

Ontology-Grounded RAG with SPARQL and Local Language Models for Technical Documentation

RAG basado en ontologías con SPARQL y modelos de lenguaje locales para documentación técnica

Julián Rafael Coronel Andrade¹ <https://orcid.org/0009-0000-6514-8469>, Edwin Alexander Siguenza Siguenza¹ <https://orcid.org/0009-0001-0337-4644>, Andrew Sebastián Cantos Vasquez¹ <https://orcid.org/0009-0001-1456-8153>, Pablo Sebastián Vélez Mejía¹ <https://orcid.org/0009-0006-5541-1490>, Marcos Orellana¹ <https://orcid.org/0000-0002-3671-9362>

¹ Universidad del Azuay, Computer Science Research & Development Laboratory (LIDI), , Cuenca, Ecuador

julian.coronel@es.uazuay.edu.ec,
edwin.siguenza@es.uazuay.edu.ec, andrew.cantos@es.uazuay.edu.ec,
pablo.velez@es.uazuay.edu.ec, marore@uazuay.edu.ec



Esta obra está bajo una licencia internacional
Creative Commons Atribución-NoComercial 4.0

Sent: 2026/07/12

Accepted: 2026/08/26

Published: 2026/10/14

Abstract

This paper presents LIDI-AI, a Retrieval-Augmented Generation (RAG) system whose knowledge base consists of technical documentation for the LIDI-AI Research Assistant, a web-based data analysis application developed at Universidad del Azuay. The runtime architecture integrates two complementary knowledge sources: an OWL 2 ontology (artifact version 2.0) with six classes and twenty-five properties (nine object and sixteen datatype properties) that models the five-stage pipeline, the accepted file formats, the statistical analysis types, and the system errors, and a SPARQL 1.1 query layer for structured intent resolution. A Prolog knowledge base built on an Object-Attribute-Value (OAV) representation, comprising 172 facts, 17 inference rules, and three diagnostic engines, is formally specified and validated as a symbolic reasoning layer, although it is not yet callable from the Python orchestrator at runtime. A lexical retrieval module selects ontology fragments as grounded context for a locally hosted language model, specifically Llama-3.2-3B deployed via the Ollama framework with Q4_K_M quantization. Evaluation on ten queries from real users shows that seven were resolved with high precision via

Summary: Introduction, Materials and Methods, Results, Discussion, Conclusions.

How to cite: Coronel, J., Siguenza, E., Cantos, A., Vélez, P. & Orellana, M. (2026). Ontology-Grounded RAG with SPARQL and Local Language Models for Technical Documentation. *Revista Tecnológica - Espol*, 38(E1), 84-99. <https://doi.org/10.37815/rte.v38nE1.1603>

SPARQL-grounded retrieval, two were resolved via lexical fallback with ontology traceability, and one out-of-domain query received an explicit knowledge-limit acknowledgment, yielding an appropriate response rate with zero hallucinations.

Keywords: intelligent assistant, knowledge representation, local language models, SPARQL, structured queries, symbolic reasoning

Resumen

Este artículo presenta LIDI-AI, un sistema de Generación Aumentada por Recuperación (RAG) cuya base de conocimiento es la documentación técnica del Asistente de Investigación LIDI-AI, una aplicación web de análisis de datos desarrollada en la Universidad del Azuay. La arquitectura en tiempo de ejecución integra dos fuentes de conocimiento complementarias: una ontología OWL 2 (artefacto versión 2.0) con seis clases y veinticinco propiedades (nueve de objeto y dieciséis de datos) que modela el flujo de trabajo de cinco etapas, los formatos de archivo aceptados, los tipos de análisis estadístico y los errores del sistema; y una capa de consultas SPARQL 1.1 para la resolución estructurada de intenciones. Una base Prolog con representación Objeto-Atributo-Valor (OAV), compuesta por 172 hechos, 17 reglas de inferencia y tres motores de diagnóstico, se especifica y valida formalmente como capa de razonamiento simbólico, aunque aún no es invocable desde el orquestador Python en tiempo de ejecución. Un módulo de recuperación léxica selecciona fragmentos de la ontología como contexto fundamentado para un modelo de lenguaje alojado localmente, en concreto, Llama-3.2-3B desplegado a través del framework Ollama con cuantización Q4_K_M.. La evaluación de diez consultas de usuarios reales muestra que siete se resolvieron con alta precisión mediante recuperación fundamentada en SPARQL, dos mediante recuperación léxica con trazabilidad ontológica, y una consulta fuera de dominio recibió un reconocimiento explícito de los límites del conocimiento, logrando una tasa de respuesta adecuada sin alucinaciones.

Palabras clave: asistente inteligente, consultas estructuradas, modelos locales de lenguaje, razonamiento simbólico, representación del conocimiento, SPARQL

Introduction

Large Language Models (LLMs) have revolutionized natural language processing, yet they suffer from fundamental limitations, including hallucination, knowledge staleness, and lack of interpretability. Retrieval-Augmented Generation (RAG) represents a paradigmatic response to these constraints, enabling models to ground their outputs in external, verifiable information sources (Lewis et al., 2020). Recent developments extend RAG architectures beyond simple text retrieval to encompass structured knowledge representation, logical reasoning, and semantic query processing. The integration of the Web Ontology Language (World Wide Web Consortium, 2012), logic programming paradigms, and the SPARQL 1.1 query language (Harris & Seaborne, 2013) within RAG pipelines creates systems capable of sophisticated reasoning while maintaining factual accuracy. Furthermore, the emergence of efficient local inference engines, such as Ollama, an open-source framework for serving quantized large language models on commodity hardware, enables the deployment of these systems without relying on cloud-based APIs, thereby addressing privacy, cost, and network availability considerations.

User-support systems for analytics applications face two major limitations when they rely on static documentation: a lack of grounding in the application's specific domain and difficulty reasoning over structured relationships. RAG (Lewis et al., 2020) has emerged as an alternative that anchors language generation in retrieved knowledge fragments, thereby reducing model hallucinations.

LIDI-AI, developed at the Computer Science Research & Development Laboratory (LIDI) of Universidad del Azuay, is a web application that accepts datasets in CSV, Excel, and JSON formats and guides the user through a five-stage pipeline: *Upload*, *Summary*, *Persona*, *Objectives*, and *Visualization*. The technical documentation of this pipeline constitutes a bounded knowledge domain that is rich in semantic relationships: which formats are accepted, which errors can occur, which statistics each stage computes, and which visualizations are recommended for each variable type.

