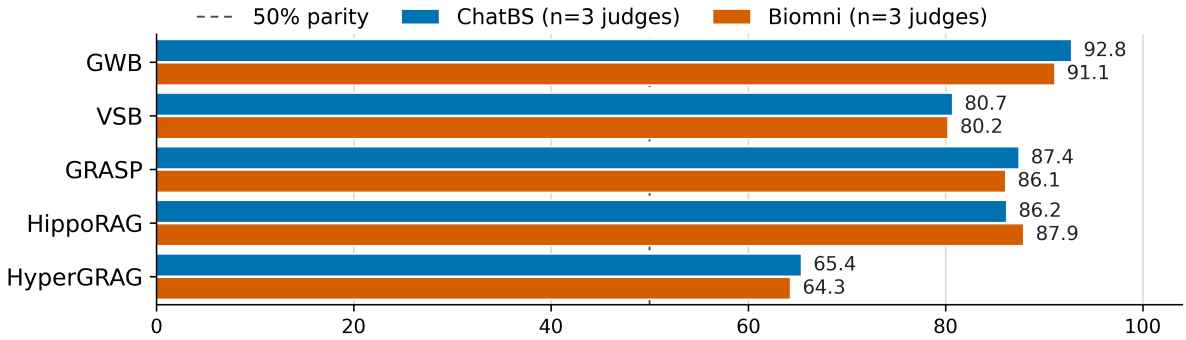


Figure 4: Pairwise answer win rates of our method against each baseline on the ChatBS and Biomni evaluations. Values show the percentage of pairwise comparisons won by our method. The dashed line marks the 50% parity threshold.

generation. Our method has the highest latency, particularly on Biomni, due to repeated SQ selection, parameter grounding, answer evaluation, and refinement. This reflects a trade-off between explanation quality and inference efficiency.

Table 4
Average latency (s) and call counts across both benchmarks. SPARQL denotes explicit query-bearing calls; dashes indicate unavailable metrics.

Method	ChatBS-NexGen			Biomni		
	Latency	**LLM**	**SPARQL**	**Latency**	**LLM**	**SPARQL**
GWB	0.89	0.00	-	0.91	0.00	-
VSB	2.94	1.00	-	3.18	1.00	-
GRASP	15.49	7.79	5.79	10.57	5.06	1.63
HippoRAG	4.59	1.00	-	4.83	1.00	-
HyperGRAG	12.67	1.74	-	14.11	1.91	-
Ours	39.98	6.95	2.59	146.06	9.32	2.74

4.10. Developer Workload

Table 5 reports the provenance-instrumentation effort. Both systems required approximately 30 annotation calls, while Biomni required more code to capture dynamic agent iterations, tool execution, critic steps, and intermediate artifacts. Instrumentation time was not recorded.

Table 5
Provenance instrumentation overhead measured by files edited, number of instrumentation function calls, lines of code added (LOC), and triples collected.

System	**Files**	**Calls**	**LOC**	**Triples**
ChatBS-NexGen	5	30	870	895
Biomni	3	31	1,172	2,022

5. Conclusion and Future Work

We presented a hybrid framework for explaining multi-stage AI systems by combining semantic workflow representation with constrained, provenance-grounded retrieval. Using ProvONE and EO, execution traces are transformed into a knowledge graph and queried through a precomputed library of executable SPARQL functions. Evaluations on ChatBS-NexGen [2] and Biomni [3] show that our approach achieves the highest completeness, reference agreement, relevance, and boolean/numeric

accuracy across both use cases. Pairwise evaluation further shows that our explanations are preferred over all baselines, with win rates above the 50% parity threshold and exceeding 80% in most comparisons.

Several limitations remain. Developer-driven provenance instrumentation introduces engineering overhead, while the multi-step LLM pipeline increases latency. In the current implementation, we focus on trace-based and data explanations, and the system can be extended to other explanation types [13]. Explanation completeness also depends on SQ library coverage, and further user-centered evaluation is needed to assess interpretability and practical utility. Future work will focus on simplifying instrumentation, reducing inference latency, and expanding SQ coverage. Overall, the results show that ontology-grounded, executable retrieval over workflow provenance provides a practical basis for more transparent and verifiable AI systems.

Supplemental Material Statement: Source code for our system, baselines (some available through submodules), benchmarks, and raw results are available at https://github.com/DevinDeSilva/LLM-Workflow-Explorer.

