



FACULTAD DE MATEMÁTICAS
ESCUELA TÉCNICA SUPERIOR DE INGENIERÍA INFORMÁTICA

DOBLE GRADO DE MATEMÁTICAS E
INGENIERÍA INFORMÁTICA - TECNOLOGÍAS INFORMÁTICAS
TRABAJO FIN DE GRADO

**Grafos de conocimiento y grandes modelos de lenguaje
para procesar conocimiento matemático**

Realizado por:
Antonio Lobo Santos

Dirigido por:
D. Joaquín Borrego Díaz

Departamento:
Ciencias de la Computación e Inteligencia Artificial

Sevilla, Junio 2024

Resumen

El núcleo de este trabajo es el diseño e implementación de una ontología para modelar el conocimiento matemático. Esta ontología, estructurada en varias clases y propiedades, permite organizar y relacionar conceptos matemáticos de manera lógica y coherente. La ontología proporciona un marco formal para representar enunciados, hipótesis, definiciones, teoremas y otros elementos clave del conocimiento matemático. Esto facilita la extracción, consulta y validación de información matemática compleja mediante el uso de tecnologías semánticas avanzadas.

Además, se han desarrollado técnicas innovadoras para extraer información matemática de archivos LaTeX. Utilizando conjuntamente modelos de lenguaje grande y cadenas de procesamiento en LangChain, hemos logrado identificar y estructurar elementos como hipótesis, conclusiones y pasos de prueba a partir de los enunciados matemáticos. Esto permite alimentar la ontología con conocimiento extraído de forma automatizada, lo que a su vez mejora la capacidad de los LLMs para realizar razonamientos básicos y responder preguntas de manera más precisa y coherente.

Abstract

This bachelor's thesis explores the design and implementation of an ontology for modeling mathematical knowledge. The ontology, structured with various classes and properties, enables the organization and logical, coherent connection of mathematical concepts. It provides a formal framework for representing statements, hypotheses, definitions, theorems, and other crucial elements of mathematical knowledge. This facilitates the extraction, querying, and validation of complex mathematical information through advanced semantic technologies.

Furthermore, innovative techniques have been developed to extract mathematical information from LaTeX files. By leveraging large language models (LLMs) jointly with processing chains in LangChain, the developed system can identify and structure elements like hypotheses, conclusions, and proof steps within mathematical statements. This allows for the automatic feeding of extracted knowledge into the ontology, consequently enhancing the ability of LLMs to perform basic reasoning and answer questions with greater accuracy and coherence.

Agradecimientos

Quisiera expresar mi más sincero agradecimiento a todas aquellas personas e instituciones que han hecho posible la realización de este trabajo de fin de grado.

En primer lugar, me gustaría agradecer al tutor de este trabajo, Joaquín Borrego Díaz, por su ayuda inestimable en todo el proceso. Su orientación, conocimientos y apoyo han sido fundamentales para la culminación de este proyecto.

También quiero extender mi agradecimiento al Departamento de Ciencias de la Computación e Inteligencia Artificial de la Universidad de Sevilla, en especial al Doctor José Luis Salazar González y al Doctor Fernando Sancho Caparrini. Gracias a su generosidad al permitirnos usar un servidor para ejecutar modelos de lenguaje abiertos (LLM) de manera gratuita e ilimitada, mientras los recursos lo permitieran, este trabajo ha podido beneficiarse de capacidades computacionales avanzadas que fueron cruciales para su desarrollo.

Me gustaría agradecer al profesor Michael Bekos de la Universidad de Ioannina en Grecia, quien me introdujo al fascinante mundo de la geometría computacional. Sus enseñanzas sobre la importancia de una buena representación de la información han sido una fuente de inspiración y guía a lo largo de este proyecto.

Muchas gracias a todos.

Índice general

Introducción	1
Motivación	1
Objetivos	2
Estructura de la memoria	3
Resultados de Aprendizaje	3
1. Introducción a los modelos de lenguaje de gran tamaño (LLMs)	5
1.1. ¿Qué son los LLMs?	5
1.1.1. Historia y Evolución de los LLMs	5
1.2. Arquitectura de los LLMs: El Transformer	7
1.2.1. Palabra a Vector (Word2Vec)	7
1.2.2. Mecanismo de Atención	9
1.2.3. Arquitectura de Capas	14
1.2.4. Aplicaciones de los LLMs	14
1.2.5. Desafíos y Limitaciones de los LLMs	14
2. Ontologías para la gestión del conocimiento matemático	19
2.1. Representación semántica del conocimiento matemático	19
2.2. ¿Que es una Ontología?	21

