Integración de grafos de conocimiento educativos con modelos de lenguaje masivos mediante arquitecturas RAG para evaluación personalizada en la enseñanza de programación

Web Semántica y Enlazado de Datos (cód. 31108018)
Máster en Investigación en IA — UNED
Adrián Bueno Junquero — Dir.: José Luis Fernández Vindel
Curso 2025/2026

Índice

1 Declaración sobre el uso de herramientas de inteligencia artificial 4
1.1 Reutilización y transparencia 4
1.2 Igualdad de género 4
2 Resumen 5
2.1 Abstract 5
2.2 Palabras clave 6
3 Introducción, objetivos e hipótesis 7
3.1 Preguntas de investigación 7
3.2 Hipótesis 8
3.3 Objetivos 8
3.4 Contraste de la hipótesis y competencias 9
4 Marco teórico: Web Semántica y grafos de conocimiento 11
4.1 RDF 1.1: el modelo de datos en grafo 11
4.2 RDF Schema: el primer escalón de la semántica 11
4.3 OWL 2: ontologías y razonamiento 12
4.4 SPARQL 1.1: la consulta de grafos 12
4.5 SHACL: validación de formas 13
4.6 Linked Data, SKOS y la nube de datos abiertos 13
4.7 Grafos de conocimiento: RDF frente a grafos de propiedades 14
4.8 Construcción, refinamiento y el grafo como base operativa 14
5 Estado del arte 16
5.1 Grafos de conocimiento en educación 16
5.2 Generación aumentada por recuperación y su variante sobre grafos 17
5.3 Generación automática de retroalimentación en programación 17
5.4 El hueco en la intersección y posicionamiento del trabajo 17
6 Metodología y reproducibilidad 20
6.1 Un diseño mixto: ingeniería y evaluación 20
6.2 Fases del trabajo 20
6.3 Reproducibilidad, entorno y herramientas 21
7 1. Artefactos del grafo: inferir, validar, consultar, exportar 22
8 2. Datos de la Grafoteca (JSON Cytoscape) 22
9 3. Diagramas RDF/TBox (si “dot” esta instalado) 22
10 4. Construir la memoria (PDF y DOCX) 22
11 Modelado ontológico del EKG 23
11.1 Dos espacios de nombres: separar el esquema de los datos 23
11.2 La taxonomía de clases 23
11.3 Propiedades de objeto: la malla de relaciones 24
11.4 Propiedades de datos: literales tipados y etiquetas multilingües 24
11.5 Reutilización de vocabularios: SKOS, Dublin Core, FOAF y schema.org 25
11.6 Dos redes superpuestas: `skos:broader` no es `rdfs:subClassOf` 25
11.7 Relaciones N-arias y reificación: cuando una propiedad binaria no basta 26
11.8 Recapitulación de las decisiones de diseño 26
12 Inferencia y razonamiento 27
12.1 De la colección de tripletas a la base de conocimiento 27
12.2 Dos motores, un mismo perfil: owlrl y GraphDB 27
12.3 El contraste NONE frente a RDFS/OWL-RL 27
12.4 Tres figuras de inferencia: transitividad, inversa y simetría 28
12.5 Reproducibilidad del perfil OWL 2 RL entre owlrl y GraphDB 29
13 Validación de la integridad con SHACL 30

13.1 Tres tareas complementarias sobre el grafo 30
13.2 Diseño del grafo de formas 30
13.3 Restricciones reforzadas 31
13.4 Validación con pyshacl e inferencia RDFS 31
13.5 Reflexión sobre el papel de SHACL en el flujo de trabajo 32
14 Consultas SPARQL y explotación 33
14.1 SELECT y CONSTRUCT 33
14.2 Errores con procedencia, la reificación 33
14.3 Prerrequisitos transitivos y CONSTRUCT que aplana el cierre 34
14.4 La consulta-caso, del envío al subgrafo de evidencia 34
14.5 SPARQL como puente entre representación y generación 35
14.6 Complemento metodológico: grafos multinaturaleza y explotación SPARQL de la ontología pedagógica (reconstrucción v2/v4 — Linaje B) 35
15 Enlazado de datos y reutilización de vocabularios 37
15.1 El principio de reutilizar antes de acuñar 37
15.2 La malla conceptual SKOS 38
15.3 Enlazado a Wikidata con `skos:exactMatch` 38
15.4 Verificar cada QID contra la fuente 39
15.5 Consultas federadas 39
15.6 VoID y datos FAIR 39
15.7 Balance de la fase 40
16 Integración del grafo con modelos de lenguaje 41
16.1 El módulo de recuperación 41
16.2 El afinado del Sistema B mediante QLoRA 42
16.3 La evaluación A/B/C/D 42
17 Discusión, limitaciones y despliegue 45
17.1 El grafo como estructura operativa 45
17.2 La complementariedad B/C y la lógica del Sistema D 45
17.3 Tensiones de diseño y limitaciones 46
17.4 Integración, despliegue y trabajo futuro 47
18 Conclusiones 48
18.1 Lo que el grafo demuestra por sí mismo 48
18.2 La garantía estructural 48
18.3 Integración con LLMs, leída sin complacencia 48
18.4 Recapitulación 49
19 Referencias 50
19.1 Nota sobre el tratamiento de las fuentes 51

1 Declaración sobre el uso de herramientas de inteligencia artificial

En la elaboración de esta memoria he utilizado herramientas de inteligencia artificial generativa como apoyo a la redacción y a la organización del texto, partiendo de borradores que he revisado, reescrito con mi propia voz y verificado en su totalidad.

El diseño del trabajo, la construcción y la validación del grafo de conocimiento, el desarrollo del software, la ejecución de las consultas y los experimentos, y todas las cifras, tablas y conclusiones que aquí se presentan son obra propia, reales y reproducibles a partir del código y de los artefactos del proyecto.

Asumo la plena responsabilidad sobre el contenido de este documento y declaro este uso de herramientas de inteligencia artificial conforme a la normativa de la Universidad Nacional de Educación a Distancia (UNED) sobre integridad académica.

1.1 Reutilización y transparencia

Esta memoria comparte sustrato (grafo de conocimiento, guiones, figuras y resultados) con el Trabajo de Fin de Máster del mismo autor (cód. 31108022). Dicha reutilización cuenta con la autorización del director y se declara aquí con transparencia, conforme a las buenas prácticas sobre reciclaje de texto del propio autor.

1.2 Igualdad de género

En coherencia con el valor de la igualdad de género, las denominaciones empleadas en esta memoria en género masculino, cuando no se hayan sustituido por términos genéricos, se entenderán hechas indistintamente en género femenino o masculino.

Adrián Bueno Junquero · Curso 2025/2026

2 Resumen

Esta memoria documenta el diseño, construcción y explotación de un grafo de conocimiento educativo (Educational Knowledge Graph, EKG) para la programación introductoria en Python. El grafo se concibe como sustrato semántico para la retroalimentación formativa automática. El feedback es una de las palancas más eficaces del aprendizaje (Hattie y Timperley, 2007), pero en programación sigue siendo costoso de producir con calidad y trazabilidad (Keuning et al., 2019). He querido explorar si una representación formal del dominio, articulada con tecnologías de la Web Semántica, aporta aquello que los modelos de lenguaje, por sí solos, no garantizan —la estructura, la procedencia y el razonamiento.

El núcleo es un grafo canónico que modela 157 conceptos mediante RDF/OWL, con una TBox (`pyedu:`) de 20 clases, 21 propiedades de objeto y 7 de datos, y una ABox (`pyr:`) en Turtle bajo el perfil OWL 2 RL. El grafo afirma 1772 enunciados que, tras el cierre deductivo, ascienden a 4786 triples (+3014 inferidos). El efecto se hace tangible al recuperar individuos de tipo `pyedu:Concepto`, que devuelve 0 sin inferencia y 157 con ella. Esos 157 se deducen por `rdfs:subClassOf`, mientras que las 30 entidades de Wikidata enlazadas con `skos:exactMatch` no se infieren como `pyedu:Concepto`, pues `skos:exactMatch` vincula sin propagar el tipo, a diferencia de `owl:sameAs`. He reutilizado vocabularios consolidados (SKOS, Dublin Core, FOAF, schema.org), enlazado el grafo a la Web de Datos con 30 vínculos `skos:exactMatch` verificados a Wikidata y modelado una red conceptual con 19 relaciones `skos:broader`, 19 `skos:narrower` y 16 `skos:related`, distinta de la jerarquía de clases. Preferí `skos:exactMatch` sobre `owl:sameAs` para no imponer la fusión lógica de individuos propia de OWL 2 RL, en un uso prudente de los datos enlazados.

La expresividad se refuerza con relaciones N-arias reificadas[1] y con reificación `rdf:Statement`, que registra mediante `dcterms:source` la procedencia bibliográfica de los 16 errores conceptuales. El grafo organiza además 16 temas. La calidad estructural se garantiza con SHACL. Sobre 10 `sh:NodeShape` el grafo es conforme (0 violaciones), y un fichero con violaciones deliberadas detecta exactamente las 6 previstas, control negativo que valida el propio validador. La explotación se demuestra con SPARQL. La consulta 02 sobre `?x a pyedu:Concepto` pasa de 0 a 157 con inferencia, la 03 recupera 5 prerrequisitos transitivos, la 04 devuelve 1 ruta de aprendizaje, el `CONSTRUCT` 06 expande de 340 a 356 triples y la consulta federada 08 retorna 20 filas contra Wikidata.

Sobre esta base he integrado el grafo con modelos de lenguaje mediante GraphRAG (Lewis et al., 2020; Edge et al., 2024) y he comparado cuatro sistemas —el LLM base (A), el LLM afinado con QLoRA (B), la base con GraphRAG (C) y el híbrido (D). El *fine tuning* de B (QLoRA sobre Qwen2.5-Coder-7B) mejora la pérdida en validación retenida de 1,051 a 1,028, con exactitud media por token de 0,842. En un banco de 50 casos retenidos, B mejora la clasificación del error mientras C mejora la identificación del concepto y la trazabilidad, ambos por encima de A. Esa complementariedad motiva el híbrido D, el sistema con mejor desempeño en distribución. Gana o empata en las siete dimensiones y alcanza 0,76 en categoría y 0,54 en concepto. Ahora bien, al probar la generalización sobre código real (corpus Dublin) se observa una inversión fuera de distribución, pues A obtiene 0,802, C 0,733, B 0,433 y D 0,323; el sistema base resiste así mejor lo no visto. Las cautelas pesan. Ningún objetivo del anteproyecto se alcanza, el panel de jueces automáticos arroja un acuerdo apenas leve (Fleiss 0,213) y la evaluación, con n=50 sobre casos generados por plantilla y sin panel humano, obliga a leer los resultados con modestia. El panel de evaluadores humanos queda como trabajo futuro.

2.1 Abstract

Feedback is one of the most effective levers on learning (Hattie and Timperley, 2007), yet in programming education it remains costly to produce with quality and traceability (Keuning et al., 2019). Language models generate feedback at scale, but on their own they do not guarantee the structure, provenance and reasoning that a formal representation of the domain would supply. Here I investigate

[1] mediante las clases `EvaluacionDeConcepto` y `EvaluacionActividad`, con 10 instancias

whether an Educational Knowledge Graph (EKG) for introductory Python programming, used as a semantic substrate for automated formative feedback, can close that gap. The canonical graph models 157 concepts in RDF/OWL, with a TBox of 20 classes, 21 object properties and 7 datatype properties in Turtle under OWL 2 RL. It asserts 1772 statements that expand to 4786 triples after closure (+3014 inferred), and reasoning is shown by a query returning 0 individuals without inference and 157 with it. The graph reuses SKOS, Dublin Core, FOAF and schema.org, links to the Web of Data through 30 verified `skos:exactMatch` mappings to Wikidata (preferred over `owl:sameAs` to avoid the logical merge of individuals entailed by OWL 2 RL) and is validated with SHACL (0 violations on the conformant graph and exactly 6 on a deliberately faulty one). I then compared four systems via GraphRAG (Lewis et al., 2020; Edge et al., 2024), namely a base model (A), a QLoRA fine-tuned model (B), a GraphRAG--augmented base model (C) and a hybrid (D). On 50 held-out cases B improves error classification while C improves concept identification and traceability, both surpassing A, and the hybrid D performs best, winning or tying across all seven dimensions (0.76 in category). Nonetheless no project target is met, and the evaluation still lacks a human panel, left as future work. I report these findings with explicit caution given the small, template-generated, held-out set, and I argue that formal knowledge graphs are a promising but not yet sufficient scaffold for trustworthy automated feedback.

2.2 Palabras clave

Grafo de conocimiento educativo; Web Semántica; RDF/OWL; SHACL; razonamiento OWL 2 RL; datos enlazados; GraphRAG; QLoRA; retroalimentación formativa automática; enseñanza de la programación en Python.

3 Introducción, objetivos e hipótesis

La docencia de la programación arrastra una tensión reconocible para quien ha corregido prácticas, y de ella nace este trabajo. La retroalimentación formativa, entendida como conversación orientada a reducir la distancia entre lo hecho y lo esperado, figura entre los factores de mayor impacto sobre el aprendizaje. Hattie y Timperley (2007) mostraron que el feedback más valioso opera sobre el procesamiento de la tarea y la autorregulación, no sobre la mera señalización de aciertos superficiales. Ese feedback rico es también el más costoso, porque exige comprender la intención del estudiante, identificar el concepto que falla y ajustar la intervención al momento de aprendizaje. La Figura 1 sintetiza esa tensión entre calidad y escala y anticipa la vía propuesta.

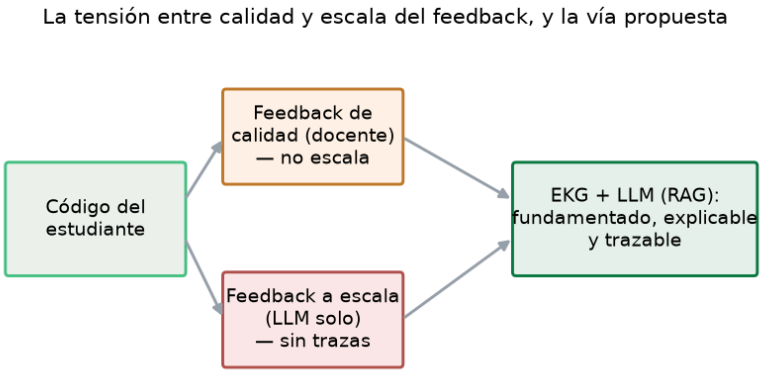

Figura 1. La tensión entre la calidad y la escala del feedback, y la vía propuesta de anclar la generación del modelo de lenguaje en un grafo de conocimiento.

Frente a esa calidad se alza la escalabilidad, pues la atención que un docente brinda a un grupo reducido se vuelve imposible ante cientos de estudiantes. Me inscribo en la tradición que busca aflojar esa tensión, aunque desde un ángulo poco explorado. Sitúo el modelado semántico explícito del dominio como condición previa para que la automatización sea, además de escalable, fiable y trazable. Keuning, Jeuring y Heeren (2019) concluyeron que la mayoría de las herramientas se concentran en feedback de bajo nivel. Dicen al estudiante *que* algo falla, y a veces *dónde*, pero rara vez *por qué* en términos conceptuales, y casi nunca *cómo* avanzar. La carencia no es accidental. Producir feedback conceptual exige representar qué conceptos componen el dominio, cómo se relacionan, qué errores típicos se asocian a cada uno y qué evidencia en el código delata un concepto mal comprendido. Construir esa representación de forma compartible y verificable resulta trabajoso.

La irrupción de los modelos de lenguaje de gran tamaño ha cambiado los términos del problema, y conviene examinarla antes de dejarse arrastrar por el entusiasmo. Un modelo contemporáneo redacta una explicación razonada que aparenta comprender la intención del programa, pero esa apariencia es, en parte, solo apariencia. Dos limitaciones pesan especialmente en un contexto educativo. La primera es la alucinación. El modelo puede afirmar con aplomo que un fragmento contiene un error inexistente o inventar la causa de un fallo, lo que consolida concepciones erróneas en quien aún construye su modelo mental. La segunda es la opacidad, ya que no ofrece justificación verificable ni podemos auditar por qué clasifica un error de una manera y no de otra. El feedback no solo debe ser correcto, sino poder mostrar por qué lo es. El trabajo explora también una vía intermedia con límites que importa reconocer. El *fine tuning* mejora la clasificación del tipo de error, pero no resuelve por sí solo la trazabilidad ni elimina las respuestas inventadas. Desplaza el conocimiento al interior de unos pesos opacos en lugar de exponerlo de forma inspeccionable.

3.1 Preguntas de investigación

De ese diagnóstico extraigo la pregunta que vertebra el trabajo, la misma que articula el artículo derivado de esta investigación y que enuncio en cursiva para destacar su carácter rector.

¿Cuándo, y bajo qué tipo de evaluación, aumentar un modelo de lenguaje masivo (LLM) con un grafo de conocimiento educativo o con fine tuning eficiente en parámetros mejora realmente la retroalimentación formativa en la enseñanza de la programación?

La descompongo en cuatro subpreguntas que ordenan el desarrollo de la memoria y se corresponden, respectivamente, con las preguntas PI1 a PI4 etiquetadas en el artículo.

- **SQ1.** ¿Es posible construir un grafo de conocimiento educativo validado con SHACL (cero violaciones) y cerrado bajo razonamiento OWL 2 RL, que cumpla objetivos estructurales explícitos (≥150 conceptos) y enlace de forma verificable con una base de conocimiento abierta como Wikidata?
- **SQ2.** ¿Mejoran el anclaje en el grafo (GraphRAG), el *fine tuning* con QLoRA y su combinación híbrida la calidad objetiva de la retroalimentación frente a un LLM base, y se transfiere esa ventaja obtenida en distribución al código real de estudiantes (fuera de distribución)?
- **SQ3.** ¿Constituye un juez LLM un instrumento fiable y válido para puntuar la calidad de esta retroalimentación cuando se contrasta su validez de criterio con la de un panel humano?
- **SQ4.** ¿Es un panel humano promediado un instrumento fiable y discriminante de la calidad de la retroalimentación, incluso cuando la concordancia entre evaluadores individuales es baja?

3.2 Hipótesis

La hipótesis nace de confrontar esas dos historias. Mi conjetura es que las dos debilidades se compensan. Si dotamos al modelo de una representación explícita, formal y consultable del dominio, conservamos su fluidez. Al tiempo, restringimos sus afirmaciones a aquello que el dominio respalda y le exigimos justificar cada juicio con una referencia rastreable. Esa representación adopta aquí la forma de un grafo de conocimiento educativo (EKG) que, en el sentido que sistematizan Hogan et al. (2021), apoya entidades y relaciones en una ontología con semántica formal y admite inferir hechos no afirmados. El grafo codifica los conceptos de programación, su jerarquía, los errores típicos y la procedencia bibliográfica de cada afirmación (Abu-Salih y Alotaibi, 2024; Qu et al., 2024). El mecanismo que conecta grafo y modelo es la generación aumentada por recuperación (RAG, Lewis et al., 2020), en la variante GraphRAG (Edge et al., 2024) que recupera un subgrafo coherente en lugar de pasajes sueltos. La hipótesis, formulada con cautela, es que un modelo anclado en un subgrafo producirá retroalimentación más correcta conceptualmente y, sobre todo, más trazable, porque cada afirmación podrá remitirse a un nodo del grafo y, a través de él, a su procedencia.

A mi juicio, el modelado semántico no es un adorno al servicio del modelo de lenguaje, sino el núcleo del que depende todo lo demás. Representar el conocimiento en RDF con esquema en RDFS y un perfil de OWL 2 RL permite que el grafo infiera. Una consulta por las instancias de la clase de concepto no devuelve nada sobre el grafo afirmado y devuelve 157 una vez aplicada la inferencia, al deducirse los conceptos propios por la jerarquía de subclases; las entidades enlazadas con `skos:exactMatch` no engrosan ese recuento, pues esa relación no propaga el tipo. La validación con SHACL, la consulta con SPARQL, la reutilización de SKOS, Dublin Core, FOAF y schema.org, el enlazado con Wikidata y la reificación de la procedencia de cada error completan ese rigor e inscriben la trazabilidad en la propia estructura. El modelo de lenguaje aprovecha ese rigor, no lo produce.

3.3 Objetivos

El objetivo general es diseñar, construir y evaluar un EKG sobre los fundamentos de la programación en Python, representado en RDF/OWL conforme a los estándares de la Web Semántica, validado mediante SHACL y explotado con SPARQL y con su integración con modelos de lenguaje en un esquema RAG, con el fin de producir retroalimentación formativa automática sobre errores conceptuales. Sus cuatro capas, representación, validación, explotación e integración, articulan la memoria, y la relación entre los cinco objetivos y esas capas se resume en Figura 2. No pretendo una ontología exhaustiva ni un sistema de producción, sino un artefacto suficientemente rico para sostener experimentos y suficientemente

acotado para ser verificable a mano. El grafo canónico reúne 157 conceptos propios, organizados en 16 temas y con 16 misconceptions modeladas, sobre 20 clases OWL, 21 propiedades de objeto y 7 propiedades de datos. Crece de 1772 enunciados afirmados a 4786 tras el cierre deductivo bajo OWL 2 RL. La generalización a otros dominios queda como línea legítima, aunque prefiero no sobreafirmarla.

Los cinco objetivos y las capas de la solución

Objetivo	Capa	Evidencia / cifra real
Construir el EKG en RDF	Representación	157 conceptos · 30 skos:exactMatch · SKOS 19/19/16
Validar con SHACL	Validación	10 sh:NodeShape · 0 violaciones (6 en el control)
Explotar con SPARQL	Explotación	consulta «?x a Concepto»: salto 0 → 157 conceptos
Integrar con modelos de lenguaje (RAG)	Integración	sistemas A/B/C/D · held-out 1,051→1,028 · token 0,842
Evaluar cuatro sistemas	Evaluación	D ≥ {B,C} > A (acierto de categoría 0,76 vs 0,26)

elaboración propia

Figura 2. Los cinco objetivos y las capas de la solución.