Declaration on Generative AI

During the preparation of this work, the author(s) used GPT-5 in order to: Grammar and spelling check. After using these tool(s)/service(s), the author(s) reviewed and edited the content as needed and take(s) full responsibility for the publication's content.

References

[1] X. Wang, Y. Chen, L. Yuan, Y. Zhang, Y. Li, H. Peng, H. Ji, Executable code actions elicit better llm agents, in: Proceedings of the 41st International Conference on Machine Learning, ICML'24, JMLR.org, 2024.

[2] J. S. Erickson, H. Santos, V. Pinheiro, J. P. McCusker, D. L. McGuinness, LLM experimentation through knowledge graphs: Towards improved management, repeatability, and verification, Journal of Web Semantics 85 (2025) 100853.

[3] K. Huang, S. Zhang, H. Wang, Y. Qu, Y. Lu, Y. Roohani, R. Li, L. Qiu, G. Li, J. Zhang, D. Yin, S. Marwaha, J. N. Carter, X. Zhou, M. Wheeler, J. A. Bernstein, M. Wang, P. He, J. Zhou, M. Snyder, L. Cong, A. Regev, J. Leskovec, Biomni: A general-purpose biomedical AI agent, bioRxiv : the preprint server for biology (2025). doi:10.1101/2025.05.30.656746.

[4] P. Q. Le, M. Nauta, V. B. Nguyen, S. Pathak, J. Schlötterer, C. Seifert, Benchmarking eXplainable AI - A Survey on Available Toolkits and Open Challenges, in: Proceedings of the Thirty-Second International Joint Conference on Artificial Intelligence, International Joint Conferences on Artificial Intelligence Organization, Macau, SAR China, 2023, pp. 6665–6673. doi:10.24963/ijcai.2023/747.

[5] G. Liu, S. Solomon, Ai agent observability: Evolving standards and best practices, https://opentelemetry.io/blog/2025/ai-agent-observability/, 2025. URL: https://opentelemetry.io/blog/2025/ai-agent-observability/, openTelemetry Blog. Accessed: 2026-05-06.

[6] AgentOps-AI, Agentops: Python sdk for ai agent monitoring, llm cost tracking, benchmarking, and more, https://github.com/AgentOps-AI/agentops, 2025. GitHub repository; latest release listed as 0.4.21 on August 29, 2025; accessed May 7, 2026.

[7] C. Mavromatis, S. Adeshina, V. N. Ioannidis, Z. Han, Q. Zhu, I. Robinson, B. Thompson, H. Rangwala, G. Karypis, BYOKG-RAG: Multi-strategy graph retrieval for knowledge graph question answering, in: C. Christodoulopoulos, T. Chakraborty, C. Rose, V. Peng (Eds.), Proceedings of the 2025 Conference on Empirical Methods in Natural Language Processing, Association for Computational Linguistics, Suzhou, China, 2025, pp. 27869–27886. URL: https://aclanthology.org/2025.emnlp-main.1417/. doi:10.18653/v1/2025.emnlp-main.1417.

[8] D. Agarwal, R. Das, S. Khosla, R. Gangadharaiah, Bring Your Own KG: Self-Supervised Program Synthesis for Zero-Shot KGQA, in: K. Duh, H. Gomez, S. Bethard (Eds.), Findings of the Association

for Computational Linguistics: NAACL 2024, Association for Computational Linguistics, Mexico City, Mexico, 2024, pp. 896–919. doi:10.18653/v1/2024.findings-naacl.57.

[9] S. Walter, H. Bast, Grasp: Generic reasoning and sparql generation across knowledge graphs, in: D. Garijo, S. Kirrane, A. Salatino, C. Shimizu, M. Acosta, A. G. Nuzzolese, S. Ferrada, T. Soulard, K. Kozaki, H. Takeda, A. L. Gentile (Eds.), The Semantic Web – ISWC 2025, Springer Nature Switzerland, Cham, 2026, pp. 271–289.

[10] S. Liu, S. Semnani, H. Triedman, J. Xu, I. D. Zhao, M. Lam, SPINACH: SPARQL-based information navigation for challenging real-world questions, in: Y. Al-Onaizan, M. Bansal, Y.-N. Chen (Eds.), Findings of the Association for Computational Linguistics: EMNLP 2024, Association for Computational Linguistics, Miami, Florida, USA, 2024, pp. 15977–16001. URL: https://aclanthology.org/2024.findings-emnlp.938/. doi:10.18653/v1/2024.findings-emnlp.938.