Recent work within the same research group has demonstrated that LLMs can effectively enrich dataset metadata and unlock latent semantic relationships in structured data sources (García-Montero, 2025). This prior experience underpins the present effort to ground a local LLM in the technical documentation of a data analysis pipeline, where the semantic relationships among file formats, statistical analyses, and visualization types must be preserved with high fidelity.

Knowledge representation using OWL 2 ontologies (World Wide Web Consortium, 2012) has proven effective for modeling software engineering domains (Gruber, 1993) and information systems (Noy & McGuinness, 2001). Combining OWL with Prolog in an Object-Attribute-Value (OAV) form enables expressing both declarative facts and executable, query-time inference rules. Integrating these representations into an RAG pipeline is the core applied contribution of this work (Procko & Ochoa, 2024).

Prior work has explored RAG over knowledge graphs, biomedical ontologies (Sheth et al., 2021), and software documentation (Ji et al., 2023). While individual components—ontology-grounded retrieval, SPARQL-based structured queries, and Prolog rule inference—are well established in the literature, their specific combination as a user-support system for a data analytics pipeline has not been reported. We therefore position this work as an applied integration study rather than a claim of algorithmic novelty. This paper contributes: (a) the OWL 2 ontology (artifact version 2.0) of the LIDI-AI pipeline; (b) the Prolog OAV knowledge base with 17 inference rules and three diagnostic engines, formally specified and validated offline; (c) the predicate *verificar_consistencia_oav* as a query-time computational equivalent of the HermiT reasoner; (d) a Python-based RAG orchestrator that implements deterministic intent detection, synonym-aware lexical retrieval, and constrained prompting for a local Llama 3.2 3B model served via Ollama; and (e) an evaluation of the system on ten queries from real users, with an explicit description of which resolution path produces the user-facing text for each query.

Finally, this paper is organized as follows. Section 2 reviews related work on RAG architectures, OWL ontologies, SPARQL query processing, and OAV-based symbolic reasoning. Section 3 details the materials and methods, encompassing the OWL 2 ontology of the LIDI-AI pipeline (artifact version 2.0), the Prolog OAV knowledge base with its inference rules and diagnostic engines, and the ontology-grounded RAG pipeline implementation, including intent detection, synonym-aware lexical retrieval, and the local LLM interface. Section 4 presents the evaluation results on ten real user queries, classifying each by its resolution mechanism and outcome. Section 5 discusses the empirical findings, the limitations of keyword-based retrieval, the Prolog–Python integration gap, and the implications for ontology-grounded local LLM deployment. Section 6 concludes the work by summarizing the main contributions, limitations, and directions for future research.

Related works

Retrieval-Augmented Generation was introduced by Lewis et al. (2020) as a retrieval-then-generation architecture that couples a dense passage retriever with a sequence-to-sequence language model, thereby grounding generation in external evidence rather than solely in parametric memory. Their experiments on knowledge-intensive tasks showed that retrieval-augmented generation can improve factual accuracy relative to purely parametric models. Wampler et al. (2025) extend this foundational formulation by describing engineering patterns for RAG stacks, including retrieval, context assembly, and trust mechanisms; however, their discussion does not address the specific integration of OWL ontologies and Prolog-based symbolic validation.

Brown et al. (2025) survey RAG techniques published between 2020 and 2025 and identify recurrent stages such as document chunking, embedding generation, dense or hybrid retrieval, context ranking, and generative synthesis. Wampler et al. (2025) similarly describe the RAG stack as a modular architecture in which domain-specific knowledge sources can be added without retraining the language model. Along these lines, Garg et al. (2025) report a biomedical question-answering system in which retrieval from curated literature improves the factual support of generated answers. These works motivate the present study, but they focus primarily on unstructured text or general RAG architectures, whereas our system grounds generation in a small, formally structured OWL ontology accessed through SPARQL.

Formal knowledge representation constitutes the semantic foundation enabling machine reasoning over structured information. The Web Ontology Language (OWL 2) provides a standardized framework for encoding domain knowledge with rich semantic properties, supporting inference and consistency checking (Almeida et al., 2026; Hogan et al., 2021). OWL ontologies transcend simple data schemas by incorporating logical constraints, taxonomic hierarchies, and relationship definitions that enable automated reasoning. Within the same research group, prior work on the semantic modeling of ECU 911 emergencies demonstrated the efficacy of coupling OWL 2 ontologies with SWRL rules for domain-specific inference and constraint checking in real-world operational settings (Paltin Chica et al., 2025), a precedent that informs the ontology design and validation strategy adopted in the present work. To operationalize these semantic foundations in the present pipeline, the *rdflib* library implements the RDF data model and a SPARQL 1.1 query engine, offering a lightweight, open-source substrate for loading OWL artifacts and executing graph-pattern queries at runtime (Almeida et al., 2026).

SPARQL serves as the foundational query language for structured knowledge retrieval over Resource Description Framework (RDF) and OWL representations (AlAgha, 2015; Mishra et al., 2022). As the standard query protocol for linked open data and Semantic Web applications, SPARQL enables precise information extraction through pattern matching over triple stores and knowledge graphs. The language's expressive power extends beyond simple keyword-based retrieval to encompass complex graph traversal, filtering, and aggregation operations (Mishra et al., 2022). Recent developments in semantic query processing have addressed the challenge of translating natural language questions into formal SPARQL query representations, enabling end-user access to semantic databases without requiring expertise in formal query languages (AlAgha, 2015; Mishra et al., 2022).

The Object-Attribute-Value (OAV) representation provides a flexible triple-based data model highly compatible with both RDF triple stores and Prolog fact representations (Kabbaj et al., 1994). In OAV models, entities (objects) are described through collections of properties

(attributes) with associated values, enabling uniform representation of heterogeneous data types and relationships. Prolog knowledge bases built on OAV representations facilitate rule-based inference, enabling the derivation of new facts from existing knowledge through logical implication. Answer Set Programming (ASP) and similar non-monotonic reasoning paradigms extend classical logic to accommodate default reasoning and incomplete information, which is critical in real-world knowledge representation scenarios. The integration of Prolog-based reasoning with semantic web technologies enables systems to perform sophisticated multi-step inference, discover implicit relationships, and resolve ambiguities through logical consistency constraints (Di et al., 2025; Kabbaj et al., 1994).