Los objetivos específicos descienden de esa meta. El primero es construir el EKG en RDF dialogando con la nube de datos abiertos. Para ello empleo 30 relaciones `skos:exactMatch` verificadas contra Wikidata y 19 `skos:broader` (con sus 19 `skos:narrower` recíprocas y 16 `skos:related`). Estas relaciones separan la subsunción lógica, sobre la que razona el motor, de la asociación pedagógica más laxa; usar `skos:exactMatch` en lugar de `owl:sameAs` evita que el razonador fusione el concepto propio con el individuo externo. Incluyo además relaciones N-arias por reificación y procedencia con `rdf:Statement` y `dcterms:source` en dos espacios de nombres, `pyedu:` para el esquema y `pyr:` para las instancias. El segundo es validar con SHACL mediante 10 `sh:NodeShape`, con 0 violaciones en el grafo bueno y 6 en el control negativo. El tercero es explotar con SPARQL y poner de manifiesto la inferencia, pues el salto de 0 a 157 conceptos al cerrar bajo RDFS y OWL 2 RL prueba que el conocimiento útil emerge del razonamiento sobre el esquema. El cuarto es integrar con modelos de lenguaje mediante RAG, que inyecta el subgrafo relevante al error para que el modelo no tenga que «recordar» la taxonomía sino recibirla explícita y trazable. El quinto es evaluar comparativamente cuatro sistemas — el A, un LLM base sin grafo; el B, afinado con QLoRA sobre Qwen2.5-Coder-7B; el C, base aumentado con GraphRAG; y el D, híbrido. El *fine tuning* del B exigió cautela. El run inicial sobreajustaba, y el regularizado redujo la pérdida en held-out de 1,051 a 1,028 con una precisión de token de 0,842. Como el conjunto se generó por plantillas, la evaluación se restringe a esqueletos held-out no vistos, para prevenir fugas, sobre una muestra pequeña.

3.4 Contraste de la hipótesis y competencias

La hipótesis central puede enunciarse así. Dotar a un modelo de un grafo de conocimiento educativo explícito, validado y razonado mejora la calidad de la retroalimentación formativa, especialmente en las dimensiones conceptual y de trazabilidad. Es deliberadamente refutable, y los datos la matizan más de lo que esperaba. La descompongo en tres subhipótesis sobre n=50 casos held-out juzgados por un modelo juez. Sobre precisión de categoría, el *fine tuning* y el híbrido destacan, con D en 0,76, mientras el GraphRAG sobre el base apenas aventaja a la línea de partida. Sobre acierto de concepto, los sistemas con grafo o afinado superan al base y el híbrido encabeza con D en 0,54. Sobre trazabilidad, la estructura externa hace más rastreable la explicación y el híbrido la lleva un paso más allá. El cuadro revela una complementariedad que no había anticipado. B mejora la clasificación, C mejora el concepto y la trazabilidad, y el Sistema D recoge ambas fortalezas a la vez y gana o empata en las siete dimensiones evaluadas. Eso, y no la simple superioridad del grafo, motivó el híbrido. Los límites son claros. Las conclusiones se sostienen sobre un único juez automático y un conjunto generado por plantillas, sin panel humano de validación, que queda como trabajo futuro. No deben leerse como un éxito rotundo. Ninguno de los objetivos cuantitativos del anteproyecto se alcanza y el mejor sistema se queda en 0,76 de acierto de categoría, lejos de los umbrales fijados. No proclamo el cumplimiento de las metas numéricas; aporto, eso sí, una indicación robusta de que el conocimiento internalizado y el externalizado atienden a necesidades distintas y de que combinarlos es la vía más prometedora.

Estos objetivos recorren las competencias de la asignatura, tal como las fija su guía de estudio (UNED, 2025), que enmarca este trabajo en el contexto del curso *Web Semántica y Enlazado de Datos* (cód.

31108018) del Máster Universitario en Investigación en Inteligencia Artificial. El grafo en RDF/RDFS/OWL 2 RL y su serialización en Turtle ejercitan los lenguajes de la Web Semántica; el enlazado con Wikidata y la reutilización de SKOS, Dublin Core, FOAF y schema.org materializan la publicación e interconexión de datos enlazados; la validación con SHACL cubre la calidad; la explotación con SPARQL y el salto inferencial de 0 a 157 atienden a interrogar y razonar sobre grafos; y la integración con modelos de lenguaje sitúa el trabajo en el cruce entre la Web Semántica y la investigación actual en inteligencia artificial. La publicación en MyST Markdown cierra el ciclo al convertir el conocimiento en un recurso web reutilizable.

4 Marco teórico: Web Semántica y grafos de conocimiento

La Web Semántica no es una tecnología única, sino un programa de ingeniería sostenido durante más de dos décadas por el W3C, cuya ambición era extender la Web de documentos hacia una Web de datos en la que las máquinas tratasen el significado de forma explícita y procesable. Desarrollo aquí los estándares que sostienen el resto de la memoria, porque el Grafo de Conocimiento Educativo (EKG) que he construido no se entiende sin ellos; me concentro en los que utilizo de forma efectiva (RDF, RDFS, OWL 2, SPARQL, SHACL y SKOS). La Figura 3 resume esa pila y las cifras que cada capa aporta al grafo.

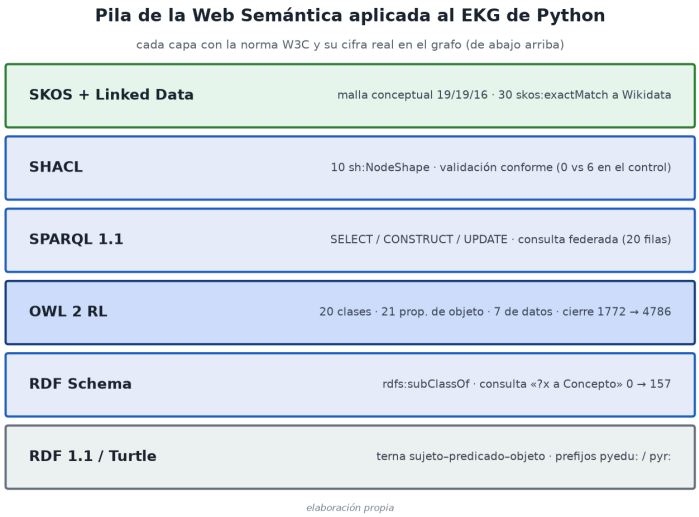

Figura 3. Pila de la Web Semántica aplicada al EKG (cifras por capa).

4.1 RDF 1.1: el modelo de datos en grafo

En la base está el Resource Description Framework, en su versión 1.1 (W3C, 2014a), cuyo modelo casi minimalista debe su flexibilidad a esa parquedad. La unidad elemental es la terna (sujeto, predicado y objeto), y su acumulación da lugar a un grafo dirigido y etiquetado adecuado para representar el conocimiento heterogéneo, incompleto y evolutivo que pretendo capturar. La identidad de los recursos descansa sobre los IRI, lo que garantiza que dos conjuntos de datos independientes se refieran al mismo objeto sin coordinación previa; los abrevian los espacios de nombres, y he reservado `pyedu:` para los términos del esquema (la TBox) y `pyr:` para las instancias (la ABox). Los objetos son recursos o literales (estos siempre hojas del grafo), más los nodos en blanco, que complican la fusión por su identidad local. Por esa razón he preferido acuñar IRI explícitos incluso para entidades intermedias, cautela que reaparece en la reificación de relaciones n-arias. RDF es independiente de la sintaxis, así que he optado por Turtle como formato principal por su legibilidad y por JSON-LD para la interoperabilidad.

4.2 RDF Schema: el primer escalón de la semántica

RDF afirma hechos, pero no describe la estructura conceptual de un dominio. Ese vacío lo llena RDFS, en su versión 1.1 (W3C, 2014b), vocabulario con semántica acordada que habilita las primeras inferencias. Define la relación de subclase, transitiva, de manera que si un recurso es instancia de una clase y esa clase es subclase de otra, lo es también, por inferencia, de la superior. A ello se añaden el dominio y el rango, cuya semántica es inferencial y no restrictiva, pues declarar un dominio no hace que el motor rechace su uso con un sujeto de otra clase, sino que concluirá que ese sujeto pertenece a la clase del dominio. A diferencia de las bases de datos, donde una restricción de tipo es una guarda que prohíbe, en RDFS dominio y rango son fábricas de afirmaciones, no barreras, distinción entre inferencia y validación que reaparecerá, resuelta de otro modo, al tratar SHACL.

La inferencia de RDFS tiene en mi trabajo un efecto cuantificable. Una consulta por los recursos instancia de mi clase general de concepto, sobre el grafo afirmado, devuelve cero resultados, pues no escribí

que cada concepto fuese instancia de esa clase, sino que organicé los conceptos en una jerarquía de subclases; al activar el razonamiento, la misma consulta devuelve ciento cincuenta y siete resultados. Las treinta entidades externas enlazadas, eso sí, no engrosan ese recuento, porque las he vinculado mediante `skos:exactMatch` que, a diferencia de la identidad estricta de `owl:sameAs`, no propaga el tipo de concepto. El contraste ilustra la diferencia entre lo que un grafo dice y lo que implica, argumento a favor de modelar de forma declarativa en lugar de enumerar cada hecho derivable.

4.3 OWL 2: ontologías y razonamiento

La expresividad de RDFS tiene un techo bajo. Para capturar relaciones más finas (clases disjuntas, propiedades inversas o transitivas, o la identidad de individuos con distintos identificadores) hace falta OWL, en su segunda versión (W3C, 2012), que añade sobre RDF un vocabulario más amplio y una semántica formal anclada en la lógica descriptiva. De su repertorio empleo la distinción entre propiedades de objeto y de datos, las características de las propiedades (transitividad, simetría, funcionalidad) para deducir relaciones que no escribí, y la equivalencia `skos:exactMatch` en lugar de `owl:sameAs`. Mi ontología comprende veinte clases (`owl:Class`), veintiuna propiedades de objeto y siete de datos, dimensión modesta pero suficiente para articular la enseñanza de la programación en Python.

Lo que más ha condicionado mis decisiones es la existencia de perfiles, sublenguajes restringidos que garantizan propiedades computacionales favorables. EL clasifica jerarquías muy grandes en tiempo polinómico, QL atiende grandes volúmenes consultados como base de datos relacional, RL se implementa mediante reglas y escala bien sobre conjuntos grandes de instancias, y OWL 2 DL designa el lenguaje completo, con máxima expresividad pero coste muy elevado en el peor caso. He optado de manera deliberada por OWL 2 RL, compromiso adecuado para un grafo con muchas instancias y un esquema de expresividad moderada como el mío, cuya implementación por reglas se integra con naturalidad tanto con la biblioteca de razonamiento en Python como con el triplestore. El efecto sobre el tamaño es medible, pues el grafo afirmado contiene mil setecientos setenta y dos enunciados y, tras el cierre por las reglas del perfil, asciende a cuatro mil setecientos ochenta y seis. El razonamiento RL no es completo respecto de toda la semántica de OWL 2, pero esa incompletitud no me afecta, porque las construcciones que uso caen dentro del fragmento que RL trata correctamente, y he preferido su escalabilidad y previsibilidad a una expresividad que no iba a aprovechar.

4.4 SPARQL 1.1: la consulta de grafos

Un grafo de conocimiento sería de poca utilidad si no pudiéramos interrogarlo, y para ello la pila dispone de SPARQL 1.1 (W3C, 2013), que opera sobre patrones de grafo donde algunas posiciones son variables. Recurro a SELECT de forma masiva para las consultas analíticas, mientras que CONSTRUCT me ha servido para extraer subgrafos del EKG (una de estas consultas pasa de trescientos cuarenta a trescientos cincuenta y seis ternas), operación central en la integración con modelos de lenguaje, pues el subgrafo extraído alrededor de un concepto es el contexto que se inyecta en la generación aumentada por recuperación.

Dos incorporaciones de la versión 1.1 me han resultado valiosas. Las rutas de propiedad permiten recorrer transitivamente una jerarquía sin conocer la profundidad ni materializar los pasos intermedios; mi consulta de prerrequisitos transitivos de la búsqueda binaria devuelve cinco resultados, y la consulta de ruta entre dos temas devuelve uno. La consulta federada dirige parte de una consulta a un punto de acceso remoto y la combina con el grafo local, concreción técnica del ideal de una Web de datos interconectados; una de mis consultas federadas devuelve veinte filas. Debo ser cauto, no obstante, porque la federación introduce dependencias de disponibilidad y latencia respecto de servicios que no controlo, y por ello en la versión definitiva he tendido a materializar localmente la información externa estable; pero la posibilidad está presente como vía natural de ampliación.

4.5 SHACL: validación de formas

La semántica de RDFS y OWL es de mundo abierto, ya que las declaraciones de dominio, rango o tipo generan nuevas afirmaciones en lugar de rechazar las existentes, filosofía adecuada para razonar sobre conocimiento incompleto pero que no cubre una necesidad real, comprobar que los datos cumplen condiciones de integridad y se ajustan a la forma esperada. Para ello el consorcio estandarizó SHACL (W3C, 2017), que adopta una postura de mundo cerrado, donde una forma describe las condiciones que un nodo objetivo debe satisfacer y el motor reporta como violaciones los incumplimientos. Donde OWL diría «si no tiene tipo, infiérelo», SHACL dice «si no tiene tipo, es un error»; ambos enfoques se complementan y en mi sistema conviven sin fricción.

La validación SHACL ocupa en mi flujo un papel de control de calidad imprescindible, articulado mediante diez formas de nodo (`sh:NodeShape`). El grafo canónico resulta plenamente conforme, con cero violaciones. Para asegurarme de que esa conformidad no es un artefacto de una validación inerte, he construido un fichero con violaciones deliberadas, sobre el que el validador detecta las seis que sembré; este control negativo me da confianza en que el aparato no solo acepta lo correcto, sino que rechaza lo incorrecto en la medida esperada. La validación, eso sí, solo es tan buena como las restricciones declaradas, pues un grafo conforme lo es respecto de las formas escritas, no respecto de toda noción concebible de corrección.

4.6 Linked Data, SKOS y la nube de datos abiertos

La visión de la Web Semántica solo se realiza cuando los datos se publican e interconectan según los principios de los Linked Data —usar IRI, hacerlos dereferenciables e incluir enlaces a otros recursos. Esa interconexión exige reutilizar vocabularios compartidos, y en mi grafo he reutilizado varios consolidados (Dublin Core, FOAF, schema.org y, de manera destacada, SKOS), cuyo detalle corresponde al capítulo de enlazado. SKOS (W3C, 2009) ofrece un vocabulario pensado para describir redes de conceptos y sus relaciones semánticas blandas, distintas de la subsunción lógica estricta; he tejido una red de diecinueve relaciones `skos:broader` (con sus diecinueve `skos:narrower` recíprocas) y dieciséis `skos:related`, sobre un total de dieciséis temas y dieciséis errores típicos catalogados. La Figura 4 representa esa malla conceptual.

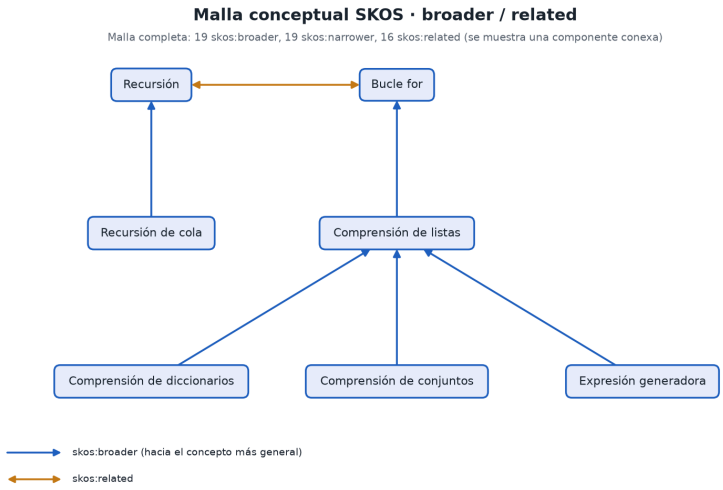

Figura 4. Malla conceptual SKOS del EKG, con 19 relaciones broader, 19 narrower y 16 related.

Esta red es deliberadamente distinta de la jerarquía de subclases de RDFS, pues una cosa es afirmar, en términos lógicos, que toda instancia de una clase lo es de otra, y otra muy distinta afirmar, en términos de organización del conocimiento, que un tema engloba o contextualiza a otro; SKOS me permite expresar lo segundo sin comprometer la semántica formal de lo primero. El enlazado hacia el exterior lo materializo conectando mis conceptos con Wikidata mediante treinta enlaces de equivalencia verificados (`skos:exactMatch`), que enlazan sin fundir, por lo que la consulta por instancias de concepto

bajo inferencia sigue devolviendo ciento cincuenta y siete resultados, y no ciento ochenta y siete. Su alcance es limitado, pues treinta correspondencias son una muestra modesta frente a los ciento cincuenta y siete conceptos, comprobada una a una para evitar identificaciones espurias, y no una integración exhaustiva; he preferido un enlazado pequeño pero fiable a uno amplio pero dudoso.

4.7 Grafos de conocimiento: RDF frente a grafos de propiedades

Tomo como referencia la revisión de Hogan et al. (2021),[2] que entiende un grafo de conocimiento como un grafo de datos cuyo propósito es acumular y transmitir conocimiento del mundo real; se define menos por una tecnología que por una intención, la de organizar la información de modo que el significado quede en la propia topología y no confinado en columnas cuyo sentido haya que reconstruir desde fuera. Donde el modelo relacional esconde las conexiones en claves foráneas, en un grafo la relación es un ciudadano de primera clase, lo que resulta pertinente en un dominio educativo, donde lo interesante casi siempre es la trama de dependencias entre conceptos.

La literatura contrapone dos familias de modelos. RDF representa el conocimiento como tripletas atómicas nombradas con IRI globales, sobre las que se apilan capas de semántica (RDFS, OWL 2), pila estandarizada que es, a mi juicio, su principal ventaja. Los grafos de propiedades (LPG) son más pragmáticos, pues nodos y aristas llevan pares clave-valor sin declararlos como entidades de pleno derecho, lo que hace inmediato atribuir un peso a una arista, pero tradicionalmente han carecido de semántica formal estandarizada y de mecanismos de razonamiento e identidad global comparables a los de RDF. La Figura 5 contrapone ambos modelos sobre el mismo dominio.

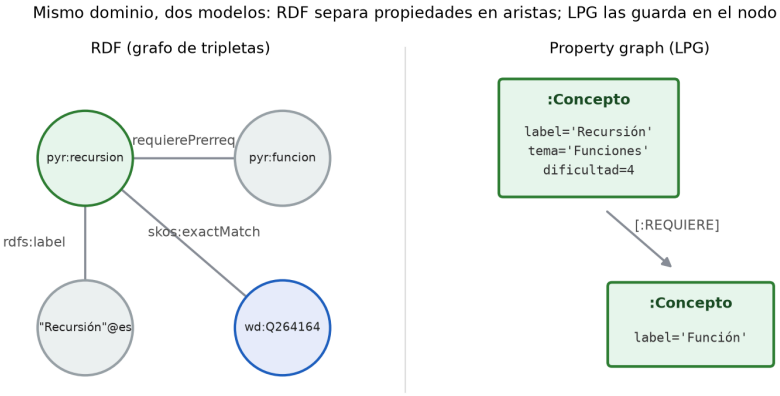

Figura 5. Mismo dominio en dos modelos, donde RDF separa las propiedades en aristas y el grafo de propiedades (LPG) las guarda en el nodo.

Para escenarios analíticos donde priman el rendimiento del recorrido esa ligereza es una virtud; para un proyecto cuyo objetivo es razonar, validar e integrar con vocabularios externos pesan más las garantías formales de RDF —la inferencia predecible de OWL 2 RL, la validación mediante SHACL y la integración con la nube de datos abiertos. La decisión no está exenta de costes, pues el modelado en RDF de relaciones que involucran a más de dos entidades exige técnicas de reificación que recargan el grafo y que un grafo de propiedades resolvería más directamente; asumo ese coste conscientemente, porque las garantías semánticas que obtengo a cambio son centrales para los objetivos del trabajo.

4.8 Construcción, refinamiento y el grafo como base operativa

La creación del grafo fue un proceso, no un acto puntual. Creció y se depuró en sucesivas iteraciones, alternando el modelado del esquema con el poblado de instancias y el control de calidad. Combina una columna vertebral diseñada manualmente, que refleja la estructura conceptual de la programación en Python tal como la concibo desde la docencia, con las instancias que la pueblan (actividades, conceptos,

[2] complementada por tratamientos de manual como el de Kejriwal et al. (2021)

errores típicos y evaluaciones), sobre la separación ya descrita entre la TBox de ciento cincuenta y siete conceptos y la ABox que la encarna.

El refinamiento ocupó un lugar tan importante como la construcción. La inferencia actúa como completado automático (casi dos terceras partes del conocimiento finalmente disponible no se escribieron a mano) y la validación SHACL como corrección, con cero violaciones sobre el grafo canónico y seis sobre la versión sembrada. No quiero, sin embargo, idealizar este proceso, pues la construcción manual del esquema introduce sesgos del autor, y el poblado por plantillas de algunas instancias (del que dependerá la generación de datos de entrenamiento) produce un material regular pero poco diverso, limitación que reconozco y que condiciona la interpretación de los resultados; el refinamiento mitiga problemas, pero no fabrica conocimiento que no esté ya presente o implícito.