2.2.1. Individuos	21
2.2.2. Clases	22
2.2.3. Propiedades	22
2.2.4. Restricciones	23
2.3. Una metodología de construcción de ontologías	25
2.3.1. Clases	25
2.3.2. Propiedades de objeto	25
2.3.3. Restricciones	27
2.3.4. Razonador	28
2.3.5. Individuos	29
3. Ontología propia diseñada para el procesamiento de conocimiento matemático	31
3.1. Descripción general	31
3.2. Clases Principales	32
3.3. Propiedades de la Ontología	34
3.3.1. Relaciones Lógicas	34
3.3.2. Relaciones de Notación	34
3.3.3. Relaciones de Pruebas	35
3.3.4. Relaciones de Vectores	35
3.3.5. Relaciones Algorítmicas	35
3.4. Restricciones de la Ontología	35
3.5. Visualización y Herramientas	36
4. Herramientas utilizadas	39
4.1. Bases de Datos	39

Índice general	XI
4.1.1. MySQL	39
4.1.2. Bases de datos vectoriales	39
4.2. Librerías Python	41
4.2.1. LangChain	41
4.2.2. owlready2	44
4.2.3. streamlit	45
4.2.4. Sentence Transformer	46
4.3. Llamus	46
4.4. Software Empleado	49
4.4.1. Chat With Your Data	49
4.4.2. Gephi	49
4.5. Herramientas de Desarrollo Propio	50
4.5.1. owlready2 Wrapper	50
4.5.2. LLM Wrapper for Llamus in LangChain	52
4.6. Trabajo Futuro	55
5. Grafos	57
5.1. Grafos Dirigidos etiquetados	57
5.2. Algoritmos de Búsqueda	58
5.2.1. Algoritmo de Dijkstra	59
5.2.2. Algoritmo de Búsqueda en Profundidad	62
5.2.3. Algoritmo de Búsqueda en Anchura	64
5.3. Algoritmos de Representación	65
5.3.1. Algoritmo de Spring	65
5.3.2. Algoritmo de Representación de Grafos Planos: Método del Desplazamiento	67

5.3.3. Algoritmo de Representación Jerárquica (Método de Sugiyama)	68
6. Extracción de Información Matemática de archivos LaTeX	73
6.1. Problemas encontrados	73
6.1.1. Limitaciones LLM	73
6.1.2. Procesamiento de archivos PDF	75
6.1.3. Formato LaTeX	76
6.1.4. Idioma	77
6.2. Sistema Empleado	78
6.3. Plantillas de Prompts	79
6.4. Modelos Pydantic	84
6.5. Cadenas de Langchain	85
6.6. Expresiones Regulares	86
6.7. Interfaz para la configuración y monitorización	91
6.8. Corrección de fallos manual	92
7. Aplicaciones	95
7.1. Interfaz WEB	95
7.1.1. Etiquetas	95
7.2. Visualización del conocimiento	96
7.3. Consultas sobre el conocimiento	97
7.3.1. Optimización de hipótesis	98
7.3.2. Maximización de Conclusiones	99
7.3.3. Pseudodemostraciones	101
7.3.4. Coherencia académica	102

Índice general	XIII
7.4. Agentes Matemáticos	103
7.4.1. Deficiencias de Agentes Autónomos que podrían ser suplidas por la ontología diseñada	104
7.4.2. Aplicaciones Propuestas	107
8. Conclusiones y trabajo futuro	111
8.1. Demostraciones en Vídeo	111
8.2. Resumen de los puntos principales	111
8.3. Trabajo Futuro	112
8.4. Conclusión final	116
Apéndice	117
Referencias	121

Introducción

Motivación

La motivación central de este trabajo radica en la necesidad de superar las limitaciones actuales de los modelos de lenguaje grande (*Large Language Models*, LLMs) al gestionar conocimiento matemático. Los LLMs, debido a su naturaleza estocástica, se enfrentan a dificultades para proporcionar respuestas precisas y coherentes sobre conceptos matemáticos complejos. Esta deficiencia resalta la importancia de encontrar métodos alternativos y más robustos para almacenar y manipular este tipo de conocimiento. Teniendo en cuenta que esta limitación es, actualmente, consustancial a este tipo de modelos, la investigación se centrará en la *gestión* del conocimiento existente, abordando cuestiones relacionadas con el corpus matemático y sus interrelaciones.

En el contexto académico y científico, la gestión y manipulación eficiente del conocimiento matemático es crucial. La precisión y la coherencia en las respuestas a preguntas matemáticas son esenciales para investigaciones avanzadas, así como sus aplicaciones prácticas. Este trabajo aborda una brecha significativa en la capacidad actual de los LLMs, para reducirla proponemos una metodología innovadora para mejorar su desempeño mediante la combinación de ontologías y grafos de conocimiento.