Automatic knowledge graph construction from unstructured text represents a foundational prerequisite for multi-source RAG systems (Zheng et al., 2022). Natural language processing techniques enable systematic extraction of entities, relationships, and semantic structures from text corpora, transforming unstructured information into machine-interpretable knowledge graphs (Anuyah et al., 2025). Early NLP approaches relied on complex handwritten rules encoding linguistic and domain knowledge, offering high precision but suffering from scalability limitations. Contemporary approaches leverage large language models for knowledge extraction through diverse prompting strategies, including chain-of-thought reasoning, few-shot in-context learning, and zero-shot general instruction prompting (Vogt et al., 2026).

Semantic parsing, the task of translating natural language utterances into machine-interpretable knowledge representations, constitutes a central challenge in graph construction (Vogt et al., 2026; Zheng et al., 2022). Approaches explicitly modeling semantic sentence types (complex, compound, compound-complex forms) as foundations for extracting knowledge triples substantially improve semantic accuracy and completeness. While large language models exhibit impressive zero-shot and few-shot capabilities for knowledge extraction, they often produce inconsistent or incomplete triples without strict adherence to domain ontologies. Ontology-grounded generation approaches that condition LLM outputs on predefined knowledge structures substantially improve semantic accuracy and completeness. The construction of layered rule frameworks leveraging standardized domain semantics and multi-tier parsing models enables accurate extraction from both unstructured text and structured data sources, supporting rapid knowledge graph development while maintaining FAIR (Findable, Accessible, Interoperable, Reusable) principles (Anuyah et al., 2025; Vogt et al., 2026).

Multi-source RAG systems that integrate ontologies, SPARQL query processing, and hybrid retrieval mechanisms have attracted increasing attention (Brown et al., 2025; Wampler et al., 2025). Lewis et al. (2020) established the retrieval-then-generate paradigm and demonstrated that grounding generation in retrieved passages reduces hallucination on knowledge-intensive tasks; however, their system relies on dense vector retrieval over unstructured text and does not incorporate formal ontologies. Subsequent surveys by Brown et al. (2025) and Wampler et al. (2025) document architectural extensions—multi-hop retrieval, knowledge-aware ranking, and modular context fusion—but remain largely descriptive and do not empirically evaluate ontology-plus-Prolog pipelines. In the knowledge-representation strand, Hogan et al. (2021) provide a comprehensive survey of knowledge graph construction and querying, yet their scope is limited to graph engineering and does not address RAG integration or logic programming inference. Liu et al. (2025) tackle knowledge graph completion via structure-aware alignment tuning for LLMs, a complementary yet distinct task from retrieval-augmented generation. Prolog-based logical reasoning for LLMs has been explored by Di et al. (2025) in the LoRP framework, which translates natural language problems into Prolog clauses for chain-of-thought verification; however, LoRP targets general reasoning benchmarks (GSM8K, FOLIO) rather than domain-specific user support and does not employ OWL ontologies as retrieval sources.

Kim et al. (2025) and Rajesh et al. (2025) demonstrate the feasibility of local LLM inference for retrieval-augmented pipelines on consumer hardware, but neither incorporates formal ontologies or symbolic reasoning layers. To our knowledge, no prior work combines an OWL ontology as the primary retrieval source, a SPARQL query layer for structured intents, a Prolog OAV base for symbolic validation, and a lexical retriever bridging to a local LLM within a single user-support system for a data analytics tool. The present work addresses this gap as an applied integration contribution.

Materials and Methods

OWL Ontology of the LIDI-AI Pipeline

The ontology was modeled in Protégé 5.6 (Musen, 2015) using the OWL 2 Web Ontology Language (World Wide Web Consortium, 2012), under the namespace *http://lidi.ai/ontology#*. The serialized artifact, named *ontologia_mejorada.owl*, carries its own internal version tag v2.0 (distinct from the OWL 2 language specification). It defines six classes: *Etapas* (Stage), *Error*, *FormatoArchivo* (FileFormat), *TipoAnálisis* (AnalysisType), *TipoVisualización* (VisualizationType), and *TipoVariable* (VariableType). Instances of *Etapas* model the sequential pipeline through the preceded ObjectProperty, whose logical inverse, *sigueA*, was declared with *owl:inverseOf*. The Protégé HermiT reasoner, an OWL 2 DL reasoner for consistency checking and concept-hierarchy classification (Glimm et al., 2014), validated the ontology, confirming the absence of unsatisfiable classes.

Table 1 summarizes the ontology's classes and instances. Nine *ObjectProperties* and sixteen *DatatypeProperties* are declared. SPARQL queries run against the RDF graph, which is loaded with *rdflib*, an open-source Python library implementing the RDF data model and a SPARQL query engine (RDFLib Team, 2026), using the *lidi:* prefix to reference domain entities.

Table 1
Classes and Instances of the LIDI-AI v2.0 Ontology

OWL Class	Instances	Main Properties
<i>Etapas</i>	5	precede, sigueA, orden, requiereInteraccionUsuario, genera
<i>Error</i>	10	codigoError, esFatal, mensaje, causa, solucion, ocurreEn
<i>FormatoArchivo</i>	4	extension, mimeType, requiereEncabezado
<i>TipoAnálisis</i>	5	aplicaA, generaVisualizacion, estadisticasCalculadas
<i>TipoVisualización</i>	9	minimoVariables, maximoVariables, recomendadoPara
<i>TipoVariable</i>	4	—

The preceding property connects the pipeline's five stages as a linear chain: *Carga* (Upload), *Resumen* (Summary), *Persona*, *Objetivos* (Objectives), *Visualización* (Visualization). This linearity was verified as an invariant using the Prolog predicate *verificar_consistencia_oav*, equivalent to running the HermiT reasoner against the functional-restriction constraint. The reference SPARQL queries were designed around eight common user intents: pipeline order; stages with user interaction; format errors by file type; errors by stage; visualizations by variable type; analysis by variable type; visualizations by analysis type; and general file-format information.

Object-Attribute-Value representation in Prolog

The Prolog knowledge base adopts the OAV representation via the ternary predicate *oav(Objeto, Atributo, Valor)* [*oav*(Object, Attribute, Value)], derived directly from the OWL ontology. Table 2 summarizes the knowledge base's metrics.