Llego así al argumento que da sentido a todo lo anterior. Dotado de semántica formal, el grafo se convierte en una base operativa con cuatro capacidades encadenadas —clasificar, inferir, validar y explotar mediante SPARQL; la clasificación se apoya en la inferencia, esta genera el material que la validación controla, y la validación garantiza que la explotación parta de datos consistentes. Es ahí donde el grafo deja de ser un fin y se convierte en un medio, pues sus resultados alimentan un flujo de recuperación que selecciona subgrafos pertinentes del EKG para inyectarlos como contexto en un modelo de lenguaje, en la línea de la generación aumentada por recuperación (Lewis et al., 2020) y de su variante sobre grafos (Edge et al., 2024). Esas decisiones de modelado[3] prueban que los estándares descritos no son un marco abstracto ajeno al trabajo, sino el repertorio concreto con el que he resuelto los problemas reales de representación que el dominio educativo me planteaba.

[3]incluida la reificación con la que capturo relaciones n-arias, como las evaluaciones que vinculan a la vez un concepto, una actividad y un resultado, materializadas en una clase con diez instancias, o la reificación de afirmaciones para adjuntar la procedencia bibliográfica de los errores típicos

5 Estado del arte

El trabajo se sitúa en la confluencia de tres líneas que hasta hace poco han discurrido por separado —los grafos de conocimiento en educación, la generación aumentada por recuperación para anclar las respuestas de los modelos de lenguaje, y la retroalimentación automática en programación—. Cada una aporta una pieza distinta, pero es en su solapamiento donde se abre el hueco que pretendo ocupar. La Figura 6 sitúa el trabajo en esa intersección.

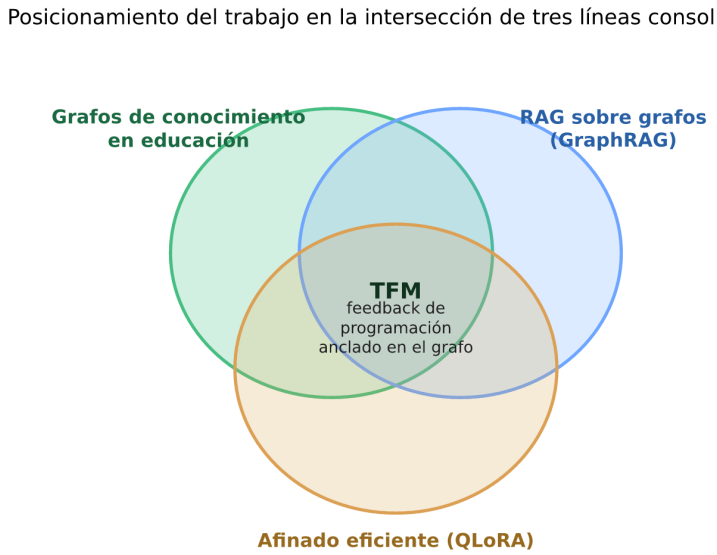

Figura 6. Posicionamiento del trabajo en la intersección de tres líneas consolidadas, los grafos de conocimiento en educación, la generación aumentada por recuperación sobre grafos y el afinado eficiente de modelos.

5.1 Grafos de conocimiento en educación

Hogan et al. (2021) definen el grafo de conocimiento como un grafo de datos enriquecido con un esquema, capaz de soportar inferencia, validación y enlazado. Dos revisiones cartografían su aplicación educativa. Abu-Salih y Alotaibi (2024) inventarían las tareas en que ha demostrado utilidad (currículos, perfil del aprendiz, recomendación, itinerarios, preguntas y respuestas), y Qu et al. (2024) subrayan la heterogeneidad metodológica, desde la extracción automática de entidades hasta el modelado manual del esquema que aquí adopto. Hay consenso en que la estructura explícita aporta valor, porque permite razonar sobre prerrequisitos, generalizaciones y contrastes de un modo que el texto plano no facilita. Con todo, predominan las aplicaciones de organización y recomendación, y lo que ambas revisiones señalan con menos densidad de trabajos (primera observación que vertebra el capítulo) es la evaluación formativa propiamente dicha —el diagnóstico fino de lo que un estudiante ha entendido mal en una producción concreta—. Debo ser prudente, ya que el hecho de que se destaque como área menos transitada no significa que no exista trabajo alguno, sino que su volumen y madurez son menores; esa asimetría basta para apoyar una motivación que coloca el grafo al servicio del diagnóstico antes que del currículo.

Esta orientación enlaza con una tradición pedagógica. Hattie y Timperley (2007) sostienen que el feedback más eficaz responde a tres preguntas —hacia dónde voy, cómo voy y qué sigue—. Uno que aspire a rigor conceptual necesita, en consecuencia, una representación de hacia dónde apunta cada concepto, de qué prerrequisitos lo sostienen y de qué errores típicos lo acechan; el grafo, así, no es un fin en sí mismo, sino la infraestructura que dota de contenido verificable a un feedback que de otro modo quedaría reducido a señalar errores sintácticos o casos de prueba fallidos. De ahí que represente no solo

los conceptos, sino sus relaciones de prerrequisito, de contraste y los errores conceptuales recurrentes documentados en la literatura.

5.2 Generación aumentada por recuperación y su variante sobre grafos

La segunda línea nace de una limitación conocida de los modelos de lenguaje, que almacenan su conocimiento de forma paramétrica e inauditable, lo que conduce a la alucinación (afirmaciones plausibles pero falsas) y a la imposibilidad de rastrear su procedencia. La generación aumentada por recuperación (RAG), de Lewis et al. (2020), la abordó combinando el componente paramétrico con un índice de documentos del que se recuperan en inferencia los pasajes pertinentes; actualiza el conocimiento sin reentrenar, reduce la alucinación y abre la puerta a la trazabilidad, hasta ser hoy un estándar de facto. Ahora bien, la formulación original recupera texto plano, y cuando el conocimiento relevante reside en la articulación de muchas piezas dispersas la similitud entre textos resulta insuficiente. De ahí la variante GraphRAG, en que Edge et al. (2024) transforman el corpus en un grafo de entidades y relaciones con resúmenes jerárquicos de comunidades, de modo que la recuperación opera sobre estructura y no sobre pasajes aislados.

El alcance pide un matiz. Ese GraphRAG construye el grafo automáticamente desde texto y se orienta a la sumarización de preguntas amplias, escenario distinto del que aquí abordo, en el que parto de un grafo modelado manualmente, validado con SHACL y enlazado a datos abiertos, explotado mediante SPARQL para recuperar subgrafos reducidos dirigidos al diagnóstico de un fragmento de código. La filiación conceptual es clara —recuperar sobre estructura supera a recuperar sobre texto—, pero la realización difiere, y sería un exceso presentar el sistema como aplicación directa de aquella propuesta; esa familia proporciona, eso sí, el marco dentro del cual adquiere sentido integrar un grafo educativo con un modelo de lenguaje.

5.3 Generación automática de retroalimentación en programación

La tercera línea es la de mayor solera, y sitúa el problema concreto. La revisión de Keuning et al. (2019), sobre más de un centenar de herramientas, propone una taxonomía de los tipos de feedback con un diagnóstico revelador. La inmensa mayoría se concentra en identificar qué está mal —un fallo de compilación, un caso de prueba que no pasa, una desviación respecto de una solución de referencia—, mientras que los niveles más exigentes (pistas sobre cómo proceder, explicación conceptual del error) quedan desatendidos; la corrección automática escala bien pero diagnostica mal.

El sustrato técnico lo explica. Los enfoques clásicos se apoyan en el análisis estático, la ejecución contra casos, la comparación con soluciones modelo o la transformación a forma canónica, eficaces para localizar discrepancias pero ciegos al marco conceptual que el estudiante no domina; saber que un programa falla en un caso límite no equivale a saber que no ha comprendido la indexación de listas. La propia revisión documenta catálogos de errores conceptuales recurrentes, y en este trabajo la afirmación de que un error afecta a la indexación se ha reificado con procedencia que apunta a esa revisión, de modo que el grafo registra también de dónde procede. La irrupción de los modelos de lenguaje masivos aporta por primera vez un mecanismo capaz de generar explicaciones conceptualmente articuladas, pero esa capacidad llega lastrada por la alucinación y la opacidad, de manera que el feedback puede ser tan elocuente como infundado, lo que justifica buscar un anclaje externo que discipline la generación.

5.4 El hueco en la intersección y posicionamiento del trabajo

Recorridas las tres líneas, el hueco se dibuja en su intersección, como resume la tabla.

Trabajo	Enfoque	Aportación	Relación con este proyecto
Hogan et al. (2021)	Síntesis sobre grafos de conocimiento	Define el grafo de datos enriquecido con esquema, inferencia y enlazado	Fundamento teórico del modelado RDF/RDFS/OWL adoptado
Abu-Salih y Alotaibi (2024)	Revisión de grafos educativos	Inventario de tareas (currículo, perfil, recomendación, itinerarios)	Confirma el predominio de recomendación y el déficit en diagnóstico formativo
Qu et al. (2024)	Revisión de construcción y uso	Heterogeneidad metodológica; modelado manual frente a extracción automática	Justifica la opción por un esquema modelado a mano
Lewis et al. (2020)	RAG textual	Anclaje no paramétrico que reduce alucinación y gana trazabilidad	Marco general de la integración grafo-LLM
Edge et al. (2024)	GraphRAG	Recuperación sobre estructura mediante resúmenes de comunidades	Filiación conceptual; aquí, grafo manual y subgrafos SPARQL acotados
Keuning et al. (2019)	Revisión de feedback en programación	Taxonomía; déficit en feedback explicativo y conceptual	Define el problema concreto y aporta los errores conceptuales modelados

Cada línea tiene su madurez por separado; ninguna cubre el problema completo. El espacio vacío es el de un sistema que, sobre un grafo educativo validado con el rigor de los estándares de la Web Semántica, recupere subgrafos pertinentes para alimentar a un modelo de lenguaje y produzca una retroalimentación formativa de la programación a la vez conceptualmente rica y trazable hasta una estructura inspeccionable.

Ahí se posiciona el trabajo, con mesura. La conjetura que lo vertebra es que esa integración mejora la precisión, la personalización y, sobre todo, la trazabilidad de la retroalimentación frente al modelo en solitario; la aportación específica es su componente semántica, esto es, un grafo propio sobre programación en Python con 157 conceptos, modelado en RDF, RDF Schema y OWL 2,[4] organizado en 16 temas, con 16 misconceptions reificadas con procedencia bibliográfica y articulado mediante 19 skos:broader, 19 skos:narrower y 16 skos:related. El cierre OWL 2 RL eleva la base de 1772 a 4786 triples; la validación con 10 sh:NodeShape resulta conforme, frente a un control negativo que detecta 6; y el enlazado a Wikidata se materializa en 30 skos:exactMatch, que preferí deliberadamente a owl:sameAs para no imponer la fusión lógica de individuos que el perfil OWL-RL deduciría. La explotación se comprueba con SPARQL, donde la consulta 02 (?x a pyedu:Concepto) pasa de 0 a 157 tras la inferencia, el CONSTRUCT 06 va de 340 a 356 y la federada 08 devuelve 20 filas. La novedad no reside en inventar ninguna de las tres líneas, sino en articularlas sobre un caso de uso que ninguna resuelve aisladamente, de forma reproducible en GraphDB y en una interfaz web, pues la aportación es integradora y de demostración, no de ruptura teórica.

[4] con 20 owl:Class, 21 propiedades de objeto y 7 de datos

Para cerrar, delimito lo que no reclamo. No sostengo que los grafos educativos no se hayan aplicado nunca a la evaluación, sino que su uso para el diagnóstico conceptual fino está mucho menos consolidado que el de recomendación; ni que el sistema sea una variante novedosa de GraphRAG en sentido estricto, sino que se inscribe en su estela con una realización propia y más acotada; ni que la evaluación experimental zanje la cuestión. El benchmark se obtuvo sobre 50 casos retenidos, con el Sistema D (híbrido) como mejor opción, que aun así se queda en 0,76 en categoría y 0,54 en concepto; la generalización a código real (Dublin) revela una inversión fuera de distribución (A 0,802, B 0,433, C 0,733, D 0,323), y el panel de jueces arroja un acuerdo apenas leve (Fleiss 0,213). Con esas cautelas, la posición del trabajo es clara, pues es un puente, construido con los estándares de la Web Semántica, entre tres tradiciones maduras que rara vez se habían encontrado sobre el terreno común de la retroalimentación formativa automática de la programación.

6 Metodología y reproducibilidad

El trabajo participa de dos naturalezas. Por un lado es ingeniería, pues he construido un artefacto no trivial —un grafo de conocimiento educativo (EKG) y los componentes que lo explotan— que exige justificar las decisiones de diseño tanto como los resultados. Por otro, aspiraba a contrastar empíricamente si integrar conocimiento estructurado en un grafo aporta algo medible a la retroalimentación formativa que un modelo de lenguaje ofrece a quien aprende a programar en Python; esa doble condición me llevó a un diseño mixto, en el que desarrollo y evaluación se entrelazan. Debo declarar de entrada una cautela que recorre el capítulo. El desarrollo descansa sobre estándares maduros (RDF, RDFS, OWL, SHACL y SPARQL) y es mecánicamente verificable, pero la evaluación opera sobre una muestra pequeña y sobre casos generados en parte por plantillas. He preferido, por ello, reportar resultados modestos pero defendibles antes que inflar las conclusiones.

6.1 Un diseño mixto: ingeniería y evaluación

La elección no es acomodaticia. La literatura sobre grafos de conocimiento en educación insiste en que su valor no se demuestra por su existencia, sino por su capacidad de soportar tareas concretas (Abu-Salih & Alotaibi, 2024; Qu et al., 2024), y la investigación sobre retroalimentación advierte que su eficacia depende de que oriente sobre la tarea, el proceso y la autorregulación (Hattie & Timperley, 2007); ahora bien, las revisiones del feedback automático en programación describen un campo fértil pero fragmentado (Keuning et al., 2019), por lo que juzgué necesario confrontar sistemas con y sin grafo en una tarea idéntica. El diseño articula así tres tipos de evidencia. La de construcción es el propio grafo, con sus métricas y garantías formales, verificable por sí misma. La cuantitativa es un experimento comparativo, donde mido aciertos de categoría y de concepto del error mediante un juez automático. La cualitativa son valoraciones graduadas de identificación y trazabilidad que un acierto binario no refleja. Esta triangulación es, a mi juicio, lo que distingue la investigación de un mero ejercicio de programación.

6.2 Fases del trabajo

He organizado el desarrollo en tres fases sucesivas pero realimentadas, y lo presento ordenado por claridad, aunque en la práctica iteré sobre el esquema cada vez que la fase experimental revelaba carencias de cobertura conceptual. La Figura 7 resume esas tres fases.

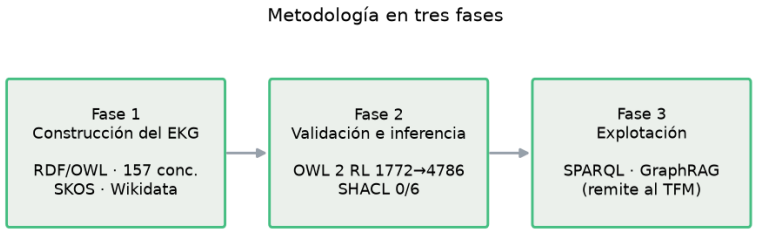

Figura 7. Las tres fases del trabajo, a saber, construcción del EKG, validación e inferencia, y explotación.

La primera fase modeló la programación introductoria en Python como un grafo conforme a la Web Semántica (W3C, 2014a; Hogan et al., 2021), con una separación deliberada entre el esquema, en `pyedu:` (la TBox), y las instancias, en `pyr:` (la ABox). El grafo canónico reúne 157 conceptos organizados mediante 20 clases (`owl:Class`), 21 propiedades de objeto y 7 de datos, y materializa 1772 enunciados afirmados; para la semántica jerárquica recurrí a RDF Schema y a un perfil OWL 2 RL (W3C, 2014b; W3C, 2012), de modo que las relaciones de subsunción (`rdfs:subClassOf`) pudieran razonarse, y el cierre deductivo eleva los enunciados a 4786 (3014 inferidos), expansión que manifiesta conocimiento implícito hecho explícito.

En el modelado cuidé tres aspectos. El primero, reutilizar vocabularios consolidados, con SKOS para las relaciones conceptuales blandas (W3C, 2009), Dublin Core para la procedencia, FOAF para los agentes y schema.org para la alineación. El segundo, el enlazado externo, con 30 enlaces `skos:exactMatch`

verificados hacia Wikidata; opté por `skos:exactMatch` y no por `owl:sameAs` para no imponer la fusión lógica de individuos que el perfil OWL-RL realizaría, y afirmé la equivalencia sin colapsar identidades; a ello sumé 19 relaciones `skos:broader`, 19 `skos:narrower` y 16 `skos:related`, de granularidad pedagógica distinta de la jerarquía formal, junto con 16 temas y 16 errores típicos. El tercero, la expresividad, pues como RDF solo admite relaciones binarias, reifiqué las N-arias mediante clases intermedias (`EvaluacionDeConcepto` y `EvaluacionActividad`, esta con 10 instancias) y empleé la reificación clásica con `rdf:Statement` y `dcterms:source` para anclar cada error a su fuente, lo que después permitió evaluar la trazabilidad. Cerré la fase validando con SHACL (W3C, 2017), donde definí 10 `sh:NodeShape` y comprobé que el grafo canónico es conforme sin violaciones; para no validar solo lo que ya sé correcto, preparé un fichero con violaciones deliberadas y verifiqué que el validador detecta las seis introducidas, prueba negativa que demuestra que la conformidad no resulta vacua.

La segunda fase trasladó el grafo de la representación a la explotación. La consulta se apoya en SPARQL (W3C, 2013), donde la pregunta por los individuos de tipo `pyedu:Concepto` no devuelve nada sin inferencia y, activado el razonamiento, devuelve los 157 conceptos deducidos a través de `rdfs:subClassOf`; ese contraste de cero a ciento cincuenta y siete ilustra por qué un grafo razonado ofrece una base más rica que la suma de aserciones. Las consultas confirman la coherencia, pues prerrequisitos transitivos de la búsqueda binaria devuelve 5, ruta de aprendizaje 1, el `CONSTRUCT` 06 pasa de 340 a 356 enunciados y una consulta federada hacia Wikidata recupera 20 filas. Sobre esta base construí la integración con modelos según dos estrategias. La primera, GraphRAG, variante de RAG (Lewis et al., 2020) en la que el contexto procede de un subgrafo extraído por SPARQL (Edge et al., 2024). La segunda, el *fine tuning* mediante QLoRA, que cuantiza los pesos a 4 bits y entrena solo adaptadores de bajo rango (Hu et al., 2021; Dettmers et al., 2023). Afiné un Qwen2.5-Coder-7B-Instruct con LoRA de rango 16 y alpha 32 sobre una RTX 5090 de 32 GB. Una primera versión sin regularizar mostró sobreajuste claro.[5] Una segunda, regularizada con NEFTune y dropout, invirtió la tendencia y redujo la pérdida held-out de 1.051 a 1.028, con una precisión de token sobre datos no vistos de 0.842. Reporto ambos intentos porque el contraste forma parte del resultado. Debo advertir, no obstante, que el conjunto de entrenamiento se generó en parte mediante plantillas, lo que introduce riesgo de fuga de información, y por ello la evaluación se restringe a esqueletos held-out que el modelo no vio.

La tercera fase es la evaluativa. Diseñé un experimento entre cuatro sistemas que aíslan las contribuciones del grafo y del *fine tuning*, donde A es el modelo base (llama3.1:8b; Grattafiori et al., 2024) sin grafo, línea de referencia; B el *fine tuning* con QLoRA sin grafo; C base aumentado con GraphRAG; y D combina *fine tuning* y GraphRAG, partición factorial que permite atribuir las diferencias al conocimiento estructurado, a la adaptación o a su interacción. La medición se hizo sobre 50 casos held-out con un juez independiente y mayor (qwen2.5:32b; Yang et al., 2024), lo que reduce, sin eliminar, el sesgo de evaluar un modelo con otro de escala semejante. D resultó el mejor, con 0.76 en acierto de categoría y 0.54 en concepto, y la lectura es complementaria, pues el *fine tuning* destaca en clasificar el tipo de error mientras el GraphRAG lo hace en el concepto concreto y la trazabilidad, justo donde el grafo aporta conocimiento explícito. Mantengo, sin embargo, la cautela. Ningún objetivo cuantitativo del anteproyecto se alcanza, la evaluación se hizo con juez automático y sin panel humano, y una muestra de este tamaño no autoriza inferencias estadísticas firmes. Un dato modera el optimismo. Al trasladar los sistemas a código real ajeno al dominio de entrenamiento (ejercicios de Dublin), la jerarquía se invierte, con un comportamiento fuera de distribución que penaliza al híbrido (A 0.802, B 0.433, C 0.733, D 0.323) y un panel de jueces con acuerdo apenas leve (Fleiss 0.213).

6.3 Reproducibilidad, entorno y herramientas

La credibilidad de un trabajo así depende de que un tercero pueda reconstruir sus resultados, y lo he procurado desde el diseño. El grafo se serializa íntegramente en Turtle, legible y versionable, y todo el proceso —construcción, razonamiento, validación SHACL, consultas, extracción de subgrafos, *fine

[5] la pérdida de entrenamiento descendía hasta valores próximos a 0.01 mientras la held-out crecía

tuning* y evaluación— está implementado en scripts ejecutables y documentados; para que los resultados no dependan de un entorno irrepetible fijé las versiones en un fichero de requisitos y, en las etapas aleatorias, las semillas. El entorno de referencia es Python 3.12.10 con `rdflib` 7.6, `owlrl`, `pyshacl`, `matplotlib` y `networkx`; `typst-py` y `python-docx` para la memoria; y Ollama (`llama3.1:8b`, `qwen2.5:32b`, `nomic-embed-text`) sobre una RTX 5090 (CUDA 12.8). La secuencia de reconstrucción se resume en estos comandos, ejecutados desde el directorio del entregable salvo donde se indique. La enuncio aquí una sola vez, porque este capítulo es el que concentra la reproducibilidad.