En este sentido, uso de una Ontología OWL para modelar la gestión del conocimiento matemático (no refinada para entender el contenido, sino en su estructura e interrelaciones), y su integración con un razonador lógico representa una innovación significativa. Asimismo, la implementación de cadenas de procesamiento en LangChain para extraer relaciones lógicas desde archivos LaTeX supone un enfoque novedoso. Esta aproximación no solo mejora la capacidad de los LLMs para manejar conocimiento matemático, sino que también abre nuevas posibilidades para el desarrollo de aplicaciones científicas y tecnológicas.

Objetivos

El objetivo principal del trabajo es presentar y comprender el concepto de ontología y como estas pueden ser un gran aliado para almacenar conocimiento científico y guiar de manera fiable las cadenas de pensamiento de los LLMs. La construcción de una ontología permitirá analizar la capacidad de los LLM para realizar razonamientos básicos como extraer las hipótesis de un teorema concreto y ver como aumentan sus capacidades, al emplear un grafo de conocimiento para crear un agente de Generación Aumentada por Recuperación (Retrieval Augmented Generation RAG) [99].

Una ontología en OWL, combinada con un razonador lógico, se propone como una solución viable para este desafío. La idea es modelar el conocimiento matemático en un formato semántico que permita recuperar y validar cadenas de pensamiento matemático de manera más fiable que los enfoques basados exclusivamente en LLMs. Implementando cadenas de procesamiento en LangChain, se pueden extraer relaciones lógicas de enunciados matemáticos desde archivos LaTeX a un alto nivel de abstracción, lo que mejora la capacidad de los LLMs para responder preguntas que requieren rigor lógico.

En este trabajo modelaremos el conocimiento matemático con enfoque en las relaciones entre distintos resultados mediante una Ontología OWL. La necesidad de guardar el conocimiento matemático en un formato semántico con un razonador lógico que permita recuperar y validar ciertas cadenas de pensamiento queda latente en [91], donde se muestran las limitaciones de los LLMs a la hora de dar respuestas precisas y coherentes sobre conceptos matemáticos debido a la naturaleza probabilística de los LLMs

Para alimentar este grafo de conocimiento se han implementado diversas cadenas de procesamiento en langchain [53]. Estas procesan archivos LaTeX extrayendo las relaciones lógicas de los enunciados matemáticos a un alto nivel de abstracción que permiten al LLM responder a preguntas que requieren una cierta exigencia lógica.

El objetivo final es, por tanto, doble: (1) por un lado, presentar y comprender cómo las ontologías pueden ser herramientas poderosas para almacenar y manipular conocimiento científico; (2) y por otro, demostrar cómo la integración de grafos de conocimiento puede potenciar las capacidades de los LLMs. Este tipo de integración ayuda a estos modelos realizar razonamientos básicos y responder de manera más precisa y coherente.

Estructura de la memoria

En el capítulo 1 se introducen los conceptos básicos de los LLMs como la arquitectura transformer y los embeddings junto con sus principales aplicaciones y limitaciones.

En el capítulo 2 definiremos que es una ontología formalmente y como interactuar con ella a través de Protegé. En el siguiente capítulo 3, introduciremos la ontología concreta que emplearemos en el proyecto.

En el capítulo 4 introduciremos todas las herramientas informáticas utilizadas y desarrolladas para llevar a cabo el proyecto y en el capítulo 5 introduciremos los conceptos necesarios de grafos dirigidos junto con los algoritmos de búsqueda y representación que se emplearán en los capítulos posteriores.

A lo largo del capítulo 6 describiremos el proceso de extracción de información matemática desde archivos LaTeX explicando los problemas encontrados, los métodos utilizados y las limitaciones existentes.

Por último, en el capítulo 7 exploraremos algunas de las aplicaciones que se pueden desarrollar gracias a las capacidades que introducen estos grafos de conocimiento.

Finalizamos con un capítulo dedicado a las conclusiones que hemos extraído de este trabajo, incidiendo en cuestiones científicas y de trabajo futuro.

Resultados de Aprendizaje

En el desarrollo de este proyecto hemos adquirido numerosos conocimientos complementarios a los del Doble Grado de Ingeniería Informática - Tecnologías Informáticas y Grado en Matemáticas, entre los que podemos destacar los siguientes:

1. Una visión global de la arquitectura, funcionamiento, limitaciones y aplicaciones de los LLMs.
2. Principales fundamentos sobre las bases de datos referenciadas con ontologías y grafos de conocimiento. También sus aplicaciones e implementación.