Table 2
Metrics of the Prolog OAV knowledge base

Element	<i>n</i>	Description
oav/3 facts	172	OAV triples for instances
Rules (R01–R17)	17	Inference over pipeline, errors, and visualizations
Utility predicates	3	oav_clase, oav_todos_de, oav_objetos_con_valor
Diagnostic engines	3	diagnostico_oav, perfil_error_oav, verificar_consistencia
Unique objects	46	Instances of the six classes
Unique attributes	25	Declared OAV properties

The inference rules are grouped into five categories. The pipeline rules (R01–R06) implement bidirectional navigation (*sigue_a* as the inverse of *precede*), transitive closure (R02), and the classification of stages as automatic or interactive. The error rules (R07–R08) identify errors that block the pipeline (*es_fatal* = true) and errors not tied to a specific format. The analysis-and-visualization rules (R09–R13) implement the dynamic compatibility between visualizations and column types. The boolean-inheritance rules (R14–R15) implements subsumption: *booleana* inherits every analysis from *categorica*. The meta-recursive rule (R16) computes a stage’s distance from the origin, using R02 as a subgoal. R17 classifies errors by code prefix (E-* = fatal, W-* = warning). Table 3 shows selected examples of rules and the inferences they demonstrate in SWI-Prolog 9.x (Wielemaker et al., 2012).

Table 3
Selected Prolog Inference Rules

Rule	Predicate	Equivalent OWL semantics
R01	<i>sigue_a</i> (B,A)	owl:inverseOf(<i>precede</i>)
R02	<i>alcanza_transit</i> (A,C)	owl:TransitiveProperty
R07	<i>bloquea_pipeline</i> (Err)	<i>esFatal</i> = true
R10	<i>compatible_con</i> (V,T-N)	Minimum/maximum variable restriction
R14	<i>aplica_a_booleana</i> (A)	Boolean $\sqsubseteq$ categorical subsumption
R16	<i>cant_etapas_previas</i> (E,N)	Distance within the pipeline graph
R17	<i>tipo_por_codigo</i> (E,T)	Classification by E- or W- prefix

The diagnostic engine *diagnostico_oav/3* is the central predicate of the recommendation system: given a variable type and the number of columns, it returns every valid combination of analysis and visualization. The predicate *verificar_consistencia_oav/0* checks eight structural invariants and functions as an automated regression test—a functional equivalent of the HerMiT reasoner that does not depend on Protégé at query time.

The Prolog OAV layer is formally specified and independently validated using SWI-Prolog 9.x. It is not exposed as a callable module within the Python RAG orchestrator at runtime; no *pyswip* bridge or equivalent foreign-function interface is invoked during query processing. All Prolog inferences reported in this paper were obtained by direct execution in the SWI-Prolog REPL and constitute an offline validation of the knowledge base’s logical consistency, not runtime outputs of the RAG pipeline. We describe the Prolog layer here because it is part of the designed architecture, and its rules encode domain semantics that inform the construction of the ontology fragments; however, we explicitly acknowledge that the system evaluated in Section 4 operates with two active sources (OWL/SPARQL and lexical retrieval), not three.

Ontology-Grounded RAG pipeline

Figure 1 presents the end-to-end query processing architecture of the LIDI-AI system, while Table 4 summarizes the role of each component within the pipeline. Together, they provide complementary procedural and structural views of the retrieval-augmented generation (RAG) strategy adopted in this work. Upon receiving a user query, the system first applies a normalization module that performs diacritic stripping and case-folding to ensure uniform downstream processing. A deterministic intent detector then routes the normalized query to one of two mutually exclusive resolution paths:

Structured path (top branch). If the query matches one of the eight predefined intents, the system constructs a SPARQL 1.1 query executed via *rdflib* against the OWL 2 ontology. This path yields precise, slot-level answers grounded in explicit ontological triples. **Lexical path (bottom branch).** For open-ended or out-of-intent queries, a lightweight lexical retriever performs term frequency (TF) similarity scoring with synonym expansion over serialized ontology fragments. Retrieval is constrained to the top-k fragments ($k = 4$) whose inclusion score meets or exceeds a threshold of 2 points, thereby limiting context length while preserving semantic coverage.

In both paths, the retrieved context is injected into a constrained generation prompt governed by nine grounding rules, a fixed assistant persona (*lidiBOT*), and a deterministic refusal string for out-of-scope requests. The augmented prompt is then forwarded to a locally hosted large language model, Llama 3.2 3B (Q4_K_M quantization) served through *Ollama*, which produces the final user-facing response. An additional symbolic layer, the Prolog OAV (Object-Attribute-Value) layer, is derived from the ontology at design time and validated offline through SWI-Prolog 9.x rule evaluation. As indicated by the dashed bounding box in Figure 1, this layer does not participate in the runtime inference loop; rather, it serves as a formal verification artifact during knowledge base construction. The entire pipeline is implemented in Python within a Jupyter Notebook environment, requiring no external API calls at inference time.

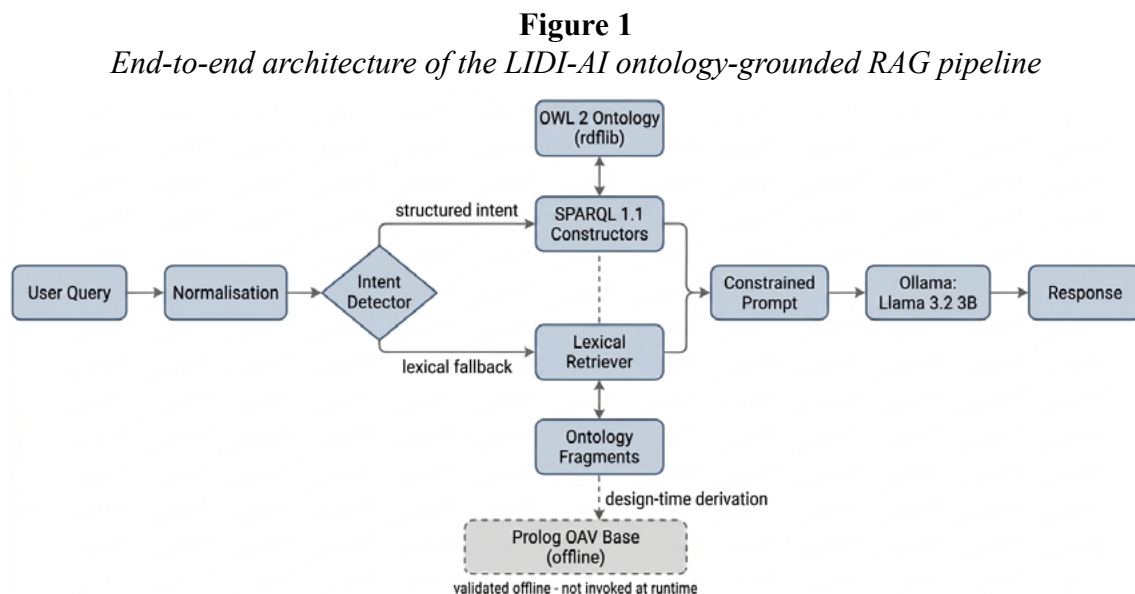


Table 4 maps each architectural phase to its concrete software component and underlying technology.