7 1. Artefactos del grafo: inferir, validar, consultar, exportar

python scripts/inferir.py && python scripts/validar.py \

&& python scripts/consultar.py && python scripts/exportar.py

8 2. Datos de la Grafoteca (JSON Cytoscape)

python scripts/exportar_grafos_json.py # -> salidas/grafos/grafoteca.json

9 3. Diagramas RDF/TBox (si “dot” esta instalado)

python -m rdflib.tools.rdf2dot ontologia/ekg-python-150.ttl | dot -Tsvg -o salidas/grafos/ekg_rdf.svg

python -m rdflib.tools.rdfs2dot ontologia/ekg-python-150.ttl | dot -Tsvg -o salidas/grafos/ekg_tbox.svg

10 4. Construir la memoria (PDF y DOCX)

cd memorias && python generar_pdf.py todo && python ensamblar_docx.py

Tras la secuencia, las figuras de grafo se regeneran sobre `grafoteca.json` y la curva de pérdida desde el `loss_history.json` del *fine tuning* v3; un cuaderno (`notebook/ekg.ipynb`) reúne RDFLib, owlrl, las ocho consultas y la validación SHACL, y la web `proyecto.html` es autocontenida. Reconozco que la reproducibilidad de la parte basada en modelos es más frágil que la de la semántica. Mientras un razonador OWL-RL produce siempre el mismo cierre sobre el mismo grafo, la generación de un modelo varía entre ejecuciones por la coma flotante y el muestreo. Las métricas experimentales son, por tanto, estimaciones, no constantes. En cuanto a las herramientas, en su mayor parte las que la asignatura introduce, en el núcleo semántico trabajé con `rdflib` (RDFLib Team, s. f.), `owlrl`, `pyshacl` (Sommer & Car, s. f.) y `SPARQLWrapper`, y empleé GraphDB con sus reglas OWL2-RL (Ontotext, s.f.) como razonador alternativo, para contrastar el razonamiento programático con el de un triplestore consolidado; la interfaz se construyó con FastAPI y Cytoscape.js, la inferencia se orquestó con Ollama y esta memoria se redacta con MyST Markdown. La elección de estándares abiertos y utilidades documentadas es, en sí misma, una decisión orientada a la transparencia que he defendido a lo largo del capítulo.

11 Modelado ontológico del EKG

Este capítulo justifica las decisiones que dieron forma a la ontología que sustenta el Grafo de Conocimiento Educativo (en adelante, EKG) sobre programación en Python; no enumero clases y propiedades, sino que explico por qué cada elección sirve al propósito último, que es ofrecer retroalimentación formativa automática y trazable a quienes aprenden a programar. El modelo no es neutro respecto de esa finalidad, pues declarar una propiedad como transitiva, reificar una relación o renunciar a fijar un dominio repercute en lo que el grafo puede contestar y demostrar. El lenguaje es OWL 2 en su perfil RL, la serialización es Turtle y el razonamiento se materializa con `owlrl` sobre RDFLib y, de forma equivalente, con el motor OWL2-RL de GraphDB (Ontotext, s.f.). Las cifras corresponden al grafo canónico congelado para la validación. Se trata de 157 conceptos propios, 1772 enunciados afirmados que ascienden a 4786 tras el cierre deductivo, y 20 clases, 21 propiedades de objeto y 7 de datos. La Figura 8 muestra la jerarquía de clases de la TBox.

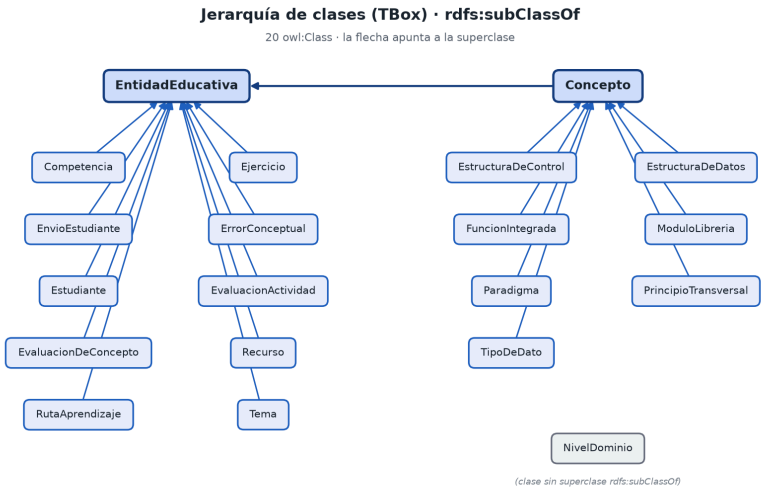

Figura 8. Jerarquía de las 20 clases del EKG mediante rdfs:subClassOf (TBox).

11.1 Dos espacios de nombres: separar el esquema de los datos

La primera decisión estructural, y posiblemente la más consecuente, fue separar el esquema de las instancias mediante dos espacios de nombres, donde `pyedu:` (por *Python education*) alberga la TBox —clases, propiedades, jerarquías y restricciones—, y `pyr:` (por *Python resources*) alberga la ABox, esto es, los individuos concretos. La adopté porque esquema y datos evolucionan con cadencias distintas y conviene versionarlos por separado, porque quien reutilice `pyedu:` no debe arrastrar mis instancias, y porque muchas consultas SPARQL y formas SHACL operan sobre un solo plano y la disciplina de prefijos hace aflorar antes los errores de confusión de niveles. Es una convención de diseño y no una barrera lógica, pues OWL no impide que un recurso de `pyr:` sea sujeto de una aserción cuyo objeto vive en `pyedu:`, y ese es el puente entre planos.

11.2 La taxonomía de clases

El armazón se apoya en veinte clases OWL que conforman dos troncos taxonómicos más unas clases auxiliares; procuré una jerarquía poco profunda pero significativa, donde cada nivel de subsunción habilite inferencias útiles. El primer tronco gira en torno a `pyedu:EntidadEducativa`, raíz de todo aquello con función pedagógica —conceptos, actividades, errores típicos, recursos—, de manera que cualquier «materia de enseñanza o evaluación» se reconozca por subsunción y una sola consulta sobre la clase raíz recupere, gracias al cierre RDFS, todos los individuos de cualquier subclase. El segundo gira en torno a `pyedu:Concepto`, corazón semántico del grafo. Bajo él despliego subcategorías por naturaleza temática y nivel de abstracción —sintácticos, de control, de datos, de orientación a objetos, de manejo de errores—, en una partición operativa y no en una clasificación canónica. La despliego

con dos fines. El primero, que el sistema de feedback ubique el error de un estudiante en una categoría reconocible (la métrica de «acierto de categoría» del capítulo de resultados); el segundo, dar estructura a la red de prerrequisitos. Los 16 temas de este nivel y las 16 misconceptions catalogadas se apoyan en esta partición.

Completan las veinte las clases auxiliares o de relación, sobre todo `pyedu:EvaluacionDeConcepto` y `pyedu:EvaluacionActividad`, que materializan relaciones N-arias mediante reificación. Que sumen veinte es fruto de la parsimonia, pues introduje cada clase solo cuando aportaba una inferencia, una restricción SHACL o una consulta que de otro modo no podía expresarse con naturalidad.

11.3 Propiedades de objeto: la malla de relaciones

Sobre el esqueleto de clases tendí una malla de veintiuna propiedades de objeto (Figura 9), que traté con más cuidado axiomático que las clases, porque un descuido en sus características lógicas puede desencadenar inferencias incorrectas que contaminan todo el cierre. La más cargada de semántica es `pyedu:esPrerrequisitoDe`, que declaré `owl:TransitiveProperty`, de modo que si las variables son prerrequisito de las asignaciones, y estas de los bucles, las variables lo son también de los bucles, y el cierre transitivo recupera toda la clausura previa de un concepto, lo que permite al feedback razonar sobre qué conocimientos anteriores podrían estar fallando. Ahora bien, la transitividad no es inocua. En presencia de un ciclo haría que todos sus elementos fueran prerrequisito de sí mismos, de manera que traté la red como un grafo dirigido acíclico y lo verifiqué. OWL 2 RL no detecta ni prohíbe los ciclos, por lo que esa garantía descansa en una validación externa, no en una propiedad lógica del modelo. En el polo opuesto declaré `pyedu:contrastaCon` como `owl:SymmetricProperty`, para conceptos que se confunden —listas frente a tuplas, `==` frente a `is`—, con lo que axiomaticé la reciprocidad sin afirmar ambos sentidos; y empleé `rdfs:subPropertyOf` para declarar una propiedad como especialización de otra y consultar al detalle o al agregado sin duplicar aserciones (W3C, 2014b).

Características lógicas de las propiedades de objeto

Propiedad / mecanismo	Característica OWL	Ejemplo real	Efecto
esPrerrequisitoDe	owl:TransitiveProperty	variables → asignaciones → bucles	cierra la cadena de prerrequisitos
contrastaCon	owl:SymmetricProperty	«listas frente a tuplas», «== frente a is»	relación bidireccional
inversas	owl:inverseOf	esPrerrequisitoDe ↔ tienePrerrequisito	navegación en ambos sentidos
especialización	rdfs:subPropertyOf	relacionadoConceptualmenteCon	jerarquía de relaciones

elaboración propia

Figura 9. Características lógicas de las propiedades de objeto.

Llego a una decisión que ilustra el tono de cautela del proyecto, pues dominios (`rdfs:domain`) y rangos (`rdfs:range`) no son en RDFS y OWL restricciones de validación sino axiomas de inferencia; declarar el dominio de una propiedad no impide usarla con un sujeto de otra clase, sino que infiere que ese sujeto pertenece también a la clase del dominio, lo que puede propagar tipos erróneos por todo el cierre. Por eso los apliqué con cautela deliberada. Los declaré donde la inferencia de tipo es siempre correcta y los omití en las propiedades más polimórficas, y delegué el control de su buen uso en SHACL, que valida sin inferir (W3C, 2017). Esta división de responsabilidades, OWL para lo que debe inferirse y SHACL para lo que debe validarse, es a mi juicio una de las lecciones más valiosas del trabajo, pues el razonador opera bajo mundo abierto y no detecta ausencias, mientras que SHACL opera bajo mundo cerrado y sí puede exigir presencias. Usar el dominio como restricción de integridad habría sido un error categorial.

11.4 Propiedades de datos: literales tipados y etiquetas multilingües

Definí siete propiedades de datos conforme a la distinción de OWL entre `owl:ObjectProperty` y `owl:DatatypeProperty`, porque el razonador y el validador tratan de modo distinto a un literal y a un recurso. He puesto cuidado en tipar los literales con XML Schema (XSD), porque `xsd:integer`, `xsd:boolean` o `xsd:date` habilitan comparaciones y ordenaciones correctas en SPARQL que serían frágiles sobre cadenas, y permiten que SHACL valide el tipo esperado (W3C, 2014a). El texto en lenguaje natural lo expreso mediante `rdfs:label` con etiquetas de idioma, típicamente español e inglés, donde

la inglesa facilita el alineamiento con Wikidata, y las etiquetas bien marcadas mejoran el texto que recibe un modelo de lenguaje cuando un fragmento del grafo se inyecta como contexto en recuperación aumentada. La cobertura multilingüe, en cambio, no es uniforme, porque prioricé el español y el inglés solo donde el enlazado lo hacía conveniente; es una asimetría asumida y no un descuido.

11.5 Reutilización de vocabularios: SKOS, Dublin Core, FOAF y schema.org

Uno de los principios que más quise honrar es no reinventar lo que la comunidad de datos enlazados ya ha estandarizado, condición para que el grafo sea interoperable y desreferenciable. SKOS (W3C, 2009) es el vocabulario con que tejo la red conceptual temática, donde `skos:broader`, `skos:narrower` y `skos:related` organizan los conceptos sin las ataduras lógicas de la subsunción OWL.[6] De Dublin Core tomé `dcterms:source`, con la que adscribo procedencia bibliográfica a los enunciados reificados, ya que el hecho de que un error típico esté documentado en una fuente concreta no es accesorio en un sistema cuya trazabilidad —una de las métricas que evalúo— depende de remontar cada afirmación hasta su origen. De FOAF tomé el vocabulario para agentes y personas y de schema.org términos de gran adopción.

El enlazado externo culmina en la conexión con Wikidata mediante treinta aserciones `skos:exactMatch` verificadas, no `owl:sameAs`, porque este habría forzado al razonador a fusionar mi concepto con la entidad de Wikidata como un único individuo y a propagar todas sus pertenencias de clase, lo cual es semánticamente incorrecto. En cambio, `skos:exactMatch` los enlaza como equivalentes a efectos de alineamiento sin imponer esa fusión. El efecto inferencial se manifiesta por otra vía, pues la consulta que pregunta por los individuos de tipo `pyedu:Concepto` devuelve cero resultados sin razonamiento[7] y devuelve 157 con razonamiento RDFS/OWL-RL activado; las treinta entidades de Wikidata no engrosan la cifra, porque `skos:exactMatch` no propaga el tipo. Este contraste de cero frente a 157 es la demostración más nítida de que el razonamiento hace aflorar conocimiento implícito.

11.6 Dos redes superpuestas: `skos:broader` no es `rdfs:subClassOf`

El EKG mantiene dos redes jerárquicas en paralelo sin confundirlas, distinción sutil pero de consecuencias precisas. Por un lado, la taxonómica de `rdfs:subClassOf`, que organiza las clases en subsunción, y la conceptual de las diecinueve relaciones `skos:broader`, que organiza los conceptos temáticamente. Afirmar que la clase A es `rdfs:subClassOf` de B es una implicación estricta y monótona, pues todo individuo de A lo es necesariamente de B, y el razonador propaga esa pertenencia de manera inexorable; que el concepto X sea `skos:broader` que Y dice algo más laxo, que es un tema más amplio, sin implicar pertenencia de instancias ni inferencia de subsunción. Si modelara la red temática con `rdfs:subClassOf` —tratando «estructuras de control» como superclase de «bucle while»—, el razonador propagaría consecuencias innecesarias; con `skos:broader` expreso la organización temática que el sistema de feedback necesita para navegar de lo general a lo particular sin inflar el cierre (W3C, 2009). La Figura 10 ilustra el subgrafo del concepto «Recursión».

[6] el grafo contiene 19 relaciones `skos:broader`, otras 19 `skos:narrower` y 16 `skos:related`

[7] porque los conceptos no se declaran con ese tipo, sino que lo heredan por subsunción

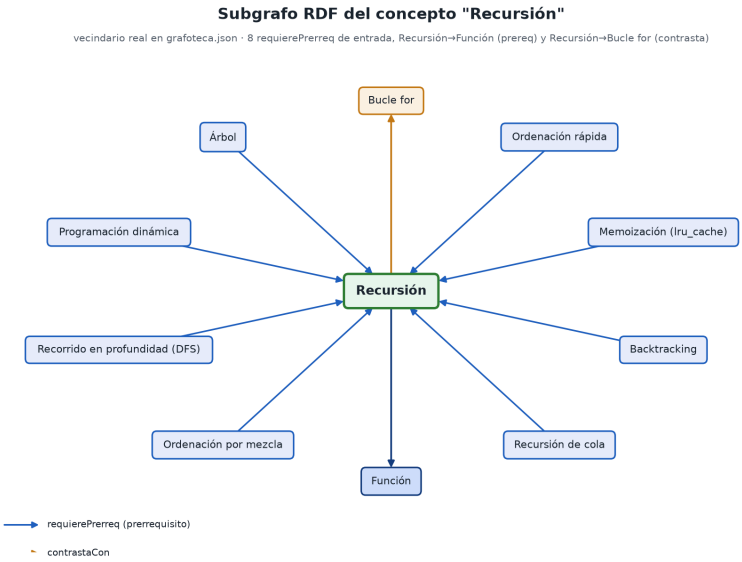

Figura 10. Subgrafo RDF del concepto «Recursión», con prerrequisitos, contrastes, error y tema.

11.7 Relaciones N-arias y reificación: cuando una propiedad binaria no basta

La última familia de decisiones aborda un límite expresivo de RDF y OWL. El modelo es binario, pues toda aserción es un triple sujeto-predicado-objeto que relaciona dos recursos, y el mundo educativo está lleno de relaciones que vinculan a tres o más participantes. Para estas relaciones N-arias recurrí a la reificación, que cosifica la relación misma como un individuo al que se enganchan como propiedades sus participantes. El primer caso es `pyedu:EvaluacionDeConcepto`, donde la evaluación involucra al menos al contexto, al concepto evaluado y al resultado, y aplanarla en triples sueltos haría ambiguo qué resultado corresponde a qué concepto en qué contexto, mientras que al reificarla cada evaluación pasa a ser un nodo identificable. El segundo, más poblado, es `pyedu:EvaluacionActividad`, del que el grafo contiene diez instancias que enlazan la actividad, los conceptos que pone en juego y los atributos del resultado; ese volumen me permitió comprobar que el patrón se comporta correctamente bajo razonamiento y validación.

Para registrar procedencia empleo además `rdf:Statement`, el vocabulario estándar de reificación de enunciados (con `rdf:subject`, `rdf:predicate` y `rdf:object`); cuando afirmo que cierto error es característico de cierto concepto, reifico ese enunciado y le adjunto `dcterms:source`. La motivación es metodológica, porque un sistema de retroalimentación formativa no debe ser una caja negra que dictamina sin justificar, sino remontar cada afirmación hasta una fuente verificable, trazabilidad que mido y que la incorporación del grafo mejora frente al modelo base. `rdf:Statement` tiene un coste conocido, pues multiplica los triples, complica algunas consultas y carece de un tratamiento semántico tan integrado como alternativas más modernas. Opté por él, decisión conservadora tomada a sabiendas de sus contrapartidas, porque aquí pesaban más la robustez y la compatibilidad con OWL 2 RL.

11.8 Recapitulación de las decisiones de diseño

Las decisiones descritas conforman un diseño coherente, pues separé esquema y datos en `pyedu:` y `pyr:`; declaré las características lógicas de las propiedades donde axiomatizan correctamente el dominio, delegando la integridad en SHACL; reutilicé SKOS, Dublin Core, FOAF y schema.org en lugar de acuñar vocabulario propio; enlacé el grafo con treinta `skos:exactMatch` a Wikidata, cuyo efecto se manifiesta en el salto de cero a 157 conceptos al activar el razonamiento; y resolví las relaciones N-arias mediante reificación, con `rdf:Statement` anotado con `dcterms:source` para la procedencia. El modelo materializado parte de 1772 enunciados afirmados y crece hasta 4786 tras el cierre deductivo OWL-RL; cada elección se tomó con conciencia de sus consecuencias y, cuando una técnica traía contrapartidas, las he declarado. El diseño ontológico no es aquí un trámite previo a lo interesante, sino el lugar donde se decide qué podrá el grafo demostrar, validar y explicar más adelante.

12 Inferencia y razonamiento

El rasgo que separa con más nitidez una base de conocimiento de un simple repositorio es la capacidad de inferencia. Una colección de tripletas RDF, por cuidadosamente modelada que esté, no deja de ser una enumeración de hechos explícitos; un grafo de conocimiento aspira a que la semántica formal del esquema se traduzca en consecuencias lógicas computables. El sistema afirma así por derecho propio hechos que nadie escribió pero que se siguen necesariamente de los que sí. Esa es la diferencia entre almacenar y razonar.

12.1 De la colección de tripletas a la base de conocimiento

Las normas RDF (W3C, 2014a) definen un modelo de tripletas, pero por sí solas no aportan capacidad inferencial; lo hacen las capas superpuestas. RDF Schema (W3C, 2014b) introduce un vocabulario mínimo para jerarquías de clases y propiedades, dominios y rangos, y OWL 2 (W3C, 2012) lo amplía con propiedades transitivas, simétricas, inversas y relaciones de identidad. Cada constructo lleva una teoría semántica que especifica qué tripletas son verdaderas dadas ciertas afirmadas, y un motor de razonamiento es el programa que calcula ese conjunto de consecuencias, el *cierre* del grafo. La cuestión no es, entonces, si mis tripletas «contienen» información implícita —siempre la contienen—, sino si dispongo de maquinaria que la haga explícita. Sin ella, una consulta SPARQL puede devolver vacío no porque la respuesta falte, sino porque está latente y nadie la ha desplegado (Hogan et al., 2021). El grafo afirmado y el razonado son dos objetos distintos, y conviene saber sobre cuál se opera.

12.2 Dos motores, un mismo perfil: owlrl y GraphDB

He optado por una arquitectura redundante, con dos motores del mismo perfil. El primero es `owlrl`, biblioteca de Python sobre los grafos en memoria de `rdflib`; funciona por materialización, pues dado un grafo afirmado calcula su cierre y añade físicamente las tripletas inferidas. La combinación que empleo une RDFS con el perfil OWL 2 RL, que aporta el razonamiento sobre jerarquías junto a transitividad, inversas, simetría e identidad, y mantiene la terminación y tratabilidad que distinguen al perfil RL frente a otros más expresivos pero intratables; resulta cómodo en desarrollo, porque razono en el mismo proceso en el que cargo datos, ejecuto SHACL y lanzo consultas. El segundo es GraphDB (Ontotext, s.f.), el triplestore que uso como almacén persistente y acceso SPARQL, que ofrece un *ruleset* OWL 2 RL y, a diferencia de `owlrl`, razona de forma incremental, mantiene el cierre actualizado al modificar tripletas y lo expone a través del endpoint. Que ambos compartan el perfil es un requisito de diseño, que me permite trasladar los resultados experimentales a la explotación sin sorpresas y detectar errores que, con un único motor, habrían pasado inadvertidos.