[11] Y. Zhang, K. Tsuda, Saga: Schema-aware grounding for agentic text-to-sparql generation, 2026. URL: https://arxiv.org/abs/2607.14494. arXiv:2607.14494.

[12] Y.-H. Chen, E. J.-L. Lu, C.-N. Tsao, Enhancing sparql query generation using multi-label text-to-text models, Data & Knowledge Engineering 164 (2026) 102584. URL: https://www.sciencedirect.com/science/article/pii/S0169023X26000315. doi:10.1016/j.datak.2026.102584.

[13] S. Chari, O. Seneviratne, D. M. Gruen, M. A. Foreman, A. K. Das, D. L. McGuinness, Explanation ontology: A model of explanations for user-centered ai, in: J. Z. Pan, V. Tamma, C. d'Amato, K. Janowicz, B. Fu, A. Polleres, O. Seneviratne, L. Kagal (Eds.), The Semantic Web – ISWC 2020, Springer International Publishing, Cham, 2020, pp. 228–243.

[14] Y. Cao, C. Jones, V. Cuevas, M. Jones, B. Ludäscher, T. McPhillips, P. Missier, C. Schwalm, P. Slaughter, D. Vieglais, L. Walker, Y. Wei, ProvONE: extending PROV to support the DataONE scientific community (2016).

[15] L. Vonderhaar, T. T. Procko, O. Ochoa, A survey of machine learning lifecycle provenance: Models, approaches, and tools, in: M. Mannion, T. Männistö, L. A. Maciaszek (Eds.), Proceedings of the 21st International Conference on Evaluation of Novel Approaches to Software Engineering, ENASE 2026, Benidorm, Spain, May 22-24, 2026, Volume 2, SCITEPRESS, 2026, pp. 1059–1068. URL: https://doi.org/10.5220/0014732400004015. doi:10.5220/0014732400004015.

[16] M. Schlegel, K.-U. Sattler, Capturing end-to-end provenance for machine learning pipelines, Information Systems 132 (2025) 102495. URL: https://www.sciencedirect.com/science/article/pii/S0306437924001534. doi:10.1016/j.is.2024.102495.

[17] T. Lebo, S. Sahoo, D. McGuinness, K. Belhajjame, J. Cheney, D. Corsar, D. Garijo, S. Soiland-Reyes, S. Zednik, J. Zhao, PROV-O: The PROV Ontology, W3C Recommendation, World Wide Web Consortium, United States, 2013.

[18] K. O'Sullivan, M. Markovic, J. Dymiter, B. Scheliga, C. Odo, K. Wilde, Semi-automated data provenance tracking for transparent data production and linkage to enhance auditing and quality assurance in trusted research environments, Int J Popul Data Sci 10 (2025) 2464.

[19] R. L. de Oliveira, J. C. Duarte, K. de Faria Cordeiro, Machine learning model explainability supported by data explainability: a provenance-based approach, Journal of Information and Data Management 15 (2024) 93–102. URL: https://journals-sol.sbc.org.br/index.php/jidm/article/view/3337. doi:10.5753/jidm.2024.3337.

[20] S. M. Lundberg, S.-I. Lee, A unified approach to interpreting model predictions, in: Proceedings of the 31st International Conference on Neural Information Processing Systems, NIPS'17, Curran Associates Inc., Red Hook, NY, USA, 2017, p. 4768–4777.

[21] M. T. Ribeiro, S. Singh, C. Guestrin, "why should i trust you?": Explaining the predictions of any classifier, in: Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, KDD '16, Association for Computing Machinery, New York, NY, USA, 2016, p. 1135–1144. URL: https://doi.org/10.1145/2939672.2939778. doi:10.1145/2939672.2939778.

[22] N. Targino, D. Souza, A. C. Salgado, An approach to associate provenance from ml pipelines with transparency and accountability, in: 2024 15th International Conference on Information, Intelligence, Systems & Applications (IISA), 2024, pp. 1–8. doi:10.1109/IISA62523.2024.107

86622.

[23] E. Daga, P. Groth, Data journeys: Explaining AI workflows through abstraction, Semantic Web 15 (2024) 1057–1083. URL: https://journals.sagepub.com/doi/abs/10.3233/SW-233407. doi:10.3233/SW-233407.

[24] R. Souza, A. Gueroudji, S. DeWitt, D. Rosendo, T. Ghosal, R. Ross, P. Balaprakash, R. F. Da Silva, Prov-agent: Unified provenance for tracking ai agent interactions in agentic workflows, in: 2025 IEEE International Conference on eScience (eScience), 2025, pp. 467–473. doi:10.1109/eScience65000.2025.00093.