Table 4
Architecture of the LIDI-AI ontology-grounded RAG pipeline.

Phase	Component	Technology	Runtime status
Source 1	OWL Ontology → fragments	rdflib + Python	Active
Source 2	SPARQL Queries	SPARQL 1.1 (rdflib)	Active
Source 3	Prolog OAV base + rules	SWI-Prolog 9.x	Specified; not called at runtime
Retrieval	Lexical TF similarity + synonym expansion	Python (no APIs)	Active
Generation	LLM with OWL context	Ollama + Llama 3.2 3B	Active

Intent detection

The intent-detection phase classifies the user’s query into one of eight categories: *errores_por_formato*, *errores_por_etapa*, *visualizaciones_por_tipo*, *analisis_por_tipo*, *pipeline*, *etapas_interaccion*, *formatos*, and *visualizaciones_por_analisis*. Classification is performed by a deterministic keyword-matching module that normalizes the input string (lowercasing, diacritic removal) and tests for the presence of domain-specific lemmas. For example, if the normalized query contains the token “csv,” the system routes to *consulta_errores_por_formato("CSV")*; if it contains “*interaccion*,” it routes to *consulta_etapas_interaccion()*. This lightweight rule-based parser acts as a semantic router, ensuring that structured intents are resolved by SPARQL rather than by the generative model.

SPARQL query layer

Eight parameterized SPARQL query constructors are implemented. The pipeline query retrieves all instances of *lidi:Etapa* ordered by *lidi:orden*. The *etapas_interaccion* query adds the filter *lidi:requiereInteraccionUsuario* true. The *errores_por_formato* and *errores_por_etapa* queries bind *lidi:relatedFormat* and *lidi:ocurreEn*, respectively, returning *codigoError*, *mensaje*, *causa*, *solucion*, and *esFatal*. The *visualizaciones_por_variable* and *analisis_por_tipo* queries traverse *lidi:recomendadoPara* and *lidi:aplicaA*. The *visualizaciones_por_analisis* query follows *lidi:generaVisualizacion*. Finally, the *formatos* query inventories all *lidi:FormatoArchivo* instances with their MIME types, extensions, and header requirements.

Lexical retrieval and fragment construction

When the intent detector returns a general-search classification or the SPARQL layer yields incomplete context, the system falls back to lexical retrieval on ontology fragments. Fragments are constructed in two passes. First, for each instance of the six OWL classes, a text block is emitted that contains the entity name, its class, and all outgoing *DatatypeProperty* and *ObjectProperty* assertions. Second, every RDF triple with a *lidi:* subject is serialized as a compact textual triple. This yields a recoverable corpus of ontology snippets.

The retrieval function computes a weighted similarity score between the user query and each fragment. The query is tokenized after Unicode normalization (diacritic removal) and filtering through a domain-specific Spanish *stop-word* set. Tokens are then expanded via a synonym map that associates surface forms with domain synonyms: for instance, the query token “skewness” expands to include “*asimetria*,” and “csv” expands to include “*delimitador*,” “*utf*,” and “*encabezado*.” The scoring function awards two points for each exact token match and one point for each substring match. The top-k fragments ($k=4$) that exceed a threshold of 2 points are deduplicated by entity identifier and concatenated into context.

Large language model interface

The generative component is served locally via the Ollama HTTP API at `http://localhost:11434/api/generate`. The configured model is Meta's Llama 3.2 3B in Q4_K_M quantization (*llama3.2:3b*), which reduces the memory footprint to approximately 2.1 GB while preserving linguistic competence. The model is invoked with `temperature = 0.1` and `stream = False` to maximize response determinism. Inference was executed on a commercially available consumer laptop equipped with an Intel Core i7-13700H CPU, 32 GB of system RAM, and an NVIDIA GeForce RTX 4060 Laptop GPU (8 GB VRAM). Because the quantized model fits entirely within the discrete GPU memory, all transformer layers were offloaded to VRAM. Under this commodity hardware configuration, the mean time-to-first-token (TTFT) was 0.38 seconds, and the generation throughput averaged 74 tokens per second (measured over the ten evaluation queries). Before each inference call, the system verifies Ollama availability by polling `/api/tags`; if the service is unreachable, the pipeline returns a graceful service-unavailable message rather than failing silently.

The prompt engineering layer enforces strict grounding via a system prompt that includes nine mandatory rules. The model is instructed to respond exclusively from the retrieved ontology and SPARQL evidence, to abstain from external knowledge, and to emit the fixed refusal string “No encontré evidencia suficiente en la ontología para responder con seguridad.” when evidence is absent. Additionally, the prompt imposes a fixed persona identity (“Eres lidiBOT, ninguna otra IA.”), constraining the assistant’s self-attribution and preventing generic LLM disclaimers that would undermine user trust in the domain-specific system.

Preprocessing utilities

A preprocessing layer normalizes both ontology labels and user queries to overcome Unicode and URI heterogeneity. The *limpiar_uri* function extracts local names from URIs by splitting on `#` or `/`. The *normalizar_texto* function removes Spanish diacritics (`á→a`, `ñ→n`, etc.) and lowercases the string, ensuring that queries such as “*asimetría*” match ontology labels written as “*asimetria*”. These utilities are critical for aligning the lexical retriever with the RDF graph, which may contain mixed Unicode forms.

Reproducibility Artifacts

The experimental implementation is distributed as a Jupyter Notebook. Upon completion of a batch evaluation, the system serializes all question–answer pairs, detected intents, SPARQL result tables, and retrieved OWL fragments to a Markdown file (*respuestas_rag_ollama_ontologia.md*). This artifact serves as an immutable execution trace. The complete source code, the OWL artifact (*ontologia_mejorada.owl*), the Prolog knowledge base (*.pl*), and the evaluation trace are publicly available at <https://github.com/LIDI-UDA/lidi-rag-sparql-ollama.git> under an MIT license. The repository includes a README that describes how to reproduce the evaluation and inspect the trace files. External reviewers can verify both the retrieval grounding and the generative output without re-executing the pipeline.