12.3 El contraste NONE frente a RDFS/OWL-RL

La forma más elocuente de mostrar qué aporta la inferencia es desactivarla. He comparado el grafo canónico sin régimen (`NONE`) frente al combinado RDFS más OWL 2 RL, donde el grafo afirmado contiene 1772 enunciados y tras el cierre OWL-RL asciende a 4786, esto es, 3014 hechos inferidos que estaban latentes. No conviene leer este crecimiento como virtud en sí —un grafo inflado con tripletas triviales no es mejor—, pero sí como señal de que la semántica formal hace trabajo efectivo. La Figura 11 cuantifica ese salto.

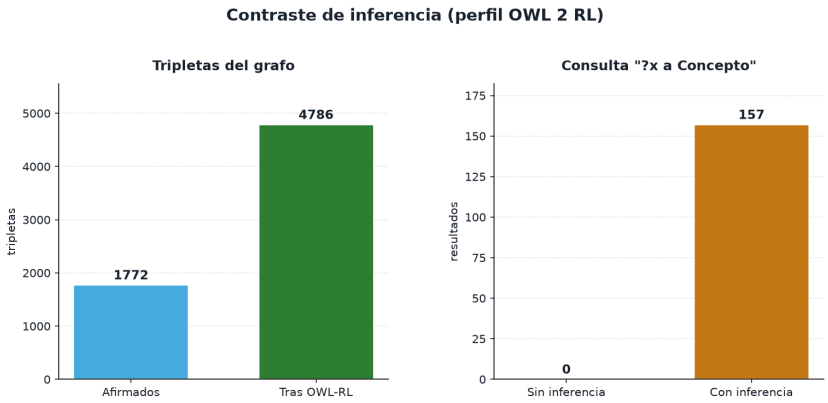

Figura 11. El razonamiento OWL 2 RL materializa 3.014 triples nuevos, 1.772⟶4.786, y la consulta de conceptos pasa de 0 a 157.

El segundo contraste toca la utilidad práctica de las consultas. La consulta `?x a pyedu:Concepto` devuelve sin inferencia cero resultados, porque los conceptos se declaran como instancias de subclases más específicas y es `rdfs:subClassOf` la que deduce que todo individuo de una subclase lo es de la superclase, y con el régimen RDFS/OWL-RL devuelve 157, los conceptos propios del grafo. La diferencia entre cero y 157 no es de grado sino de naturaleza. Conviene reparar en lo que *no* aparece, ya que las 30 entidades de Wikidata que enlazo no se infieren como instancias de `pyedu:Concepto`, y eso es lo que persigo. Esos 30 enlaces se afirman mediante `skos:exactMatch`, cuya semántica declara una equivalencia sin fusionar lógicamente los recursos, de modo que no propaga el tipo de un extremo al otro; con `owl:sameAs` el razonador habría tratado ambos como un único individuo y habría fundido con mi modelo un recurso externo que no controlo. Un enlazado riguroso consiste tanto en establecer correspondencias como en delimitar hasta dónde propagarlas.

12.4 Tres figuras de inferencia: transitividad, inversa y simetría

Más allá de la subsunción de clases, el perfil OWL 2 RL me permite tres constructos sobre propiedades que la Figura 12 ilustra en paralelo. El primero es la transitividad, que aplico a los prerrequisitos, donde la relación es transitiva por su naturaleza pedagógica, pues si los bucles `while` presuponen las condicionales y estas las expresiones booleanas, los bucles las presuponen indirectamente; declarándola `owl:TransitiveProperty`, el razonador materializa todas las dependencias indirectas y recupera de un paso la cadena completa de conocimientos previos. El segundo es la inversa, mediante `owl:inverseOf`, donde, declarada la propiedad inversa de los prerrequisitos, el razonador genera las tripletas en sentido opuesto sin duplicar afirmaciones, lo que me permite consultar qué conceptos dependen de uno dado y garantiza la consistencia entre ambas direcciones.

El tercero es la simetría, mediante `owl:SymmetricProperty`, para relaciones que no distinguen sentido. Aquí debo precisar un aspecto del modelado, pues la red conceptual tejida con `skos:broader`[8] es deliberadamente distinta de la jerarquía formal expresada con `rdfs:subClassOf`. La primera es una red de navegación al estilo de SKOS (W3C, 2009), para organización y exploración; la segunda, una jerarquía con semántica de subsunción que sí alimenta el razonamiento de tipos. Confundir ambas habría sobrecargado la jerarquía formal con relaciones que no son inclusiones de clases, o atribuido a SKOS una capacidad inferencial que no posee.

[8] diecinueve relaciones de generalidad, junto a diecinueve `skos:narrower` y dieciséis `skos:related`

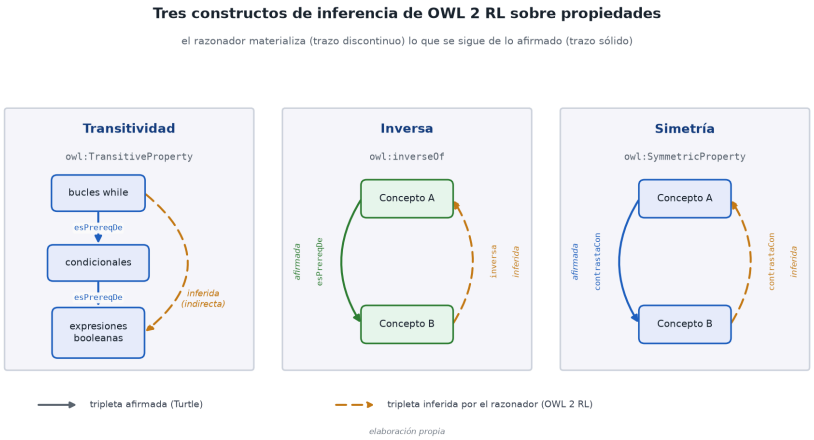

Figura 12. Los tres constructos de propiedad de OWL 2 RL: la transitividad encadena prerrequisitos indirectos (while ⟶ booleanas), la inversa genera la dirección opuesta y la simetría replica la relación; en cada caso el razonador materializa (trazo discontinuo) lo que se sigue de lo afirmado (trazo sólido).

12.5 Reproducibilidad del perfil OWL 2 RL entre owlrl y GraphDB

La elección de dos motores con el mismo perfil deja de ser comodidad para convertirse en validación. Como OWL 2 RL está especificado por el W3C (2012) mediante reglas bien definidas, esperaba que el grafo razonado fuese sustancialmente el mismo en ambos, y he aprovechado esa reproducibilidad como control de calidad, ya que una inferencia presente en `owlrl` y ausente en GraphDB, o a la inversa, sería una señal de alarma. En la práctica, el comportamiento sobre los constructos centrales se reproduce de manera consistente, lo que me da confianza razonable en que el salto de cero a 157 conceptos y el crecimiento de 1772 a 4786 enunciados no son artefactos de una herramienta sino consecuencias de la semántica OWL 2 RL.

Con todo, compartir nominalmente el perfil no garantiza coincidencia tripleta a tripleta, ya que ciertos detalles de implementación pueden producir diferencias menores en el conteo bruto sin afectar a las consecuencias relevantes; por eso he anclado la verificación no en la igualdad numérica exacta del cierre, sino en las inferencias que importan, los tipos que recupero y las cadenas de prerrequisitos que navego. Soy consciente, además, de que OWL 2 RL es deliberadamente limitado en expresividad[9] y de que el grafo sigue acotado por lo que decidí modelar; asumo estas limitaciones como fronteras conscientes del diseño.

[9] renuncia a razonamiento más potente a cambio de terminación y eficiencia

13 Validación de la integridad con SHACL

La expresividad de RDF y OWL arrastra un riesgo recurrente, ya que ningún mecanismo impide introducir datos mal formados. RDF admite cualquier afirmación, y nada impide que un nivel de dominio se exprese como cadena donde se esperaba un número, ni que una evaluación apunte a un concepto inexistente. OWL tampoco lo resuelve, porque obedece a la hipótesis del mundo abierto, donde sus restricciones se interpretan como premisas de las que inferir, no como condiciones que un dato deba satisfacer. Esa distinción justifica incorporar SHACL (Shapes Constraint Language), recomendación del consorcio desde 2017, que aporta un lenguaje declarativo para describir las formas de los datos y, bajo la hipótesis del mundo cerrado, señala las desviaciones como violaciones explícitas. La Figura 13 recoge las formas definidas y sus clases destino.

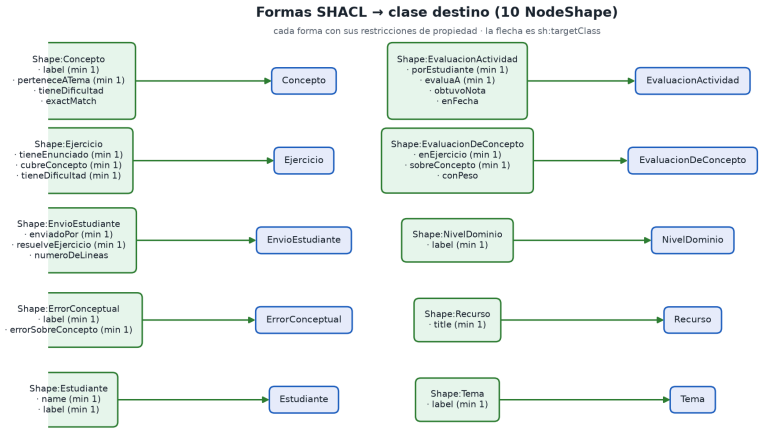

Figura 13. Las 10 formas SHACL (NodeShape) del EKG y sus clases destino.

13.1 Tres tareas complementarias sobre el grafo

Conviene distinguir la validación de las otras operaciones. La consulta recupera con SPARQL información ya presente o deducible, sin modificarlo. La inferencia lo enriquece materializando hechos implícitos de RDFS y OWL, pues en el caso canónico lo hace crecer desde 1772 enunciados afirmados hasta 4786 tras el cierre OWL 2 RL, y su efecto más visible es que la consulta «?x a pyedu:Concepto» pasa de 0 a 157 resultados.[10] La validación, en cambio, no recupera ni deduce, sino que contrasta el grafo de datos contra un grafo de formas y dictamina si es conforme. En mi configuración se ejecuta sobre el grafo ya enriquecido, de modo que las formas razonan sobre tipos deducidos y no solo afirmados.

13.2 Diseño del grafo de formas

He seguido el principio más natural en SHACL —una forma de nodo (sh:NodeShape) por cada clase relevante, vinculada a sus instancias mediante sh:targetClass. He elaborado diez: Concepto, Ejercicio, Estudiante, Tema, NivelDominio, ErrorConceptual, EnvioEstudiante, EvaluacionDeConcepto, EvaluacionActividad y Recurso. La elección de estas diez, y no de las veinte owl:Class del TBox (que conviven con 21 propiedades de objeto y 7 de datos), responde a un criterio de proporcionalidad. He reforzado la validación allí donde la integridad resulta crítica para la explotación con SPARQL y la retroalimentación, y dejé sin formas las clases auxiliares cuyas instancias se derivan de relaciones ya restringidas. Es una decisión revisable, pues una cobertura exhaustiva reforzaría la garantía a costa de un grafo más frágil. Dentro de cada forma he declarado formas de propiedad (sh:PropertyShape); las dos clases de evaluación reificada fueron las que más cuidado requirieron, por representar relaciones N-arias.

[10] los 157 conceptos propios deducidos por rdfs:subClassOf; las 30 entidades de Wikidata enlazadas con skos:exactMatch no cuentan, pues ese predicado correlaciona sin propagar el tipo, a diferencia de owl:sameAs

13.3 Restricciones reforzadas

La primera versión pecaba de laxitud, ya que un dato sintácticamente plausible pero semánticamente erróneo pasaba inadvertido. Reforcé entonces tres familias.

La primera es la restricción de tipo de nodo, donde he añadido sh:nodeKind sh:IRI a las propiedades de objeto (skos:broader, pyedu:evaluaConcepto, pyedu:tieneNivel), cuyos valores deben ser entidades identificadas y no literales, pues solo así esa entidad puede enlazarse con Wikidata o recuperarse en una consulta. La segunda atañe a los rangos numéricos, donde el nivel de dominio y las puntuaciones son valores acotados, y con sh:minInclusive y sh:maxInclusive rechazo los que caen fuera del intervalo, errores que ninguna comprobación de tipo detectaría porque un número fuera de rango sigue bien tipado; refuerzo además el tipo con sh:datatype.

La tercera familia, la más característica del enfoque de Web Semántica, valida mediante patrones los identificadores de Wikidata, pues como el EKG se enlaza con 30 relaciones skos:exactMatch verificadas, he empleado sh:pattern con una expresión regular que exige la estructura canónica de las entidades. La comprobación es deliberadamente conservadora, ya que no verifica que la entidad exista realmente,[11] pero impide los errores tipográficos comunes. SHACL valida la forma del enlace, no su veracidad referencial.

A estas familias he añadido restricciones de cardinalidad con sh:minCount para las propiedades obligatorias —que todo concepto tenga al menos una etiqueta, que toda evaluación conecte con el estudiante y con el elemento evaluado— y sh:maxCount para las funcionales. La combinación convierte el grafo de formas en una especificación legible del EKG correcto, que sirve como contrato entre su construcción y su explotación.

13.4 Validación con pyshacl e inferencia RDFS

He empleado pyshacl, la implementación de referencia de SHACL en Python, que se integra con rdflib y permite indicar un razonamiento previo (inferencia RDFS) antes de contrastar el grafo contra las formas. Esa opción resultó decisiva, pues sin ella las formas dirigidas por sh:targetClass solo alcanzarían a los individuos cuyo rdf:type figura afirmado literalmente, y dejarían fuera a las entidades cuya clase se deriva de rdfs:subClassOf —el salto de 0 a 157 conceptos—, lo que produciría una falsa sensación de conformidad. pyshacl construye el grafo enriquecido, lo somete a las formas y descarta luego los hechos materializados, sin tocar el grafo en disco. La Figura 14 resume ese flujo. La inferencia RDFS precede al contraste y el dictamen es la conformidad o una lista de violaciones que identifican el nodo, la forma y la restricción incumplida.

[11] eso requeriría una consulta remota que excede el alcance de la validación local

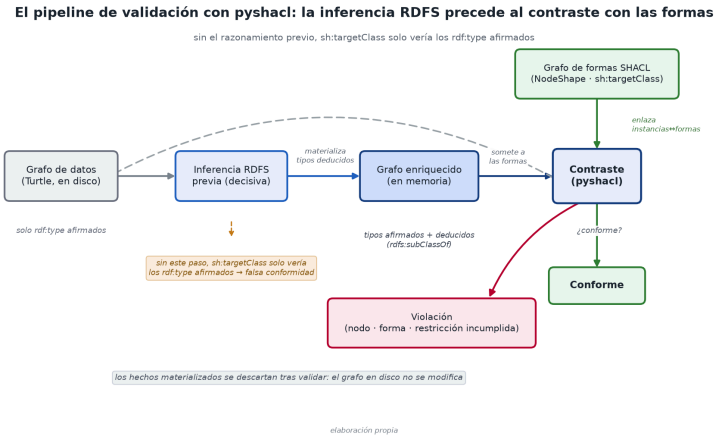

Figura 14. El pipeline de validación con pyshacl: la inferencia RDFS precede al contraste con el grafo de formas, de modo que sh:targetClass alcanza también las clases deducidas; el dictamen es conforme o una violación (nodo, forma, restricción) y el grafo en disco no se modifica.

El grafo canónico es conforme, con cero violaciones (SHACL 0/6). Su alcance, eso sí, es limitado, pues la conformidad garantiza que el grafo respeta las formas que he sabido especificar, no que esté libre de cualquier error concebible, ya que una validación es tan exhaustiva como su grafo de formas, y el mío cubre las restricciones que juzgué críticas, no todas las situaciones. Aun así, ofrece una garantía verificable, pues ningún enlace de objeto aloja un literal donde debería ir un IRI, ningún valor numérico se sale de su rango y todos los enlaces a Wikidata respetan el patrón.

Una validación que siempre dice «conforme» es indistinguible de una que no comprueba nada, de modo que preparé un fichero con violaciones deliberadas que activan cada familia, y reporta exactamente seis violaciones (SHACL 0/6): un nivel de dominio fuera del intervalo (sh:maxInclusive); un literal donde debía ir una entidad (sh:nodeKind sh:IRI); un skos:exactMatch malformado (sh:pattern); y la omisión de una propiedad obligatoria (sh:minCount). Cada una identifica el nodo, la forma y la restricción incumplida con un mensaje legible, y el contraste entre el canónico, con cero violaciones, y el defectuoso, con seis, evidencia que la validación discrimina de modo reproducible.

13.5 Reflexión sobre el papel de SHACL en el flujo de trabajo

La incorporación de SHACL ha convertido la confianza implícita en que los datos generados por los guiones eran correctos en una comprobación explícita y automatizable que ejecuto tras cada modificación sustancial del EKG. Ocupa un lugar análogo al de las pruebas automáticas en el software, ya que no demuestra la ausencia de todos los errores, pero impide la regresión silenciosa de aquellos que ya he sabido caracterizar. En un sistema que aspira a integrar el grafo con modelos de lenguaje mediante recuperación aumentada, esa garantía es un requisito, pues cada afirmación errónea que sobreviviese se propagaría al contexto que recibe el modelo y al juicio formativo que se ofrece al estudiante.

14 Consultas SPARQL y explotación

Modelado el grafo en RDF/OWL, validado con SHACL y enriquecido con el cierre de OWL 2 RL, queda la pregunta que da sentido al edificio, ¿para qué sirve un grafo de conocimiento si no se interroga? El conocimiento en tripletas solo se vuelve valor cuando alguien —en mi caso, un modelo de lenguaje— formula una pregunta y recibe una respuesta trazable. SPARQL es el lenguaje de esa interrogación y el puente entre las dos mitades del trabajo, la declarativa, que representa la programación en Python con rigor semántico, y la generativa, que produce retroalimentación con un LLM. Describo cómo he explotado el Grafo de Conocimiento Educativo (EKG) con SPARQL 1.1 (W3C, 2013), cuya unidad no es la tabla sino el patrón de grafo (Hogan et al., 2021). Trabajé con dos motores, `rdflib` en memoria para desarrollo y GraphDB (Ontotext, s.f.) con OWL2-RL en despliegue. Así el cierre queda disponible al consultar.

14.1 SELECT y CONSTRUCT

`SELECT` devuelve una tabla, herramienta de la recuperación y la analítica. `CONSTRUCT` devuelve un grafo nuevo a partir de un patrón hallado, y esa diferencia es la clave de la integración con el LLM, porque un modelo no consume tablas sino contexto, y un subgrafo es justo el contexto que conviene inyectar en el *prompt* de un RAG (Lewis et al., 2020; Edge et al., 2024).

La consulta más elemental recupera todos los conceptos del dominio (consulta 02) pidiendo los individuos de tipo `pyedu:Concepto`, y encierra una de las lecciones más instructivas del proyecto.[12] Sobre el grafo afirmado (1772 tripletas en Turtle, sin cierre) devuelve cero, pues ningún individuo se declara literalmente como `rdf:type pyedu:Concepto`; sobre el cierre (4786 tripletas) devuelve 157, los conceptos propios deducidos por `rdfs:subClassOf` (0⟶157), pues los he modelado como subclases de `pyedu:Concepto` y la regla de RDFS propaga el tipo por la jerarquía aunque yo nunca lo afirmara. Las 30 entidades de Wikidata enlazadas no engrosan el recuento, pues se vinculan con `skos:exactMatch` y no `owl:sameAs`, para que el razonador no fusione el concepto propio con la entidad externa ni le propague el tipo; el enlace declara correspondencia, no identidad. De ahí que consulte siempre la base materializada de GraphDB.

Para caracterizar el dominio, una consulta agrupa los conceptos por tema con `GROUP BY`, `COUNT` y `ORDER BY DESC`, y da un mapa de los 16 temas con el que detecté desequilibrios de cobertura antes de la evaluación. Importa no confundir la jerarquía formal de `rdfs:subClassOf`, con consecuencias inferenciales, y la red asociativa de `skos:broader` (W3C, 2009), con 19 relaciones (y sus 19 `skos:narrower` recíprocas y 16 `skos:related`), que expresa proximidad sin la carga de la subsunción. Las he mantenido separadas.

14.2 Errores con procedencia, la reificación

El núcleo pedagógico del grafo no son los conceptos sino los errores, las equivocaciones típicas de un estudiante de Python, cada una asociada a un concepto mal comprendido y a una fuente que documenta esa categorización, de modo que toda afirmación del sistema sea rastreable (Hattie & Timperley, 2007; Keuning et al., 2019). Para anotar la procedencia de la propia tripleta «el error E afecta al concepto C» no bastan las propiedades ordinarias de RDF; lo resuelve la reificación, que trata una afirmación como recurso de primera clase (W3C, 2014a). He empleado `rdf:Statement`: materializo cada enunciado con sus componentes (`rdf:subject`, `rdf:predicate`, `rdf:object`) y cuelgo la anotación `dcterms:source` de Dublin Core. El coste es la verbosidad —una arista requiere cuatro o cinco tripletas—; el beneficio es la trazabilidad. Cada afirmación viaja con su justificación al subgrafo de evidencia, y esa fue la propiedad más valiosa del sistema. El grafo emplea además reificación N-aria mediante clases intermedias (`EvaluacionDeConcepto` y `EvaluacionActividad`, esta con diez instancias) para relaciones que vinculan más de dos entidades.