[25] Y. Cai, Z. Guo, Y. Pei, W. Bian, W. Zheng, SimGRAG: Leveraging similar subgraphs for knowledge graphs driven retrieval-augmented generation, in: W. Che, J. Nabende, E. Shutova, M. T. Pilehvar (Eds.), Findings of the Association for Computational Linguistics: ACL 2025, Association for Computational Linguistics, Vienna, Austria, 2025, pp. 3139–3158. URL: https://aclanthology.org/2025.findings-acl.163/. doi:10.18653/v1/2025.findings-acl.163.

[26] B. J. Gutiérrez, Y. Shu, W. Qi, S. Zhou, Y. Su, From rag to memory: Non-parametric continual learning for large language models, 2025. URL: https://arxiv.org/abs/2502.14802. arXiv:2502.14802.

[27] H. Luo, H. E, G. Chen, Y. Zheng, X. Wu, Y. Guo, Q. Lin, Y. Feng, Z. Kuang, M. Song, Y. Zhu, L. A. Tuan, Hypergraphrag: Retrieval-augmented generation via hypergraph-structured knowledge representation, 2025. URL: https://arxiv.org/abs/2503.21322. arXiv:2503.21322.

[28] V. Emonet, J. Bolleman, S. Duvaud, T. Mendes de Farias, A. C. Sima, LLM-based SPARQL query generation from natural language over federated knowledge graphs, in: ISWC 2024 Special Session on Harmonising Generative AI and Semantic Web Technologies, November 13, 2024, Baltimore, Maryland, volume 3953 of *CEUR Workshop Proceedings*, 2025. CEUR-WS.org, online https://ceur-ws.org/Vol-3953/355.pdf.

[29] P. Smeros, V. Emonet, R. Wang, A.-C. Sima, T. M. de Farias, SPARQL-LLM: Real-time SPARQL query generation from natural language questions, 2025. URL: https://arxiv.org/abs/2512.14277. arXiv:2512.14277.

[30] X. Pan, V. de Boer, J. van Ossenbruggen, FIRESPARQL: A LLM-based framework for SPARQL query generation over scholarly knowledge graphs, 2025. URL: https://arxiv.org/abs/2508.10467. arXiv:2508.10467.

[31] A. Gashkov, A. Perevalov, M. Eltsova, A. Both, SPARQL query generation with LLMs: Measuring the impact of training data memorization and knowledge injection, in: Web Engineering: 25th International Conference, ICWE 2025, Delft, Netherlands, June 30-July 3, 2025, Proceedings, 2025. Winner of Best Paper Award at the 25th International Conference on Web Engineering (ICWE 2025).

[32] Y. Taghzouti, F. Michel, T. Jiang, L. F. Nothias, F. Gandon, Q²forge: Minting competency questions and sparql queries for question-answering over knowledge graphs, in: Proceedings of the 13th Knowledge Capture Conference 2025, K-CAP '25, Association for Computing Machinery, New York, NY, USA, 2025, p. 74–81. URL: https://doi.org/10.1145/3731443.3771350. doi:10.1145/3731443.3771350.

[33] F. Ciroku, J. de Berardinis, J. Kim, A. Meroño-Peñuela, V. Presutti, E. Simperl, Revont: Reverse engineering of competency questions from knowledge graphs via language models, Journal of Web Semantics 82 (2024) 100822. URL: https://www.sciencedirect.com/science/article/pii/S1570826824000088. doi:10.1016/j.websem.2024.100822.

[34] R. Liu, B. Luo, J. Li, B. Wang, M. Liu, D. Wu, S. Wang, B. Qin, Ontology-guided reverse thinking makes large language models stronger on knowledge graph question answering, in: W. Che, J. Nabende, E. Shutova, M. T. Pilehvar (Eds.), Proceedings of the 63rd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers), Association for Computational Linguistics, Vienna, Austria, 2025, pp. 15269–15284. URL: https://aclanthology.org/2025.acl-long.741/. doi:10.18653/v1/2025.acl-long.741.

[35] M. Dumontier, C. J. Baker, J. Baran, A. Callahan, L. Chepelev, J. Cruz-Toledo, N. R. Del Rio, G. Duck, L. I. Furlong, N. Keath, D. Klassen, J. P. McCusker, N. Queralt-Rosinach, M. Samwald, N. Villanueva-Rosales, M. D. Wilkinson, R. Hoehndorf, The Semanticscience Integrated Ontology (SIO) for

biomedical research and knowledge discovery, Journal of Biomedical Semantics 5 (2014) 14. URL: http://sio.semanticscience.org. doi:10.1186/2041-1480-5-14.