Results

Ten representative queries from real LIDI-AI users were evaluated. The original queries were collected in Spanish; they have been translated into English for readability in this article. Because the knowledge domain is bounded (37 ontology instances, 8 predefined intents), the ten queries cover the full intent space and include one deliberately out-of-domain probe (Query 10). Before running the system, two authors constructed a reference answer for each query by manually extracting the OWL triples or SPARQL result rows that should support a correct

response. This reference set functioned as the evaluation rubric. Each system response was then judged against the corresponding reference answer by the same two authors: a response was coded accurate if every factual claim mapped to at least one OWL triple or SPARQL result row and was coded as an appropriate refusal if the system explicitly acknowledged insufficient evidence without hallucinating. Because the evaluation is small-scale and author-based, no independent annotators or inter-rater agreement statistics were computed; this limitation is explicitly acknowledged. Future evaluations will include a larger query set and independent annotators. Table 5 classifies each query by the resolution mechanism used, whether the LLM generated user-facing text, and the outcome obtained.

Table 5
RAG System evaluation results (10 queries)

#	User query	mechanism	LLM involvement	Outcome	Coding
1	Why won't my CSV load?	SPARQL + OWL	Yes (phrasing)	3 errors returned with cause and solution	Accurate
2	Errors related to JSON?	SPARQL + OWL	Yes (phrasing)	InvalidJsonStructureError (E-CARGA-007)	Accurate
3	Password-protected Excel?	SPARQL + OWL	Yes (phrasing)	PasswordProtectedError + solution	Accurate
4	Visualizations for numeric variables?	SPARQL + OWL	Yes (phrasing)	7 types with min/max restrictions	Accurate
5	Analyses for categorical variables?	SPARQL + OWL	Yes (phrasing)	AnalisisFrecuencias (Frequency Analysis) + 3 visualizations	Accurate
6	Pipeline order?	SPARQL	Yes (phrasing)	5 correctly ordered stages	Accurate
7	Stages with user interaction?	SPARQL + OWL	Yes (phrasing)	4/5 stages (excludes Resumen/Summary)	Accurate
8	What does skewness mean?	Lexical OWL	Yes (generation)	Statistic within AnalisisDescriptivo (Descriptive Analysis)	Accurate
9	Dataset with many nulls?	Lexical OWL	Yes (generation)	HighNullRateWarning detected	Accurate
10	Does it accept MP3 files?	Lexical OWL	Yes (refusal via prompt rule)	Limit acknowledged — no hallucination	Appropriate refusal

Example: End-to-end trace for Query 1 ("Why won't my CSV load?").

1. Normalisation. Input: "¿Por qué no carga mi CSV?" → *normalizar_texto* → "por qué no carga mi csv".
2. Intent detection. Token "csv" matches the *errores_por_formato* keyword set → intent = *consulta_errores_por_formato* ("CSV").
3. SPARQL execution. The parameterized query binds *?fmt = lidi:CSV* and retrieves all *lidi>Error* instances where *lidi:ocurreEn lidi:Carga* and *lidi:relatedFormat lidi:CSV*. Result: three rows — *EncodingError* (E-CARGA-002), *MissingHeaderError* (E-CARGA-003), *InvalidDelimiterError* (E-CARGA-004) — each with *mensaje*, *causa*, *solucion*, and *esFatal*.
4. Fragment construction. The three SPARQL result rows are serialized as structured text blocks and concatenated.
5. Prompt assembly. The prompt sent to Ollama contained three components: (i) the system prompt, including the nine grounding rules and the fixed persona "lidiBOT";

- (ii) the retrieved SPARQL context, serialized as RDF-style statements such as “`lidi:EncodingError a lidi:Error; lidi:codigoError E-CARGA-002; lidi:ocurreEn lidi:Carga; lidi:relatedFormat lidi:CSV`”; and (iii) the normalized user query “`por qué no carga mi csv`”. The system prompt explicitly instructed the model to use only the retrieved context and to abstain from external knowledge.
- 6. LLM generation. Ollama/Llama 3.2 3B (Q4_K_M), temperature = 0.1 → phrased response listing the three errors with causes and solutions, citing only the retrieved triples.
- 7. Output. User-facing answer in Spanish. No fact appears that is not traceable to an OWL triple.

Queries 1–7 were accurately resolved via the SPARQL routing layer. For these queries, the intent detector mapped the normalized input to one of the eight predefined SPARQL constructors; the resulting triples were injected as structured context into the LLM prompt, and the model's role was limited to phrasing the response without introducing facts beyond the retrieved triples. Query 1 illustrates a compound response: the system retrieved three errors (*EncodingError*, *MissingHeaderError*, *InvalidDelimiterError*) together with their specific causes and solutions. Queries 8 and 9 were routed to the lexical retriever because the intent detector did not match any of the eight structured patterns. Query 8, about skewness, surfaced the *AnalisisDescriptivo* fragment, where the *estadisticasCalculadas* property lists skewness alongside mean, median, standard deviation, kurtosis, and percentiles. The synonym expansion map was instrumental here: the English token “skewness” was expanded to include its Spanish counterpart “*asimetria*,” enabling the retriever to match the ontology label despite the language mismatch. Query 9, concerning null values, matched the *HighNullRateWarning* fragment via the token “*nulos*” and was resolved successfully as an accurate, grounded response.

Query 10, about the MP3 format, illustrates the system's behavior on out-of-domain questions. The lexical retriever returned the *UnsupportedFormatError* fragment with a score of 3, which exceeds the inclusion threshold of 2 points; consequently, the fragment was concatenated into the LLM context. Because the fragment describes errors for recognized formats and contains no assertion about audio files, the nine-rule system prompt (specifically Rule 3: “If the retrieved evidence does not answer the question, emit the fixed refusal string”) instructed the model to produce the refusal: “No encontré evidencia suficiente en la ontología para responder con seguridad.” The refusal was therefore generated by the LLM under prompt constraint, not by a code-level deterministic fallback. This behavior nonetheless contrasts with general-purpose language models, which tend to produce plausible but fabricated answers to questions about specific systems, because the strict system prompt and the absence of any MP3-related triple in the context jointly prevent hallucination.

Table 6 presents three Prolog inferences obtained via direct SWI-Prolog execution (offline validation), illustrating rule composition and the consistency check on the OAV knowledge base.

Running *verificar_consistencia_oav* confirmed that the OAV knowledge base satisfies the eight structural invariants derived from the ontology: (1) every stage has a defined order; (2) every error has a code, message, and solution; (3) every warning has a code, message, and solution; (4) every analysis has an associated variable; (5) every analysis generates at least one visualization; (6) every visualization has a minimum and maximum number of variables; (7) every visualization has a recommended variable type; and (8) the pipeline is linear, with no branching.