[12] el esquema vive bajo `pyedu:` (TBox) y las instancias bajo `pyr:` (ABox)

14.3 Prerrequisitos transitivos y CONSTRUCT que aplana el cierre

La relación más rica es la de prerrequisito, inherentemente transitiva, pues si A es prerrequisito de B y B de C, entonces A lo es de C. Obtener su cierre admite dos caminos que conviene contrastar, sin ganador absoluto. Declarar la propiedad como `owl:TransitiveProperty` deja que el razonador OWL 2 RL calcule el cierre en la materialización; es declarativo y reutilizable, pero arrastra coste de materialización y rigidez. Un *property path* de SPARQL 1.1 lo expresa en la consulta, donde `+` significa «uno o más pasos» (y `*`, «cero o más»), de modo que `pyedu:tienePrerrequisito+` recupera los inmediatos y todos los alcanzables por la cadena; es más ágil, a costa de esconder la transitividad en la consulta. He preferido declararla en el esquema, pero conservo las consultas con *property path* para recorridos exploratorios y para el caso, importante en producción, de consultar un grafo no materializado. Los prerrequisitos transitivos de la *búsqueda binaria* recuperados con un camino de propiedad (consulta 03) son cinco conceptos. La Figura 15 muestra ese subgrafo de evidencia.

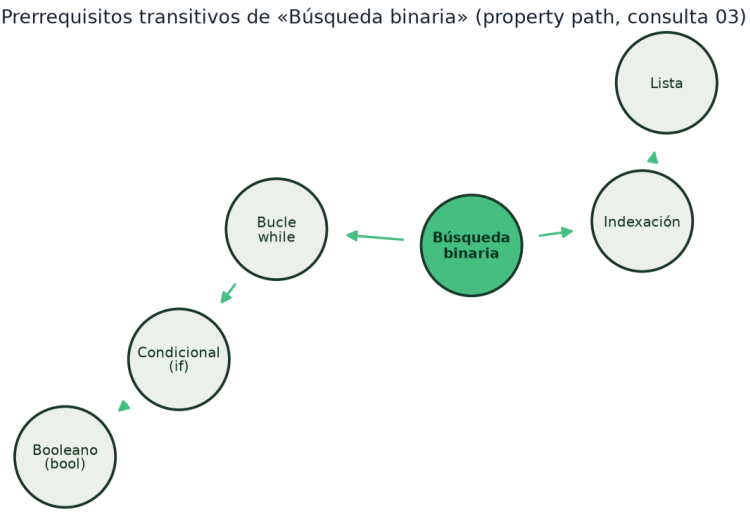

Figura 15. Prerrequisitos transitivos de «Búsqueda binaria» recuperados con un property path (consulta 03), 5 conceptos.

A ella se añade la consulta de ruta entre conceptos (consulta 04). Ahora bien, para alimentar al LLM necesito esa maraña de prerrequisitos plana. Una consulta `CONSTRUCT` combinada con `tienePrerrequisito+` aplana el cierre (consulta 06), pues el `+` encuentra los prerrequisitos a cualquier profundidad y `CONSTRUCT` emite para cada uno una tripleta directa hacia la raíz y expande el grafo de 340 a 356 tripletas en una estrella sin niveles intermedios. Es ingeniería del contexto, porque un modelo aprovecha mejor hechos planos que una jerarquía profunda que tendría que recorrer.

14.4 La consulta-caso, del envío al subgrafo de evidencia

Todas las piezas convergen en una sola consulta, corazón de la explotación, que partiendo de un envío de un estudiante extrae el subgrafo de evidencia recorriendo del envío al error que ejemplifica, de ahí al concepto mal comprendido, a su procedencia bibliográfica y al árbol aplanado de prerrequisitos; en suma, *este envío falla aquí, lo cual indica que este concepto no está dominado, según esta fuente, y para dominarlo hay que comprender antes estos otros*. La formulo como `CONSTRUCT` porque su destino es un *prompt*, con dos `OPTIONAL`, donde la primera atraviesa la estructura `rdf:Statement` para extraer la fuente, opcional porque no todo error tiene procedencia documentada y no quiero que su ausencia borre el resto de la evidencia; la segunda, con `+`, despliega el cierre aplanado. Un `FILTER` ancla la consulta al envío concreto.

El subgrafo que emerge es compacto, trazable y plano; serializado y verbalizado, se inserta en el contexto del modelo, que no inventa el concepto ni la cadena sino que los recibe del grafo. En el *benchmark*

de 50 casos *held-out*, la ventaja del sistema híbrido (Sistema D), con acierto de categoría 0.76 y de concepto 0.54, se origina en la calidad de este subgrafo. La explotación no se agota en el grafo local. Una consulta federada con `SERVICE` contra el *endpoint* de Wikidata (consulta 08), apoyada en los 30 `skos:exactMatch`, recupera 20 filas externas, lo que muestra que el enlazado de datos abierto enriquece el contexto con conocimiento que no he tenido que modelar.

14.5 SPARQL como puente entre representación y generación

Aquí reside, a mi juicio, la contribución conceptual del trabajo. La representación simbólica (RDF, OWL, SHACL) es rigurosa y verificable, pero rígida; la generación con LLM es flexible, pero opaca y propensa a afirmar sin fundamento. SPARQL los hace colaborar, ya que opera sobre el grafo materializado y `CONSTRUCT` produce el subgrafo de evidencia trazable que el modelo consume como contexto. Por eso he cuidado tanto las consultas, porque la calidad de la retroalimentación está acotada, en su límite superior, por la de la evidencia recuperada, y un GraphRAG no es más fiable que su recuperador de consultas SPARQL sobre un grafo bien modelado (Edge et al., 2024; Abu-Salih & Alotaibi, 2024).

Reconozco las cautelas. Las consultas están afinadas para la estructura concreta de este grafo y los patrones de error modelados, de modo que su generalización a otros dominios exigiría reescritura; el aplanamiento descarta información de profundidad que en algunos escenarios podría ser útil; y el rendimiento de los *property path* sobre grafos no materializados de gran tamaño no lo he sometido a una prueba de carga rigurosa, aunque en el grafo canónico, de dimensiones modestas, el coste no ha sido apremiante. Con esas reservas, SPARQL convierte un grafo estático en un proveedor activo de contexto fundamentado, y es esa función de puente lo que lo hace indispensable en los capítulos siguientes.

14.6 Complemento metodológico: grafos multinaturaleza y explotación SPARQL de la ontología pedagógica (reconstrucción v2/v4 — Linaje B)

Aviso de separación de experimentos. Todo lo anterior pertenece al **proyecto canónico** (EKG de 157 conceptos, sistemas A/B/C/D, benchmark sintético n=50 y banco real Dublin). Esta sección documenta, como **complemento metodológico**, una **reconstrucción aislada del pipeline** (Linaje B) realizada para estudiar el componente de recuperación; sus grafos y cifras **no se mezclan con las del proyecto canónico ni las sustituyen.**

Sobre el corpus combinado de la reconstrucción (`corpus_python.json`, **73 documentos / 2.272.919** caracteres**: 8 del curso de Web Semántica + 65 de libros de Python, incluidas 21 secciones de** *Think Python*) se reconstruyeron **seis grafos de naturaleza distinta**, y cada uno aporta una vista complementaria para la augmentación: (a) **ontología/conocimiento** (RDF/OWL + inferencia RDFS + validación SHACL); (b) **similitud semántica** por *k*-NN coseno sobre *embeddings* (559⟶1.400 nodos al pasar del corpus de Web Semántica al de Python); (c) **dependencias de código** por AST (23⟶43 nodos); (d) **co-ocurrencia** de términos por ventana de frase (355⟶2.316 aristas); (e) **jerarquía** de conceptos (taxonomía); y (f) **relaciones entidad** sujeto-predicado-objeto (96⟶540 aristas; 10.983 triples / 23.604 entidades). Los grafos (a) y (e) son una semilla de dominio

redactada por IA y reutilizada, no derivada del corpus, y así se etiqueta.

La pieza más relevante para esta asignatura es la **explotación aplicada del grafo por SPARQL** en la vía **v4 (RAG semántico/ontológico)**. Se construyó una ontología pedagógica específica de Python, `a_ontologia_python.ttl` (namespace `ex: <http://tfm.uned.es/ekg-python#>`, **94 triples**), con tres clases (`ex:Concepto`, `ex:ConceptoError` ⊑ `ex:Concepto`, `ex:CategoriaError`) y **19 individuos** (10 conceptos + 5 *misconceptions* + 4 categorías), cada uno anotado con una propiedad `ex:reglaPedagogica` (`owl:AnnotationProperty`), una pista de tutoría de 1–3 frases. En tiempo de inferencia, en lugar de recuperar pasajes de texto por similitud, el sistema ejecuta una **consulta** SPARQL determinista **que, con una precedencia explícita (misconception → categoría → concepto),**
recupera la `ex:reglaPedagogica` aplicable y la inyecta como guía. Por ejemplo, para el error «modificar una lista mientras se itera»:

PREFIX ex: <http://tfm.uned.es/ekg-python#>
SELECT ?regla WHERE { ex:err_mod_durante_iteracion ex:reglaPedagogica ?regla . }

o por etiqueta exacta del concepto (cuando no hay *misconception* mapeada):

PREFIX ex: <http://tfm.uned.es/ekg-python#>
PREFIX rdfs: <http://www.w3.org/2000/01/rdf-schema#>
SELECT ?regla WHERE { ?c a ex:Concepto ; rdfs:label «Bucle for»@es ; ex:reglaPedagogica ?regla . }

Es el puente representación↔generación llevado a su forma más quirúrgica, una sola regla recuperada por SPARQL en lugar de un volumen de prosa. En la evaluación de esa reconstrucción (Linaje B, n=10 *held-out*, juez local), esta vía ontológica fue la **única** que superó al modelo base sin recuperación, mientras que la recuperación por pasajes no lo lograba; el detalle cuantitativo y sus salvedades (entre ellas dos casos en que la regla no mejoró a la base) se reportan en la memoria del TFM, no aquí, para no mezclar los dos experimentos.

15 Enlazado de datos y reutilización de vocabularios

Un grafo de conocimiento solo merece tal nombre si dialoga con el resto de la Web de Datos; una ontología cerrada sobre sí misma sería una isla léxica que nadie más podría interpretar. Por eso, consolidado el modelado ontológico, he dedicado una fase a dos tareas que la literatura agrupa bajo *linked data* —reutilizar vocabularios consolidados en lugar de acuñar términos propios y enlazar las entidades del grafo con la nube de datos abiertos—. Ambas tienen consecuencias en interoperabilidad, riqueza semántica y verificabilidad. La Figura 16 resume los vocabularios reutilizados y el enlace a Wikidata.

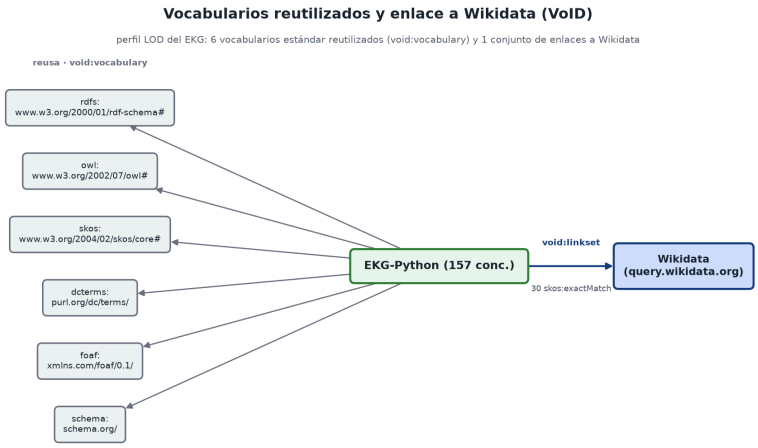

Figura 16. Vocabularios reutilizados (SKOS, Dublin Core, FOAF, schema.org) y enlace a Wikidata, según void.ttl.

15.1 El principio de reutilizar antes de acuñar

El material docente insiste, con firmeza que comparto, en una máxima, que antes de inventar un término conviene comprobar si ya existe uno consolidado que exprese lo mismo. El término reutilizado arrastra una semántica compartida y herramientas que ya saben interpretarla; la proliferación de vocabularios redundantes, en cambio, fragmenta la Web de Datos, pues si cada grafo describe «el autor de un recurso» con una propiedad distinta, ninguna consulta federada podrá recorrerlos de manera uniforme.

He aplicado este principio reservando el espacio propio (`pyedu` para el esquema, `pyr` para las instancias) a lo específico de la programación en Python que no encuentra acomodo en ningún vocabulario público; todo lo demás se toma de cuatro vocabularios. De SKOS (W3C, 2009) tomo `skos:Concept`, porque, al declarar `pyedu:Concepto` su subclase, los 157 conceptos propios quedan reconocidos como instancias del vocabulario estándar, y así cualquier herramienta que procese SKOS recorre el grafo sin conocer una línea de mi ontología. Dublin Core (`dcterms`) lo empleo para registrar el origen de las afirmaciones reificadas mediante `dcterms:source`, de modo que el conocimiento no se presenta como hecho desnudo sino como enunciado con autoría documentada. FOAF describe a estudiantes y autores y evita una propiedad propia de «nombre» cuando `foaf:name` ya cumple esa función. Y de schema.org tomo los términos de recursos de aprendizaje (`LearningResource` y afines) por su amplísima adopción y porque los datos así anotados resultan legibles por la infraestructura de indexación de la Web abierta.

No se trata de agotar la expresividad de estos vocabularios, sino de alcanzar un compromiso. Allí donde existía un término estándar adecuado, lo he preferido a acuñar uno propio. Hay zonas del dominio donde no he encontrado un vocabulario público que las capturase con precisión, y es ahí donde `pyedu` aporta valor.[13]

[13] los prerrequisitos entre conceptos, el contraste entre nociones que se confunden, la asociación de errores con conceptos

15.2 La malla conceptual SKOS

Entre las propiedades de SKOS, `skos:broader` ha resultado especialmente fértil al expresar que un concepto es más general que otro. He tejido con ella 19 relaciones que enlazan cada noción con las que la subsumen. Así, la comprensión de listas se declara `skos:broader` respecto del bucle `for`, pues comprender este es condición de aquella. A las 19 `skos:broader` se añaden sus 19 inversas `skos:narrower` y 16 `skos:related`, hasta cincuenta y cuatro enlaces que atraviesan el grafo en paralelo a su jerarquía de clases.

Esta red es distinta de la jerarquía `rdfs:subClassOf` de tipos, y no por matiz pedante, pues `rdfs:subClassOf` afirma que toda instancia de una clase lo es de otra y arrastra la inferencia de tipos que el razonador propaga de forma monótona; `skos:broader`, en cambio, opera sobre conceptos entendidos como nodos de un esquema, no como clases con extensión, y expresa una generalización temática que no implica subsunción de instancias. Mantenerlas separadas permite que la jerarquía de tipos alimente la inferencia OWL-RL mientras la malla conceptual ofrece un grafo de navegación temática valioso para la recuperación de subgrafos que más adelante alimenta al modelo de lenguaje, lo que permite ascender desde un error hacia las nociones más generales que el estudiante debería dominar.

15.3 Enlazado a Wikidata con `skos:exactMatch`

La reutilización resuelve la interoperabilidad del esquema, pero no enlaza las entidades con la nube de datos abiertos. Para ello he empleado `skos:exactMatch`, que afirma correspondencia de significado sin fundir los recursos, pues declarar que `pyr:BucleFor` es `skos:exactMatch` de su entidad en Wikidata equivale a afirmar que ambos IRI designan el mismo concepto a efectos de enlazado, sin la identidad lógica de `owl:sameAs`. Lo preferí deliberadamente para no forzar la fusión de individuos del perfil OWL 2 RL, que trataría mi concepto y la entidad de Wikidata como un único recurso y propagaría entre ambos cuanto se predicara de cualquiera. Tal fusión es incorrecta cuando solo se quiere tender un puente entre dos descripciones de la misma noción. He establecido 30 de estos enlaces. La Figura 17 muestra una parte de esos enlaces verificados.

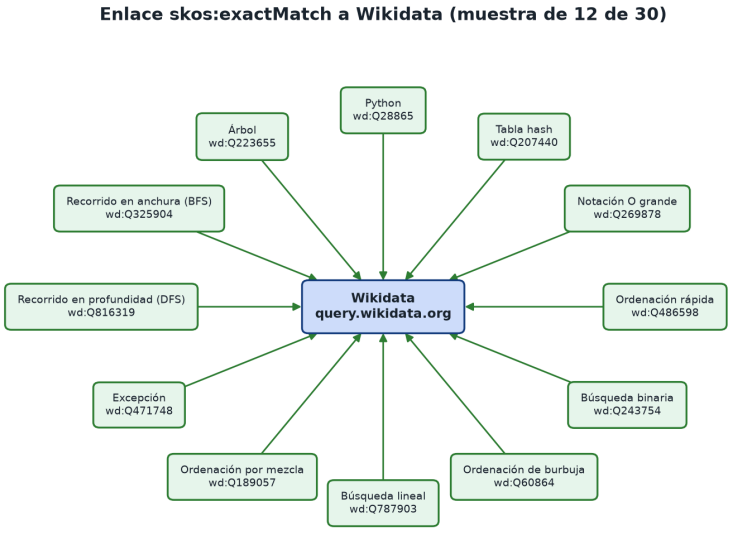

Figura 17. Enlace del EKG a Wikidata mediante skos:exactMatch (muestra de los 30 enlaces verificados).

Como `skos:exactMatch` no es relación de identidad, el razonador no propaga las afirmaciones de tipo a través de ella, de modo que cuando el cierre OWL 2 RL deduce que los conceptos propios son instancias de `pyedu:Concepto`, esa pertenencia no se transmite a las treinta entidades de Wikidata, que quedan vinculadas sin ser reclasificadas. Esto explica una de las cifras de la inferencia. La consulta que recupera

los individuos de `pyedu:Concepto` (consulta 02) devuelve 0 resultados sin razonamiento y 157 con él, sin que las entidades de Wikidata engrosen esa cifra.

15.4 Verificar cada QID contra la fuente

Debo detenerme en la cautela que considero la lección más valiosa de esta fase. Cada entidad de Wikidata se identifica con un QID, y la tentación, cuando uno cree conocer el dominio, es escribir el que recuerda. Sucumbí a ella en un primer borrador, pues el caso más memorable fue un QID que yo recordaba como el de «tabla hash» y que, consultado contra el servicio de Wikidata, designaba en realidad una «victoria pírrica»; un identificador que creía referido a una estructura de datos apuntaba a una locución histórica sin relación con la informática.

Lejos de anécdota menor, esto confirmó la necesidad de un procedimiento de verificación sistemático. En lugar de afirmar ningún `skos:exactMatch` a partir de un identificador recordado, he verificado los 30 enlaces de forma programática. Para cada QID candidato, una rutina consulta el servicio, recupera etiqueta y descripción y comprueba que corresponden al concepto previsto antes de incorporar el enlace; en caso contrario se descarta o corrige. En el enlazado de datos lo que se afirma como idéntico debe haberse comprobado como idéntico, porque la alternativa es propagar errores a través de toda la inferencia y contaminar silenciosamente cada consulta posterior.

15.5 Consultas federadas

El enlazado abre, además, la consulta federada. SPARQL 1.1 incorpora la cláusula `SERVICE`, que delega parte de la evaluación en un punto de acceso remoto y combina ambos resultados de forma transparente (W3C, 2013); así, una consulta puede partir de un concepto del grafo, seguir su `skos:exactMatch` hacia Wikidata y recuperar información que el grafo local no contiene sin duplicar esos datos.[14] La consulta federada 08 devuelve 20 filas; en lugar de copiar una porción de Wikidata, el grafo enlaza y delega los detalles en la fuente autorizada.

La federación, no obstante, tiene límites. Depende de la disponibilidad y la latencia del punto de acceso remoto; el servicio público de Wikidata está sujeto a cuotas y tiempos variables, de modo que una consulta compleja puede resultar lenta o fallar si está saturado. Por eso la federación se reserva para enriquecimientos puntuales y opcionales, no para el camino crítico del diagnóstico, que se apoya en el grafo local cargado en GraphDB; es una puerta abierta al resto de la nube de datos más que una dependencia operativa.

15.6 VoID y datos FAIR

Para que el grafo sea localizable y reutilizable por terceros, lo he acompañado de una descripción VoID (*Vocabulary of Interlinked Datasets*) en `void.ttl`. Este metadato declara el conjunto como `void:Dataset` con título, autoría y licencia (Creative Commons CC BY 4.0), documenta los vocabularios reutilizados (RDFS, OWL, SKOS, Dublin Core, FOAF y schema.org) y una partición por clases que registra, entre otras, las 157 instancias de concepto y las 16 de error conceptual. El enlazado a Wikidata se modela como un `void:Linkset` explícito, con `void:linkPredicate skos:exactMatch` y `void:triples 30`, y deja constancia formal de que la conexión se establece por equivalencia conceptual y no por identidad estricta.

Esta descripción materializa los principios FAIR —localizable por su IRI estable y sus metadatos estandarizados; accesible por publicarse en Turtle, formato abierto y dereferenciable; interoperable por reutilizar vocabularios consolidados; y reutilizable por declarar una licencia explícita y registrar la procedencia de sus afirmaciones. Hacer FAIR el grafo es la culminación natural del enlazado, pues un grafo bien modelado pero cerrado sobre sí mismo desaprovecharía buena parte de su valor.