[36] D. Beckett, T. Berners-Lee, E. Prud'hommeaux, G. Carothers, RDF 1.1 Turtle: Terse RDF Triple Language, W3C Recommendation, World Wide Web Consortium (W3C), 2014. URL: http://www.w3.org/TR/2014/REC-turtle-20140225/, latest version available at http://www.w3.org/TR/turtle/.

[37] O. Khattab, A. Singhvi, P. Maheshwari, Z. Zhang, K. Santhanam, S. V. A, S. Haq, A. Sharma, T. T. Joshi, H. Moazam, H. Miller, M. Zaharia, C. Potts, DSPy: Compiling declarative language model calls into state-of-the-art pipelines, in: The Twelfth International Conference on Learning Representations, 2024. URL: https://openreview.net/forum?id=sY5N0zY5Od.

[38] X. Wang, M. Yin, E. Koh, M. D. Dogan, Xagen: An explainability tool for identifying and correcting failures in multi-agent workflows, in: Proceedings of the Extended Abstracts of the 2026 CHI Conference on Human Factors in Computing Systems, CHI EA '26, Association for Computing Machinery, New York, NY, USA, 2026. URL: https://doi.org/10.1145/3772363.3798531. doi:10.1145/3772363.3798531.

[39] S. Auer, C. Bizer, G. Kobilarov, J. Lehmann, R. Cyganiak, Z. Ives, DBpedia: A Nucleus for a Web of Open Data, in: The Semantic Web, Springer, 2007, pp. 722–735.

A. Report Example

Figure 5: Example report provided by the system.

```
User Question: Is the function with a description as "Generates a system
prompt" made by using Large Language Model output?
```

```
KG Question: What are all the objects of class workflow:Generative_Task
with a given value for the relation/property dc:description?
Result Entities:
ChatBS-NexGen:system_prompt_generator
ChatBS-NexGen:system_prompt_generator[provone:Program]
  provone:hasInPort -> ChatBS-NexGen:system_prompt_generator-weight
                       [provone:Port](Input port for weigh ...)
  dc:description -> Generates a system prompt based on the provided
                    dataset structure and user input.@en^^<xsd:string>
        ...

Rewritten Question: Given the current evidence,focus on the following issue: The answer
does not provide any explicit literal values. To satisfy the
requirement, the answer should include specific literals from the
dataset—such as the dc:description value for
ChatBS-NexGen:system_prompt_generator—and any relevant SIO_000202
triple indicating a link to a Large Language Model Output instance.
Without these literal details the answer is incomplete.

KG Question: What type of output does this AI task produce?
Result Entities:
ChatBS-NexGen:LLM_Output-system_prompt_generator
ChatBS-NexGen:LLM_Output-system_prompt_generator
  [workflow:Large_Language_Model_Output]
  sio:SIO_000202 -> ChatBS-NexGen:system_prompt_generator
    [provone:Program](System Prompt Templa ...)
  sio:SIO_000232 -> ChatBS-NexGen:LLM-system_prompt_generator
    [workflow:Large_Language_Model](System Prompt Templa ...)

Summary Answer:
The function identified as **ChatBS-NexGen:system_prompt_generator**
has the literal description
‘‘‘
dc:description = "Generates a system prompt based on the provided
dataset structure and user input."
‘‘‘
In the provenance graph, this program is linked to an output that
is explicitly typed as a Large Language Model Output. The triple
‘‘‘
ChatBS-NexGen:LLM_Output-system_prompt_generator  sio:SIO_000202
ChatBS-NexGen:system_prompt_generator
‘‘‘
indicates that the LLM output was produced *by* (i.e., generated from)
the program **system_prompt_generator**. Because the output entity
is typed as ‘workflow:Large_Language_Model_Output‘, we can conclude
that the function’s result—i.e., the system prompt—is indeed
created using a Large Language Model. Thus, **yes**, the function
with the description “Generates a system prompt” is made by
using Large Language Model output.
```