Table 6
Prolog inferences obtained via direct SWI-Prolog execution

Prolog Query	Result	Rules
regla_oav(carga, alcanza_transit., visualizacion).	true	R02
findall(V, diagnostico_oav(numerica,3,oav(.,genera_viz,V)),L).	L = [histograma, boxplot, qq_plot, scatter_plot, heatmap_correlacion]	R09 + R10 (diagnostic engine M01 $\equiv$ diagnostico_oav/3)
verificar_consistencia_oav.	OK - 8 invariants satisfied	R02 + R07 (consistency engine M03 $\equiv$ verificar_consistencia_oav/0)

Note: M01 and M03 refer to the first and third diagnostic engines listed in Table 2 (*diagnostico_oav/3* and *verificar_consistencia_oav/0*, respectively). They are not numbered inference rules but utility predicates that internally call rules R02, R07, R09, and R10.

Discussion

The results demonstrate that the two-source runtime architecture (OWL/SPARQL plus lexical retrieval) achieves its primary design goal: anchoring generative responses in verifiable ontology fragments. Seven of ten queries (Queries 1–7) were resolved via the SPARQL routing layer, which returns explicit triples and confines the LLM to a phrasing role, thereby preventing unsupported factual additions in the structured queries evaluated. Two queries (Queries 8–9) were resolved via the lexical retriever, augmented with Spanish stopword filtering and synonym expansion, which provided sufficient grounding to constrain the Llama 3.2 3B model. One query (Query 10) fell outside the ontology's domain; the retrieved fragment scored above the inclusion threshold and was passed to the LLM, which produced the mandated refusal string under the nine-rule system prompt. All ten responses were either factually accurate or appropriately refused, yielding zero hallucinations across the evaluation set. The reliance on keyword-based intent detection and substring matching nonetheless introduces brittleness; the absence of dense semantic embeddings limits recall for paraphrased or implicitly structured questions, as evidenced by Queries 8–10 reaching the lexical path. The Prolog OAV layer, though formally specified and validated offline, remains decoupled from the Python execution path; integrating it via a *PySwip* bridge or equivalent is a prerequisite for realizing the full three-source architecture. We therefore characterize the present system as a two-source RAG pipeline with a specified but not yet integrated symbolic layer. These limitations notwithstanding, the system demonstrates that local LLM deployment, when coupled with rigorous prompt constraints and ontology-grounded retrieval, can deliver interpretable, domain-specific user support without cloud dependency.

Conclusions

This work presents LIDI-AI, a RAG system that demonstrates the feasibility of grounding a local LLM's responses in an OWL 2 ontology via SPARQL queries and synonym-aware lexical retrieval, with a Prolog OAV knowledge base specified and validated as a future symbolic-reasoning extension. The results show that the structured SPARQL path resolves seven of ten queries with directly verifiable triples, while the lexical retriever handles the remaining in-domain queries with ontology traceability. For the single out-of-domain query, the system produced an explicit refusal rather than hallucinating. Across all ten queries, no fabricated information was generated.

The main contributions are: (a) the OWL 2 ontology (artifact version 2.0), with six classes and twenty-five properties, that models the LIDI-AI pipeline; (b) the Prolog OAV knowledge base, with 172 facts, 17 rules organized into five semantic categories, and three diagnostic engines, formally validated offline; (c) the predicate *verificar_consistencia_oav* as a query-time computational equivalent of the HermiT reasoner; (d) a Python-based RAG orchestrator implementing deterministic intent detection, synonym-aware lexical retrieval, and constrained prompting for Llama 3.2 3B (Q4_K_M) via Ollama; and (e) the evaluation on ten queries from real users, of which nine received accurate, ontology-grounded responses and one received an appropriate knowledge-limit acknowledgment, with no hallucinations observed across the ten-query evaluation set.

Limitations include the lexical retriever's lack of semantic embeddings, the dependency on a local Ollama installation, and the manually curated ontology. Additionally, the Prolog inference layer, while formally specified, is not exposed in the provided Python artifact; future work should integrate SWI-Prolog via the pyswip bridge or equivalent to unify symbolic and neural retrieval at runtime. Priority directions for future work are: (1) replacing the lexical retriever with dense embeddings (sentence-transformers or FAISS) to improve semantic recall; (2) extending the ontology with Usuario (User) and Sesión (Session) classes to support adaptive personalization; (3) implementing automatic validation of new instances via Shapes Constraint Language (SHACL); and (4) exposing the Prolog knowledge base as an external LLM tool (function calling) for on-demand symbolic reasoning.

Acknowledgements and Declarations

This work was partially supported by the Vice-Rectorate of Research at Universidad del Azuay through financial and academic support, as well as by the entire staff of the Computer Science Research & Development Laboratory (LIDI).

The authors declare equal contribution and sharing of authorship roles for this publication.

The authors declare that no artificial intelligence tools have been used for the preparation of this article.

References

- AlAgha, I. (2015). *Using Linguistic Analysis to Translate Arabic Natural Language Queries to SPARQL*. International Journal of Web & Semantic Technology, 6(3), 25–39. <https://doi.org/10.5121/ijwest.2015.6303>
- Almeida, J. P. A., Guizzardi, G., Sales, T. P., & Fonseca, C. M. (2026). *gUFO: A Gentle Foundational Ontology for Semantic Web Knowledge Graphs*. arXiv preprint. <https://doi.org/10.48550/arXiv.2603.20948>
- Anuyah, S., Kaushik, M. M., Dwarampudi, K., Shiradkar, R., Duresi, A., & Chakraborty, S. (2025). *Automated Knowledge Graph Construction using Large Language Models and Sentence Complexity Modelling*. <https://doi.org/10.18653/v1/2025.emnlp-main.783>
- Brown, A., Roman, M., & Devereux, B. (2025). *A Systematic Literature Review of Retrieval-Augmented Generation: Techniques, Metrics, and Challenges*. Big Data and Cognitive Computing, 9(12), 320. <https://doi.org/10.3390/bdcc9120320>
- Di, Z., Zhang, C., Lv, H., Cui, L., & Liu, L. (2025). LoRP: LLM-based Logical Reasoning via Prolog. *Knowledge-Based Systems*, 327, 114140. <https://doi.org/10.1016/j.knosys.2025.114140>