[14] fecha de aparición de una técnica, autor, relaciones

15.7 Balance de la fase

Esta fase ha materializado dos virtudes que distinguen un grafo de conocimiento de una base de datos —la interoperabilidad semántica, mediante la reutilización disciplinada de SKOS, Dublin Core, FOAF y schema.org, y la conexión con la nube de datos abiertos, mediante 30 enlaces `skos:exactMatch` a Wikidata verificados uno por uno. Y la lección transversal —comprobar cada identidad contra la fuente en lugar de fiarla a la memoria— queda incorporada como principio de método, porque en el enlazado de datos la diferencia entre afirmar y verificar separa un grafo fiable de uno que propaga errores con la eficiencia implacable de la inferencia.

16 Integración del grafo con modelos de lenguaje

Consolidado el grafo de conocimiento educativo (EKG) —los 157 conceptos propios, las 1772 afirmaciones que crecen hasta 4786 tras el cierre OWL 2 RL, la conformidad SHACL sin violaciones y los 30 enlaces `skos:exactMatch` a Wikidata— se planteaba la pregunta difícil de cómo lograr que ese conocimiento estructurado influya en la respuesta que un modelo de lenguaje ofrece a un estudiante. Un grafo no genera por sí solo retroalimentación formativa, y un modelo no conoce la ontología que he diseñado ni distingue, sin ayuda, un error de mutabilidad de listas de uno de alcance de variables. El puente es la generación aumentada por recuperación (RAG), que sobre grafos da lugar a GraphRAG. Este capítulo recorre la recuperación, el *fine tuning* QLoRA del Sistema B y la evaluación A/B/C/D, con una cautela transversal, pues es un banco de pruebas para una pregunta de investigación, no un sistema de producción endurecido.

16.1 El módulo de recuperación

El módulo de recuperación decide qué fragmentos del conocimiento acompañan a la consulta antes de llegar al modelo. La búsqueda por similitud entre embeddings, habitual sobre texto, resulta insuficiente cuando el conocimiento es un grafo, porque ignora lo que lo hace valioso —las relaciones explícitas, la subsunción, la procedencia—. Combiné, por ello, dos señales heterogéneas y expandí el vecindario mediante consultas estructurales. La primera es la similitud semántica con los 157 conceptos, calculada con embeddings mediante `nomic-embed-text` ejecutado localmente vía Ollama. Ahora bien, los embeddings capturan mal la dimensión computacional, porque el estudiante rara vez describe su error en términos conceptuales y entrega código. Por eso añadí una segunda señal derivada del análisis sintáctico. Mediante el módulo `ast` el código se parsea para extraer marcadores que correlacionan con conceptos —`ListComp` con la comprensión de listas, `global` con el alcance, `append` con la mutabilidad—, en una inspección determinista de la estructura, en vez de pedir al modelo que adivine. Sus límites son evidentes. El mapeo entre patrones del AST y conceptos lo construí manualmente, es incompleto y está sesgado hacia los errores que anticipé al diseñar la ontología. Un patrón no visto deja la recuperación a expensas solo de los embeddings. No reclamo exhaustividad, sino que la fusión de ambas señales recupera con frecuencia los conceptos pertinentes.

Identificados esos conceptos semilla, el paso que distingue a GraphRAG es la expansión, pues en lugar de nodos aislados reconstruimos un subgrafo conexo con consultas SPARQL que recorren hasta una profundidad acotada la subsunción `rdfs:subClassOf`, la red `skos:broader`, las evaluaciones reificadas y los enlaces de procedencia que asocian a cada error su fuente bibliográfica. La expansión debe operar siempre sobre el grafo cerrado, dado que `?x a pyedu:Concepto` devuelve 0 resultados sin inferencia y 157 con ella; los 30 enlaces a Wikidata no engrosan esa cifra, porque se vinculan con `skos:exactMatch` y no con `owl:sameAs`, decisión deliberada para no imponer la fusión lógica de individuos que el perfil OWL-RL acarrearía.

Recuperado el subgrafo, la serialización no es un volcado del Turtle sino una traducción a prosa estructurada, fiel a las relaciones pero legible; opción discutible que sacrifica precisión formal a cambio de legibilidad. Con ese contexto se ensambla el prompt aumentado, formado por una instrucción de sistema que fija el rol de tutor formativo,[15] el contexto recuperado como referencia validada, el material del estudiante y una instrucción de salida que exige identificar el error, nombrar el concepto y anclar el diagnóstico en un concepto identificable del grafo. Esa exigencia de trazabilidad es, a mi juicio, una de las aportaciones más netas de inyectar el grafo, porque vuelve verificable la retroalimentación. La inferencia se realiza localmente con Ollama, cuya ventana de contexto finita es, junto a la pertinencia semántica, una razón de acotar la profundidad.

[15] en línea con Hattie & Timperley y Keuning et al., que favorecen la retroalimentación sobre el proceso

16.2 El afinado del Sistema B mediante QLoRA

Frente a la recuperación, que inyecta conocimiento en inferencia, el *fine tuning* lo incorpora directamente en los pesos. Como base opté por `Qwen2.5-Coder-7B-Instruct`, especializada en código y ya alineada por instrucciones, porque la tarea no es generar código sino analizar fragmentos para diagnosticar errores conceptuales; los siete mil millones de parámetros equilibran capacidad y viabilidad en un equipo de sobremesa. Afinar todos los pesos sería inviable, así que recurrí a QLoRA, que cuantiza el modelo base a cuatro bits en formato `nf4` y entrena solo dos matrices pequeñas mediante LoRA, con `r=16` y `alpha=32`, en una RTX 5090 de 32 GB y arquitectura Blackwell.

No existía un conjunto preparado, así que lo construí con ejemplos sintéticos mediante plantillas ancladas al EKG. Ahora bien, la generación por plantillas arrastra una contrapartida, ya que los ejemplos comparten regularidades superficiales que el modelo podría aprender en lugar de la competencia genuina. Por eso la decisión más importante fue la partición held-out por esqueleto —la estructura subyacente de la plantilla despojada de instanciaciones—. Ningún esqueleto de validación aparece en entrenamiento, de modo que la validación mide generalización real y no reconocimiento de patrones vistos.

Esa cautela demostró su valor de inmediato. La primera tanda (versión v2, sin regularización) exhibió un sobreajuste nítido. La pérdida de entrenamiento se desplomaba hacia 0,01 mientras la held-out crecía época tras época; sin la partición por esqueletos, el problema habría pasado inadvertido. La versión v3 añade NEFTune con `alpha=5` y `lora_dropout=0,1`; el efecto fue concluyente; la pérdida held-out invirtió su tendencia, de 1,051 a 1,028, con una precisión de token de 0,842. La comparación entre v2 y v3 constituye una pequeña pero genuina ablación. La Figura 18 muestra esas curvas de pérdida.

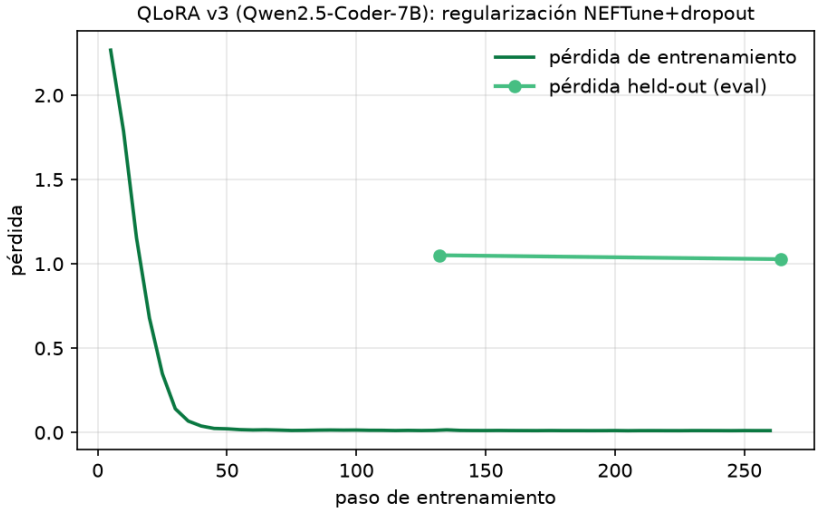

Figura 18. Curvas de pérdida de entrenamiento y held-out (QLoRA v3), donde la regularización invierte el sobreajuste, 1,051⟶1,028.

Estas cifras piden sus cautelas. El conjunto es sintético y derivado de plantillas, de modo que generalizar a esqueletos nuevos del mismo procedimiento no equivale a generalizar a la variabilidad genuina del código de estudiantes auténticos; la muestra de validación, además, es pequeña, y las cifras deben leerse como indicios consistentes antes que como estimaciones de precisión estadística.

16.3 La evaluación A/B/C/D

Diseñé un experimento factorial que cruza dos decisiones binarias e independientes —si el modelo está afinado y si recibe el subgrafo— y genera cuatro sistemas, donde A es la línea base austera (generalista `llama3.1:8b`, sin afinar y sin grafo); B, el *fine tuning* sin grafo, que aísla el fine-tuning; C, el modelo base con GraphRAG, que aísla la recuperación; y D, el híbrido. Comparar A con B y con C mide cada intervención en aislamiento; cotejarlas con D indaga si se suman o se solapan.

La evaluación se realiza sobre 50 casos held-out, de esqueletos que el Sistema B nunca vio, y combina dos familias de métricas. La primera son métricas objetivas ancladas a una verdad de referencia —el acierto de categoría y el de concepto, binarios— (Figura 19). Aunque la tabla muestre cuatro objetivas, solo dos lo son de verdad, porque identificación y trazabilidad son recodificaciones deterministas de aquellas reescaladas a 1–5. La segunda familia atiende a dimensiones graduales —divulgativa, técnica y de la sugerencia— puntuadas en escala 1–5 por un juez LLM (`qwen2.5:32b`) (Figura 20), recurso que introduzco con plena conciencia de sus límites.

Métrica	Sistema A	Sistema B	Sistema C	Sistema D
Acierto de categoría	0,26	0,70	0,36	0,76
Acierto de concepto	0,18	0,50	0,48	0,54
Identificación (1–5)	2,04	3,80	2,44	4,04
Trazabilidad (1–5)	1,72	3,00	2,92	3,16

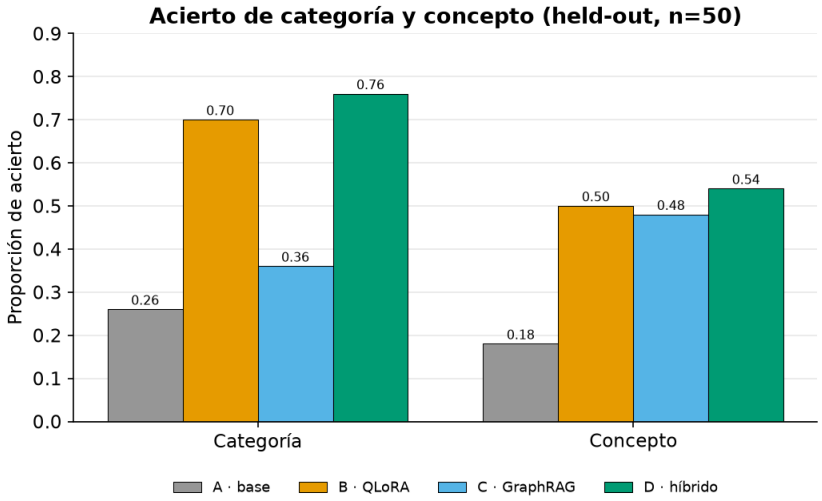

Figura 19. Métricas objetivas del benchmark n=50 (categoría y concepto) por sistema.

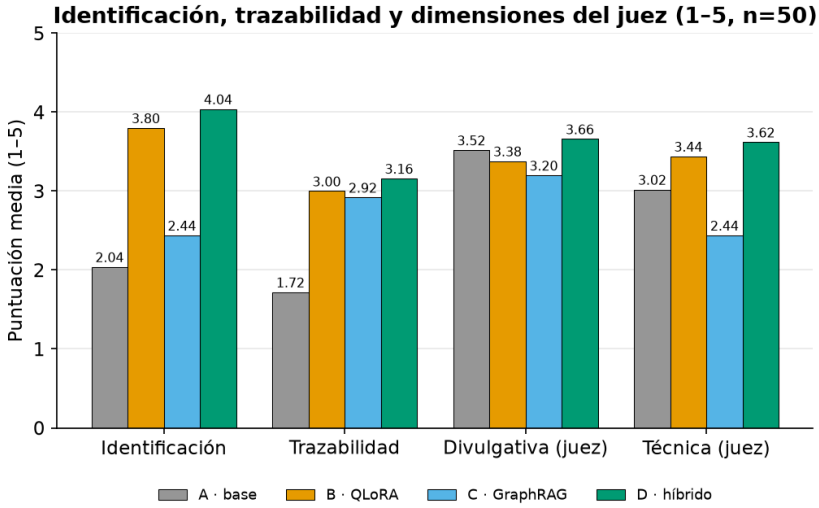

Figura 20. Dimensiones del juez (rúbrica 1-5) por sistema.

La lectura no es la de un ganador entre las vías parciales, sino la de una complementariedad que el híbrido sintetiza. El *fine tuning* del Sistema B sobresale en clasificar el error —acierta la categoría en siete de cada diez casos—, mientras que el GraphRAG del Sistema C aporta fortaleza en el concepto y en la trazabilidad, porque el subgrafo permite nombrar el concepto correcto y justificar el diagnóstico

con la evidencia recuperada. El Sistema D no obliga a elegir, pues hereda ambas virtudes, encabeza las cuatro métricas objetivas y gana o empata en las siete dimensiones (estrictamente el mejor en seis). El orden que emerge es nítido, D > {B, C} > A.

Importa, no obstante, rodear el resultado de cautelas que no son menores. Ninguno de los objetivos cuantitativos del anteproyecto se alcanza, pues el umbral comprometido (acierto de categoría ≥ 85 %) queda lejos incluso para el mejor sistema, cuyo 0,76 lo demuestra. El tamaño muestral también es modesto, con 50 casos; un análisis inferencial[16] confirma que los tres sistemas aumentados superan a la base de forma significativa (*p* < 0,001), pero la ventaja del híbrido sobre el *fine tuning* no alcanza significación (*p* ≈ 0,18 en categoría), por lo que son indistinguibles con esta muestra. La augmentación frente al modelo desnudo es el resultado firme; la superioridad del híbrido sobre su mejor componente es prometedora pero no concluyente.

Otra cautela atañe al juez. Validar su puntuación con un panel de tres modelos de familias distintas —`qwen2.5:32b`, `llama3.1:8b` y `gemma2:9b`— arroja un acuerdo inter-juez por kappa de Fleiss de apenas 0,213 (leve) y prácticamente nulo en las dimensiones divulgativa y técnica, con Qwen sistemáticamente más severo. Lo leo como una advertencia. Las dimensiones cualitativas de un juez automático no poseen validez de criterio robusta, porque dependen de qué modelo juzgue. Esto refuerza que el peso de la evidencia recaiga en las métricas objetivas, y deja la validación con un panel de docentes humanos como el trabajo futuro de mayor prioridad. Una aproximación objetiva propia de la Web Semántica —medir qué fracción de los conceptos citados en el feedback estaban en el subgrafo— sí apunta a favor, ya que el Sistema D alcanza un grounding de 0,65 frente al 0,35 del Sistema C, de manera que el *fine tuning* casi duplica la fidelidad al grafo, si bien esa cifra es una cota inferior.

La prueba más exigente es la generalización a código real de estudiantes. Para abordarla incorporé el subconjunto *Dublin repair* del corpus público `koutch/intro_prog` —entregas de Python con la corrección de su profesorado— midiendo la relevancia al arreglo real del educador. El resultado es importante y desfavorable a la hipótesis, por lo que lo reporto tal cual, ya que sobre código real el orden objetivo se invierte. El modelo base A obtiene la mayor relevancia (0,802), seguido del Sistema C con grafo (0,733), mientras que el *fine tuning* B (0,433) y el híbrido D (0,323) quedan netamente por detrás. Es evidencia directa de un sobreajuste a la estructura superficial de las plantillas; la ventaja en distribución no se transfiere a errores reales. Queda una matización,[17] pero la señal es nítida, pues la superioridad del híbrido es un fenómeno en distribución, cuya transferencia al aula exige atacar la causa raíz del sobreajuste con datos reales o destilación de retroalimentación diversa. Probar el sistema sobre datos reales y reportar un resultado adverso, lejos de debilitar el trabajo, delimita con precisión el alcance de su evidencia.

[16] Friedman, Wilcoxon con corrección de Holm, W de Kendall y bootstrap al 95 %

[17] la métrica premia mencionar los elementos literales que cambian, que el feedback conceptual en español de los sistemas afinados no reproduce, de modo que parte de la diferencia es estilo

17 Discusión, limitaciones y despliegue

Quiero detenerme ahora en lo que significa en conjunto lo construido —del modelado en RDF/OWL a la evaluación comparada de cuatro sistemas, pasando por SHACL, SPARQL y GraphRAG—, para distinguir lo que el trabajo confirma de lo que solo sugiere. La Figura 21 resume las cuatro capacidades del grafo.

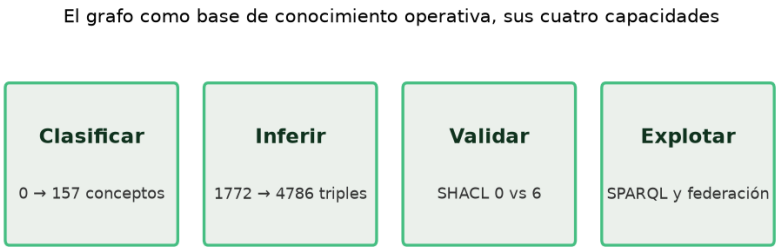

Figura 21. El grafo como base de conocimiento operativa, con sus cuatro capacidades, clasificar (de 0 a 157), inferir (de 1772 a 4786), validar (SHACL 0 frente a 6) y explotar con SPARQL.

17.1 El grafo como estructura operativa

La hipótesis era modesta en su formulación y exigente en su comprobación. Sostenía que una representación formal del dominio mejorara de forma medible y trazable la retroalimentación que un modelo de lenguaje produce por sí solo. Queda confirmada en un sentido acotado; el grafo no es, en abstracto, mejor que cualquier alternativa, sino que funciona como estructura operativa que altera de manera observable la conducta del sistema, y lo es por tres razones. Primero, razona. La consulta de individuos de tipo `pyedu:Concepto` devuelve cero sin inferencia y 157 con el cierre OWL 2 RL activo,[18] y de 1772 enunciados afirmados se pasa a 4786 tras el cierre (Hogan et al., 2021). Segundo, es trazable. La reificación `rdf:Statement` con `dcterms:source` permite remontar un error a su fuente. Tercero, es validable, ya que la conformidad SHACL —cero violaciones sobre el grafo canónico y las 6 previstas en el fichero con errores deliberados— hace que la integridad descanse en restricciones declarativas comprobables (W3C, 2017), lo que importa en educación, donde un grafo erróneo no da un fallo visible, sino una retroalimentación plausible pero incorrecta. Esa es la mejora «medible y trazable» que la literatura reclamaba, no la cantidad del feedback, sino su calidad (Keuning et al., 2019; Hattie & Timperley, 2007).

17.2 La complementariedad B/C y la lógica del Sistema D

El hallazgo que más condiciona mi interpretación es la complementariedad entre B y C, que la Figura 22 sintetiza. Sobre los 50 casos held-out, el Sistema B (Qwen2.5-Coder-7B afinado con QLoRA, sin grafo) sobresale en la clasificación del error porque el fine-tuning internaliza un patrón de respuesta (Dettmers et al., 2023; Hu et al., 2021); el Sistema C (base aumentado con GraphRAG) sobresale en el acierto de concepto porque suministra en inferencia el conocimiento declarativo del dominio con su procedencia (Lewis et al., 2020; Edge et al., 2024). De ahí nace, con lógica casi inevitable, el Sistema D, híbrido que combina ambas vías, que gana o empata en las siete dimensiones y es estrictamente el mejor en seis, con acierto de categoría 0,76 (frente al 0,70 de B), acierto de concepto 0,54 y la mejor trazabilidad de los cuatro. Debo, sin embargo, ser escrupuloso. Que D sea el mejor no significa que alcance las metas del anteproyecto —ninguna se cumple; el 85 % de acierto de categoría fijado se queda en 0,76— y todo se midió con un juez automático, sin panel humano que valide las puntuaciones. La ablación lo afina. A↔C aísla el efecto del grafo y es significativo, y A↔B aísla el del *fine tuning* y también lo es. En cambio, el contraste B↔D —qué aporta el grafo a un modelo ya afinado— es numéricamente positivo pero no alcanza significación a n=50. Cada componente aporta sobre el modelo base y ambos se combinan sin interferir, pero la muestra no basta para demostrar que el grafo aporte por encima del *fine tuning*, la afirmación más fuerte de la hipótesis y donde se juega el valor científico del trabajo.

[18] las 30 entidades de Wikidata enlazadas con `skos:exactMatch` no se infieren como `pyedu:Concepto`, porque ese predicado enlaza sin propagar el tipo, a diferencia de `owl:sameAs`

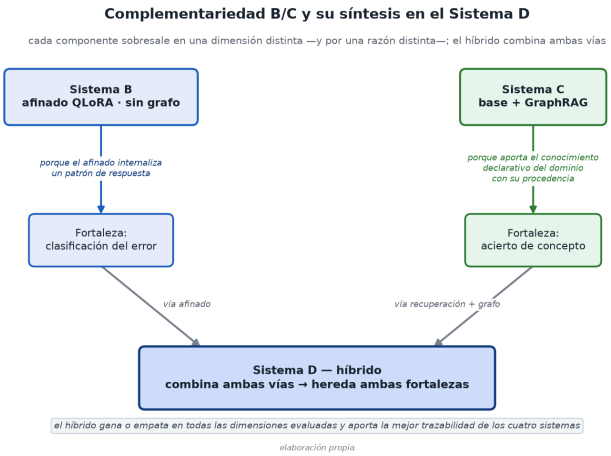

Figura 22. Complementariedad B/C y su síntesis en el Sistema D: el afinado (B) sobresale en la clasificación del error porque internaliza un patrón de respuesta, y el grafo (C) en el acierto de concepto porque aporta conocimiento declarativo con su procedencia; el híbrido D combina ambas vías y hereda ambas fortalezas.