B. Example stepwise execution

Figure 6: Example of iterative, evidence-grounded retrieval. The system executes SQs over the workflow graph, using retrieved entities to refine the question and accumulate evidence across steps, enabling a grounded explanation.

```
User Question: Was the input to the "Input port for text" in the
execution "http://testwebsite/testProgram#id_20260420105703_307"
generated by a generative task?
```

```
Step 0:

### Question:
Was the input to the "Input port for text" in the execution
"http://testwebsite/testProgram#id_20260420105703_307" generated by
a Generative Task?

### SQ:
What are the attributes of the object?
### Entities:
http://testwebsite/testProgram#id_20260420105703_307[provone:Execution]
  dcterms:identifier -> information_extraction_1_1@en^^<xsd:string>
  prov:used -> ChatBS-NexGen:Data-id_20260420105659_338-generated_answer
               [provone:Data](generated_answer@en^ ...)
        ...

Step 1:

### Question:
Does the data entity ‘ChatBS-NexGen:Data-id_20260420105659_338-
generated_answer‘ have a ‘prov:qualifiedGeneration‘ that links it to a
generation event produced by a ‘workflow:Generative_Task‘, and if so,
what is the URI of that generation event?

### SQ:
What execution produced a given data item?
### Entities:
ChatBS-NexGen:id_20260420105659_302[provone:Execution]
  dcterms:identifier -> llm_chat_1_1@en^^<xsd:string>
  sio:SIO_000369 -> ChatBS-NexGen:Generative_Task-id_20260420105659_302
                    [workflow:Generative_Task](LLM Generates an tex ...)
        ...
```

C. SQ Information

C.1. SQ Library Metadata

To characterize the coverage of the generated SQ libraries, we report the schema traversal and filtering statistics for both workflow graphs. This metadata addresses the need to make the SQ generation process more transparent by showing how many ontology-derived paths were explored, how many candidate SQs were produced, and how many remained after graph-specific filtering. As shown in Table 6, both datasets traverse the same ontology schema, covering 14 classes and 385 raw schema paths. However, the final number of retained SQs differs across datasets because the filtering step removes candidate queries that do not return results over the corresponding workflow graph. ChatBS-NexGen retains 252 of 314 generated SQs, while Biomni retains 208 of 270 generated SQs.

Table 6
Synthetic question generation metadata for ChatBS-NexGen and Biomni. Path length is measured as the number of class nodes in the schema path.

Dataset	Schema Traversal		SQ Generation		Path Length
	Classes	Raw Paths	Raw SQs	Filtered SQs	Avg.
ChatBS-NexGen	14	385	314	252	4.03
Biomni	14	385	270	208	4.03

SQ Instance

ID: `explore_path_workflow:Large_Language_Model_to_provone:Execution`

Semantic Path:
```
workflow:Large_Language_Model -> sio:SIO_000230 -> provone:Data
                    -> prov:wasGeneratedBy -> provone:Execution
```

Synthetic Question (SQ):
What execution(s) generated the data that was used as input in the Large Language Model workflow?

Input Specification ($\mathcal{I}$): `{"obj": "The URI of the starting object."}`
Output Specification ($\mathcal{O}$): `{"value": "The URI of the ending object."}`

SPARQL Query Template (Q_{sparql}):
```
SELECT DISTINCT ?value WHERE {
  <{obj}> sio:SIO_000230 ?a1 .
  ?a1 prov:wasGeneratedBy ?value .
}
```

Example Instantiated Query (U_{sparql}):
```
SELECT DISTINCT ?value WHERE {
  <http://testwebsite/testProgram#LLM-id_20260324180131_821>
      sio:SIO_000230 ?a1 .
  ?a1 prov:wasGeneratedBy ?value .
}
```

Example Output:
```
{"value": ["http://testwebsite/testProgram#id_20260324180129_451"]}
```

Figure 7: Example synthetic question instance generated from an ontology path. The instance specifies the semantic path, natural-language question, input/output specification, SPARQL query template, instantiated query, and example output.

C.2. SQ VS Evaluation Questions

The evaluation questions were not used during SQ generation, SQ selection prompt construction, or retrieval-pipeline development. Although both SQs and evaluation questions are grounded in the same workflow ontology, the evaluation SPARQL queries were manually authored after SQ library construction and were used only to derive ground-truth answers and entity sets (available at https://github.com/DevinDeSilva/LLM-Workflow-Explorer/blob/main/evaluations/biomni-base/ground_truth.ipynb). The SQ natural-language questions were not inserted into the evaluation set. Thus, the evaluation measures whether the system can select and compose SQs to answer held-out natural-language questions, rather than whether it can reproduce known SQ templates.

C.3. SQ Example

This example retrieves the execution instances responsible for generating data that was used as input to a Large Language Model within the workflow.