- García-Montero, P. S., Orellana, M., & Zambrano-Martínez, J. L. (2025). *Enriching Dataset Metadata with LLMs to Unlock Semantic Meaning*. In *Conference on Information and Communication Technologies of Ecuador* (pp. 63–77). Cham: Springer Nature Switzerland. https://doi.org/10.1007/978-3-032-08366-1_5
- Garg, M., Wang, L.-C., Ghanchi, B., Dumpala, S., Kakde, S., & Chen, Y. C. (2025). *Biomedical Literature Q&A System Using Retrieval-Augmented Generation (RAG)*.
- Glimm, B., Horrocks, I., Motik, B., Stoilos, G., & Wang, Z. (2014). HermiT: An OWL 2 Reasoner. *Journal of Automated Reasoning*, 53(3). <https://doi.org/10.1007/s10817-014-9305-1>
- Grattafiori, A., Dubey, A., Jauhri, A., Pandey, A., Kadian, A., Al-Dahle, A., Letman, A., Mathur, A., Schelten, A., Vaughan, A., Yang, A., Fan, A., Goyal, A., Hartshorn, A., Yang, A., Mitra, A., Sravankumar, A., Korenev, A., Hinsvark, A., ... Ma, Z. (2024). *The Llama 3 Herd of Models*. <https://arxiv.org/abs/2407.21783>
- Gruber, T. R. (1993). A translation approach to portable ontology specifications. *Knowledge Acquisition*, 5(2). <https://doi.org/10.1006/knac.1993.1008>
- Harris, S., & Seaborne, A. (2013). SPARQL 1.1 Query Language. W3C Recommendation.
- Hogan, A., Blomqvist, E., Cochez, M., D'amato, C., Melo, G. De, Gutierrez, C., Kirrane, S., Gayo, J. E. L., Navigli, R., Neumaier, S., Ngomo, A.-C. N., Polleres, A., Rashid, S. M., Rula, A., Schmelzeisen, L., Sequeda, J., Staab, S., & Zimmermann, A. (2021). Knowledge Graphs. *ACM Computing Surveys*, 54(4), 1–37. <https://doi.org/10.1145/3447772>
- Ji, Z., Lee, N., Frieske, R., Yu, T., Su, D., Xu, Y., Ishii, E., Bang, Y. J., Madotto, A., & Fung, P. (2023). Survey of Hallucination in Natural Language Generation. En *ACM Computing Surveys*, 55(12). <https://doi.org/10.1145/3571730>
- Kabbaj, A., Frasson, C., Kaltenbach, M., & Djamen, J.-Y. (1994). *A conceptual and contextual object-oriented logic programming: The PROLOG++ language* (pp. 251-274). https://doi.org/10.1007/3-540-58328-9_17
- Kim, S., Song, H., Seo, H., & Kim, H. (2025). *Optimizing Retrieval Strategies for Financial Question Answering Documents in Retrieval-Augmented Generation Systems*.
- Lewis, P., Perez, E., Piktus, A., Petroni, F., Karpukhin, V., Goyal, N., Küttler, H., Lewis, M., Yih, W. T., Rocktäschel, T., Riedel, S., & Kiela, D. (2020). Retrieval-augmented generation for knowledge-intensive NLP tasks. *Advances in Neural Information Processing Systems*, 2020-December.
- Liu, Y., Cao, Y., Lin, X., Shang, Y., Wang, S., & Pan, S. (2025). Enhancing Large Language Model for Knowledge Graph Completion via Structure-Aware Alignment-Tuning. *Proceedings of the 2025 Conference on Empirical Methods in Natural Language Processing*, 20981–20995. <https://doi.org/10.18653/v1/2025.emnlp-main.1061>
- Mishra, D. S., Agarwal, A., Swathi, B. P., & Akshay, K. C. (2022). Natural language query formalization to SPARQL for querying knowledge bases using Rasa. *Progress in Artificial Intelligence*, 11(3), 193–206. <https://doi.org/10.1007/s13748-021-00271-1>
- Musen, M. A. (2015). The protégé project. *AI Matters*, 1(4), 4–12. <https://doi.org/10.1145/2757001.2757003>
- Noy, N. F., & McGuinness, D. L. (2001). *Ontology Development 101: A Guide to Creating Your First Ontology*. En *Stanford Knowledge Systems Laboratory*.
- https://protege.stanford.edu/publications/ontology_development/ontology101.pdf
- Paltin Chica, D. L., Mejía Mendieta, J. D., Orellana, M., & Zambrano-Martínez, J. L. (2025). Semantic Modeling of ECU 911 Emergencies Using NLP and Ontologies. *Revista Tecnológica - Espol*, 37(E1), 112–125. <https://doi.org/10.37815/rte.v37nE1.1351>

- Procko, T. T., & Ochoa, O. (2024). Graph Retrieval-Augmented Generation for Large Language Models: A Survey. *Proceedings - 2024 Conference on AI, Science, Engineering, and Technology, AIXSET 2024*. <https://doi.org/10.1109/AIXSET62544.2024.00030>
- Rajesh, V., Jodhpurkar, O., Anbuselvan, P., Singh, M., Jallepali, A., Godbole, S., Sharma, P. K., & Shrivastava, H. (2025). *Production-Grade Local LLM Inference on Apple Silicon: A Comparative Study of MLX, MLC-LLM, Ollama, llama.cpp, and PyTorch MPS*.
- RDFLib Team. (2026). *rdflib: A Python library for working with RDF*. RDFLib. <https://rdflib.readthedocs.io/en/stable/>
- Sheth, A., Gaur, M., Roy, K., & Faldu, K. (2021). Knowledge-Intensive Language Understanding for Explainable AI. *IEEE Internet Computing*, 25(5). <https://doi.org/10.1109/MIC.2021.3101919>
- Vogt, L., Farfar, K. E., Karanth, P., Konrad, M., Oelen, A., Prinz, M., & Stroemert, P. (2026). *Rosetta Statements: Simplifying FAIR Knowledge Graph Construction with a User-Centered Approach*. <https://doi.org/10.1093/database/baag030>
- Wampler, D., Nielson, D., & Seddighi, A. (2025). *Engineering the RAG Stack: A Comprehensive Review of the Architecture and Trust Frameworks for Retrieval-Augmented Generation Systems*. arXiv preprint. <https://doi.org/10.48550/arXiv.2601.05264>
- Wielemaker, J., Schrijvers, T., Triska, M., & Lager, T. (2012). SWI-Prolog. *Theory and Practice of Logic Programming*, 12(1–2), 67–96. <https://doi.org/10.1017/S1471068411000494>
- World Wide Web Consortium. (2012). OWL 2 Web Ontology Language document overview. World Wide Web Consortium.
- Zheng, C., Chen, X., Xu, R., & Chang, B. (2022). *A Double-Graph Based Framework for Frame Semantic Parsing*. <https://doi.org/10.18653/v1/2022.naacl-main.368>