La contribución más distintiva del grafo no es, con todo, la precisión, sino la explicabilidad. Cuando el Sistema C identifica un concepto, este es un nodo con procedencia `dcterms:source` y enlaces `skos:exactMatch` a Wikidata, anclado a entidades verificables que existen con independencia del modelo, frente a la explicación de B, generada y sin garantía de corresponder al proceso real. Esa explicabilidad es real pero parcial. El grafo garantiza que el conocimiento recuperado es explícito, no que el modelo que redacta el texto final —componente opaco que el sistema conserva— lo use de forma igualmente transparente.

17.3 Tensiones de diseño y limitaciones

Ninguna decisión ha sido neutra. La privacidad pesó sobre la potencia. He optado por una pila ejecutable en local,[19] pues el trabajo del estudiante es información sensible; el coste es que los modelos locales son moderados, y parte de la modestia de las cifras absolutas se explica por esta restricción autoimpuesta. La expresividad pesó sobre la simplicidad. He recurrido a relaciones N-arias reificadas mediante `EvaluacionDeConcepto` y `EvaluacionActividad`, a `rdf:Statement` para la procedencia y a `skos:broader` como red distinta de `rdfs:subClassOf` (W3C, 2009; W3C, 2014), y el perfil OWL 2 RL gana razonamiento tratable a cambio de parte de la expresividad de perfiles más ricos (W3C, 2012). Y la generalización pesó sobre la memorización en el *fine tuning* de B, pues la primera ejecución sin regularizar mostró un sobreajuste evidente; el segundo, regularizado con NEFTune (alpha 5) y `lora_dropout` 0,1, lo invirtió, con la pérdida held-out descendiendo de 1,051 a 1,028 y la precisión de token en 0,842. Como el conjunto se generó por plantillas, con riesgo real de fuga, evalué exclusivamente sobre esqueletos held-out.

De ahí se concretan las limitaciones. La más condicionante es el tamaño muestral. El resultado se sostiene sobre 50 casos held-out, de modo que las diferencias son indicios direccionales y no estimaciones robustas; preferí un número moderado pero limpio a inflar la muestra y contaminar la separación entre entrenamiento y prueba. Se suman el sobreajuste descrito, la naturaleza sintética del dataset y el solapamiento de familia entre el modelo evaluado y el juez, pues `qwen2.5:32b` pertenece a la misma familia que el *fine tuning*, lo que abre la puerta a un sesgo de afinidad que no puedo descartar; el panel de jueces que instrumenté arroja un acuerdo de Fleiss de 0,213 — leve —, lo que aconseja no fiarlo todo a la evaluación automática. La frontera más importante es la ausencia de datos reales de estudiantes. Todo se ha validado en un entorno controlado, de modo que las afirmaciones sobre utilidad pedagógica son

[19] Ollama sobre modelos abiertos, GraphDB con su ruleset OWL 2 RL (Ontotext, s.f.), *fine tuning* QLoRA en una única RTX 5090

conjeturas razonadas y no resultados empíricos. El grafo[20] es sólido desde la ingeniería del conocimiento, pero su valor educativo permanece sin contrastar con la única evidencia que importa, la del estudiante que aprende.

17.4 Integración, despliegue y trabajo futuro

La última etapa produce artefactos ejecutables bajo una idea rectora, la verificabilidad. La carga en GraphDB, con soporte nativo de OWL 2 RL, permite al profesor reproducir las cifras nucleares con un puñado de consultas SPARQL. La de individuos de tipo `Concepto` salta de cero a 157, un recuento contrasta los 1772 afirmados frente a los 4786 inferidos, y la de prerrequisitos transitivos de la búsqueda binaria devuelve cinco. Una cautela. El ruleset de GraphDB y el cierre de `owlrl`, aunque ambos implementan OWL 2 RL, pueden diferir en detalles de cobertura, de modo que el recuento de triples inferidos podría no coincidir hasta la última unidad; invariantes son las inferencias significativas. La interfaz web —FastAPI y Cytoscape.js— encarna la explicabilidad, pues muestra el subgrafo que fundamenta cada respuesta y permite comparar las cuatro configuraciones. Es, eso sí, un prototipo de viabilidad, no un producto endurecido. Y la propia memoria es un proyecto MyST que compila una única fuente a web, PDF y Word de manera determinista.

Estas limitaciones orientan la continuación. Pretendo ampliar la muestra e incorporar un panel humano que ancle las métricas, cuya maquinaria ya está armada —anotador automático de familia ajena al juez como cribado, selección por incertidumbre de las celdas más discrepantes y el Alternative Annotator Test (Calderon et al., 2025)—, sumar recuperación híbrida que combine SPARQL con embeddings densos y avanzar hacia la alineación por preferencias mediante ORPO, que exploré de forma preliminar con resultado matizado, pues mejoró el acierto de concepto y la trazabilidad a costa de degradar ligeramente la clasificación del error. La línea más ambiciosa, donde convergen las demás, es el despliegue real en un contexto docente, única vía para sustituir las conjeturas sobre utilidad pedagógica por evidencia. Reconocerlo no debilita el trabajo, sino que lo sitúa como lo que es, un prototipo de investigación validado en sus componentes técnicos y abierto a una continuación cuya dirección conozco.

[20] 157 conceptos propios, 1772 enunciados que ascienden a 4786 tras el cierre, conforme a SHACL

18 Conclusiones

Vuelvo sobre la pregunta que motivó el trabajo para comprobar en qué medida ha quedado respondida. La hipótesis era que un Grafo de Conocimiento Educativo (EKG) sobre programación en Python, construido con los estándares del núcleo de la Web Semántica, validado formalmente y explotado mediante consulta declarativa, podía servir de cimiento verificable para un sistema de retroalimentación formativa apoyado en modelos de lenguaje. Tras diseñarlo, someterlo a inferencia y validación e integrarlo con varios sistemas de generación, mi conclusión es que ese andamiaje no es un envoltorio formal, sino la pieza que confiere transparencia y reproducibilidad a un proceso que, de otro modo, descansaría en la opacidad de un modelo neuronal. El feedback se sostiene en afirmaciones explícitas, trazables y validadas que pueden auditarse una por una, no en una intuición estadística inescrutable. Esa es la aportación de fondo, y la razón por la que considero la combinación de Web Semántica y modelos de lenguaje especialmente fértil en educación, donde rendir cuentas de lo que se afirma importa tanto como el acierto.

18.1 Lo que el grafo demuestra por sí mismo

Partí de un esquema modesto en tamaño pero deliberado en diseño, con 157 conceptos propios en 20 clases, 21 propiedades de objeto y 7 de datos, en Turtle bajo OWL 2 RL y con dos espacios de nombres que separan esquema (TBox) e instancias (ABox). En su forma afirmada el grafo contiene 1772 enunciados. Tras el cierre deductivo asciende a 4786 tripletas (+3014 inferidas), de modo que la inferencia hace explícito un conocimiento que triplica lo que escribí a mano. El contraste que mejor lo ilustra es la consulta por la clase `pyedu:Concepto`, que contra el grafo afirmado devuelve 0 resultados, porque ninguna instancia se declara como tal; con inferencia activada devuelve 157, deducidos por la cadena de `rdfs:subClassOf`. Ese paso de 0 a 157 condensa lo que la semántica formal aporta sobre un almacén plano. El enlazado externo emplea 30 relaciones `skos:exactMatch` —y no `owl:sameAs`— para enlazar a Wikidata sin fusionar ni propagar el tipo propio; el tejido conceptual se completa con 19 `skos:broader`, 19 `skos:narrower` y 16 `skos:related`, distintos de la jerarquía estricta de subclases. La expresividad se puso a prueba con relaciones N-arias reificadas y con `rdf:Statement` y `dcterms:source`, que registran la procedencia bibliográfica de los 16 errores típicos. Poder decir no solo «este es un error frecuente» sino «está documentado en tal fuente» es la trazabilidad que distingue una afirmación responsable de una conjetura.

18.2 La garantía estructural

Ninguna cifra tendría peso sin una garantía de calidad, y aquí SHACL es decisivo. Definí 10 formas de nodo (`sh:NodeShape`) que codifican las restricciones del grafo, y el grafo canónico resulta conforme, con cero violaciones. Para que esa conformidad no fuese vacía —un validador que nunca señala nada no demuestra saber validar— preparé un control negativo con errores introducidos a propósito, y la validación detectó exactamente las 6 violaciones esperadas. En un sistema que generará retroalimentación a estudiantes esa garantía no es un lujo, sino la condición que evita que un dato mal modelado se propague hasta convertirse en un consejo equivocado.

18.3 Integración con LLMs, leída sin complacencia

La parte más ambiciosa, y la más cargada de cautelas, es la integración del grafo con modelos de lenguaje. El *fine tuning* del sistema B con QLoRA sobre Qwen2.5-Coder-7B la ilustra, pues la primera versión presentaba un sobreajuste claro, y solo la versión regularizada invirtió la tendencia y redujo la pérdida en el conjunto reservado de 1.051 a 1.028, con una precisión de token de 0.842. Sobre un benchmark de 50 casos held-out —precaución contra la fuga de datos— el sistema D híbrido se impone, con 0.76 en acierto de categoría y 0.54 en acierto de concepto. Ahora bien, conviene advertir que ni siquiera D alcanza los objetivos del anteproyecto, que la evaluación se apoya en un juez automático sin panel humano, y que con código real de estudiantes (subconjunto Dublin) el orden de la métrica objetiva se invierte, ya que A obtiene 0.802, C 0.733, B 0.433 y D 0.323. La superioridad del híbrido es un fenómeno en distribución, ajuste a la estructura de las plantillas, cuya transferencia al aula queda supeditada a entrenar con datos

reales. Lo reporto sin adornos. El propio panel de jueces arroja un Fleiss de 0.213, acuerdo apenas leve, lo que refuerza la prudencia.

18.4 Recapitulación

No afirmo haber resuelto el feedback automático; afirmo, con la prudencia que el trabajo merece, haber mostrado que el camino que une representación formal del conocimiento y generación de lenguaje es transitable, auditable y digno de seguir explorándose. La Figura 23 condensa ese valor de la capa semántica.

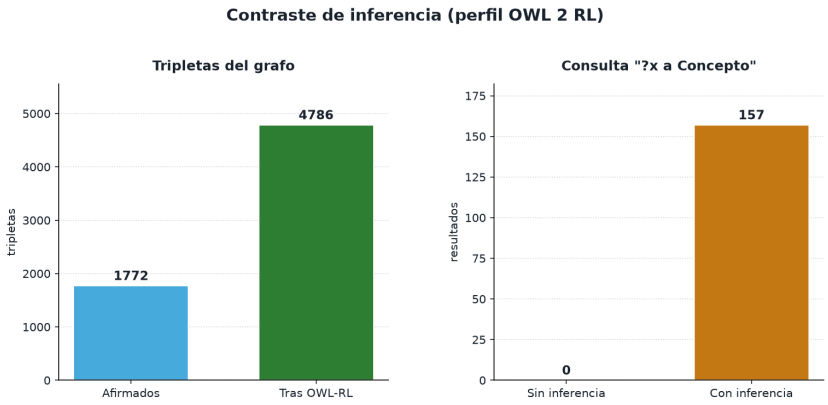

Figura 23. Recapitulación, el salto 1.772⟶4.786 y 0⟶157 condensa el valor de la capa semántica.

19 Referencias

Se recogen a continuación todas las fuentes citadas a lo largo de la memoria, ordenadas alfabéticamente por el apellido del primer autor (o por el nombre de la organización responsable en el caso de las especificaciones y la documentación técnica) y formateadas conforme a la séptima edición del estilo de la American Psychological Association (APA 7). Se aplica sangría francesa en cada entrada, con la primera línea alineada al margen y las siguientes indentadas. Cuando la fuente dispone de identificador de objeto digital se ofrece el DOI en forma de URL; en su defecto se proporciona la dirección estable de consulta. Las especificaciones del World Wide Web Consortium se citan a nombre de la organización y se distinguen con sufijos de letra los recursos publicados en un mismo año (W3C, 2014a; W3C, 2014b) para evitar ambigüedades en las citas integradas del texto. He procurado mantener la coherencia entre las citas en el cuerpo de la memoria y esta lista final, de manera que toda referencia mencionada con el formato (Autor, año) tenga aquí su correspondencia completa y, recíprocamente, ninguna entrada quede huérfana de cita.

Abu-Salih, B., & Alotaibi, S. (2024). A systematic literature review of knowledge graph construction and application in education. *Heliyon*, *10*(3), e25383. https://doi.org/10.1016/j.heliyon.2024.e25383

Calderon, N., Reichart, R., & Dror, R. (2025). The alternative annotator test for LLM-as-a-judge: How to statistically justify replacing human annotators with LLMs. En *Proceedings of the 63rd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)* (pp. 16051–16075). Association for Computational Linguistics. https://aclanthology.org/2025.acl-long.782/

Dettmers, T., Pagnoni, A., Holtzman, A., & Zettlemoyer, L. (2023). QLoRA: Efficient finetuning of quantized LLMs. En *Advances in Neural Information Processing Systems 36 (NeurIPS 2023)* (pp. 10088–10115). Curran Associates. https://doi.org/10.48550/arXiv.2305.14314

Edge, D., Trinh, H., Cheng, N., Bradley, J., Chao, A., Mody, A., Truitt, S., & Larson, J. (2024). *From local to global: A graph RAG approach to query-focused summarization* [Preprint]. arXiv. https://doi.org/10.48550/arXiv.2404.16130

Grattafiori, A., et al. (2024). *The Llama 3 herd of models* [Preprint]. arXiv. https://doi.org/10.48550/arXiv.2407.21783

Hattie, J., & Timperley, H. (2007). The power of feedback. *Review of Educational Research*, *77*(1), 81–112. https://doi.org/10.3102/003465430298487

Hogan, A., Blomqvist, E., Cochez, M., d'Amato, C., Melo, G. de, Gutierrez, C., Kirrane, S., Labra Gayo, J. E., Navigli, R., Neumaier, S., Ngonga Ngomo, A.-C., 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

Hu, E. J., Shen, Y., Wallis, P., Allen-Zhu, Z., Li, Y., Wang, S., Wang, L., & Chen, W. (2021). *LoRA: Low-rank adaptation of large language models* [Preprint]. arXiv. https://doi.org/10.48550/arXiv.2106.09685

Kejriwal, M., Szekely, P., & Knoblock, C. A. (2021). *Knowledge Graphs: Fundamentals, Techniques, and Applications*. MIT Press.

Keuning, H., Jeuring, J., & Heeren, B. (2019). A systematic literature review of automated feedback generation for programming exercises. *ACM Transactions on Computing Education*, *19*(1), 1–43. https://doi.org/10.1145/3231711

Lewis, P., Perez, E., Piktus, A., Petroni, F., Karpukhin, V., Goyal, N., Küttler, H., Lewis, M., Yih, W., Rocktäschel, T., Riedel, S., & Kiela, D. (2020). Retrieval-augmented generation for knowledge-intensive NLP tasks. En *Advances in Neural Information Processing Systems 33 (NeurIPS 2020)* (pp. 9459–9474). Curran Associates. https://doi.org/10.48550/arXiv.2005.11401

Ontotext. (s. f.). *GraphDB documentation*. Ontotext. Recuperado el 18 de junio de 2026, de https://graphdb.ontotext.com/documentation/

Qu, K., Li, K. C., Wong, B. T. M., Wu, M. M. F., & Liu, M. (2024). A survey of knowledge graph approaches and applications in education. *Electronics*, *13*(13), 2537. https://doi.org/10.3390/electronics13132537

RDFLib Team. (s. f.). *RDFLib: A Python library for working with RDF* [Software]. Zenodo. https://doi.org/10.5281/zenodo.6845245

Sommer, A., & Car, N. J. (s. f.). *pySHACL: A Python validator for SHACL* [Software]. RDFLib. https://github.com/RDFLib/pySHACL

UNED. (2025). *Guía de estudio de la asignatura Web Semántica y Enlazado de Datos* (cód. 31108018, curso 2025/2026). Máster Universitario en Investigación en Inteligencia Artificial, Universidad Nacional de Educación a Distancia. https://www.uned.es/universidad/inicio/estudios/masteres/master-universitario-en-investigacion-en-inteligencia-artificial/asignaturas.html?codAsignatura=31108018&codTitulacion=310801

W3C. (2009). *SKOS Simple Knowledge Organization System reference* (W3C Recommendation, 18 de agosto de 2009). World Wide Web Consortium. https://www.w3.org/TR/skos-reference/

W3C. (2012). *OWL 2 Web Ontology Language primer* (2.ª ed., W3C Recommendation, 11 de diciembre de 2012). World Wide Web Consortium. https://www.w3.org/TR/owl2-primer/

W3C. (2013). *SPARQL 1.1 query language* (W3C Recommendation, 21 de marzo de 2013). World Wide Web Consortium. https://www.w3.org/TR/sparql11-query/

W3C. (2014a). *RDF 1.1 concepts and abstract syntax* (W3C Recommendation, 25 de febrero de 2014). World Wide Web Consortium. https://www.w3.org/TR/rdf11-concepts/

W3C. (2014b). *RDF Schema 1.1* (W3C Recommendation, 25 de febrero de 2014). World Wide Web Consortium. https://www.w3.org/TR/rdf-schema/

W3C. (2017). *Shapes Constraint Language (SHACL)* (W3C Recommendation, 20 de julio de 2017). World Wide Web Consortium. https://www.w3.org/TR/shacl/

Yang, A., Yang, B., Zhang, B., Hui, B., Zheng, B., Yu, B., Li, C., Liu, D., Huang, F., Wei, H., Lin, H., Yang, J., Tu, J., Zhang, J., Yang, J., Yang, J., Zhou, J., Lin, J., Dang, K., … Qiu, Z. (2024). *Qwen2.5 technical report* [Preprint]. arXiv. https://doi.org/10.48550/arXiv.2412.15115

19.1 Nota sobre el tratamiento de las fuentes

Conviene precisar el criterio seguido en la confección de este listado, pues no toda la bibliografía pertenece a la misma naturaleza ni se cita con idéntica intención. Por un lado figuran las contribuciones académicas revisadas por pares —artículos de revista y comunicaciones a congresos— que sustentan los fundamentos teóricos y metodológicos del trabajo. Entre ellas figuran los grafos de conocimiento como artefacto de representación (Hogan et al., 2021), la generación aumentada por recuperación y su variante orientada a grafos (Lewis et al., 2020; Edge et al., 2024), las técnicas de *fine tuning* eficiente en parámetros que hacen viable el ajuste de un modelo de lenguaje en una única tarjeta gráfica (Hu et al., 2021; Dettmers et al., 2023), la literatura específica sobre retroalimentación automática en la enseñanza de la programación (Keuning et al., 2019) y el marco teórico clásico sobre la función formativa del feedback (Hattie & Timperley, 2007), así como las revisiones recientes que sitúan los grafos de conocimiento en el dominio educativo (Abu-Salih & Alotaibi, 2024; Qu et al., 2024). Por otro lado se incluyen las especificaciones normativas del World Wide Web Consortium que definen las tecnologías empleadas en la construcción del grafo —RDF y su esquema (W3C, 2014a; W3C, 2014b), el lenguaje de consulta SPARQL (W3C, 2013), la ontología OWL 2 (W3C, 2012), el lenguaje de restricciones SHACL (W3C, 2017) y el vocabulario SKOS para la organización conceptual (W3C, 2009)—, a las que se añade

la documentación técnica del triplestore utilizado en la fase de explotación (Ontotext, s. f.). A estas se suman las bibliotecas de software del núcleo semántico —la manipulación y consulta de RDF y la validación SHACL en Python (RDFLib Team, s. f.; Sommer & Car, s. f.)— y los informes técnicos de los modelos de lenguaje empleados como línea de referencia y como juez en la evaluación (Grattafiori et al., 2024; Yang et al., 2024).

He optado por mantener estas dos familias de fuentes en una única lista alfabética, según prescribe el estilo APA 7, en lugar de segregarlas en secciones independientes. Esta decisión persigue la trazabilidad, pues cualquier afirmación del texto remite a una entrada única e inequívoca, sin obligar al lector a discernir previamente si la cita corresponde a una publicación científica o a una especificación técnica. En las entradas de las recomendaciones del W3C se ha conservado entre paréntesis la categoría del documento y la fecha de publicación, dato relevante dado que estas especificaciones evolucionan por versiones y que la elección de una versión concreta —RDF 1.1, SPARQL 1.1, OWL 2— condiciona aspectos sustantivos de la implementación. Para los recursos sin fecha de publicación estable, como la documentación de GraphDB, se ha incorporado la fecha de recuperación, conforme a la recomendación de APA 7 para contenidos en línea susceptibles de actualización. Soy consciente de que algunos de los preprints citados (Hu et al., 2021; Dettmers et al., 2023; Edge et al., 2024) podrían disponer de versiones publicadas en actas con paginación definitiva; he indicado la referencia de actas cuando la conozco con certeza y he conservado la entrada de arXiv con su DOI en los demás casos, y prefiero la fuente efectivamente consultada a una atribución editorial que no he podido verificar de primera mano.