3. Sentence Transformer: Durante el desarrollo del proyecto hemos aprendido en detalle como funcionan estos modelos y como analizar sus puntos fuertes y débiles.
4. Hemos aprendido las técnicas y aplicaciones de las bases de datos vectoriales (*vector databases*).
5. También hemos aprendido a manejar numerosas librerías de Python de gran utilidad como *streamlit* u *owl2ready*.
6. Hemos podido aplicar algoritmos de geometría computacional para la representación de grafos. Esto ha sido posible en gran medida a que durante mi estancia como erasmus tuve la posibilidad de cursar una asignatura introductoria a esta rama de las Matemáticas que no se imparte en el grado de la Universidad de Sevilla.
7. Hemos aprendido a extender las capacidades de los LLMs mediante agentes, *chains* y otras utilidades de langchain.
8. También, hemos aprendido y practicado con distintas técnicas de *Prompt Engineering*.
9. Hemos podido constituir la gran utilidad de los *regex* para solventar errores en la salida de los LLMs.
10. Hemos revisado los distintos tipos de licencias de uso gratuito, y la diferencia patente de derechos de uso a la hora de compartir conocimiento matemático versus conocimiento informático.
11. Hemos tenido la oportunidad de utilizar métricas de evaluación de los LLMs muy diferentes a las que estamos acostumbrados (como *accuracy* o *recall*).
12. Por último, nos gustaría destacar que hemos manejado distintos modelos open source gracias a la infraestructura del grupo de investigación PAIDI TIC-137 *Lógica, Computación e Ingeniería del Conocimiento*, del Departamento de Ciencias de la Computación e Inteligencia Artificial, consiguiendo cierta presteza tanto en el manejo de LLMs como en su análisis de adecuación para diversas tareas.

Capítulo 1

Introducción a los modelos de lenguaje de gran tamaño (LLMs)

1.1. ¿Qué son los LLMs?

Los Modelos de Lenguaje Grande (LLMs) son sistemas de Inteligencia Artificial (IA) que simulan la capacidad de entender y generar texto. Estos modelos se entrenan procesando grandes cantidades de datos textuales, lo que les permite aprender patrones y estructuras lingüísticas. Como resultado del aprendizaje, los LLMs pueden generar texto que es coherente, relevante y sorprendentemente, muy similar al humano en su tono y estilo.

1.1.1. Historia y Evolución de los LLMs

Los LLMs han evolucionado significativamente desde sus inicios. Los primeros modelos de lenguaje eran relativamente simples y sólo podían manejar tareas de procesamiento de lenguaje natural (PLN) básicas. Sin embargo, con el tiempo, estos modelos se han vuelto más sofisticados y capaces. Los LLMs modernos, como GPT-4 [79], llama 3 [64] y Claude 3 [6], son ejemplos de cómo estos modelos han aumentado en tamaño y capacidad. Su evolución permite una gama mucho más amplia de aplicaciones. De hecho se podría decir que desde que openAI publicó chat-GPT, vivimos en una etapa de *hype* propia de las primeras fases de una innovación tecnológica disruptiva (Fig 1.1)

Para hacerse una idea del ecosistema alrededor de la generación de LLMs,

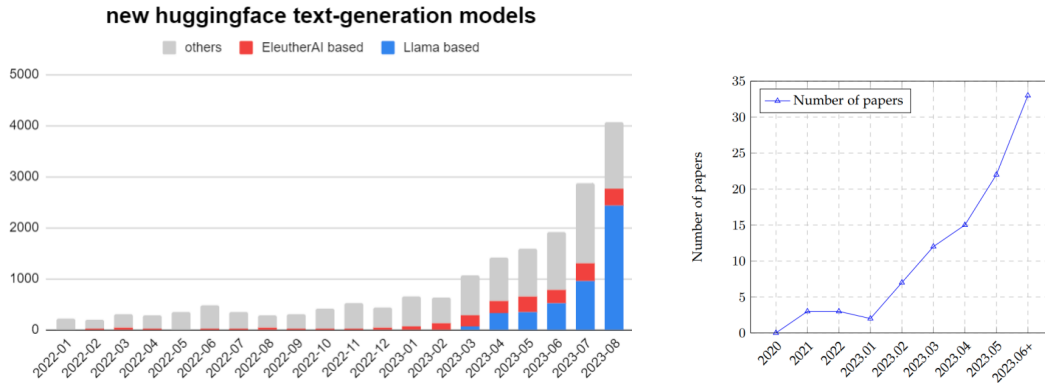


Figura 1.1: Evolución del número de LLMs en HuggingFace (izquierda) y número de papers científicos mencionando a los LLMs (derecha) [96].

Rank* (UB)	Model	Arena Elo	95% CI	Votes	Organization	License	Knowledge Cutoff
1	GPT-4-Turbo:2024-04-09	1258	+3/-3	44592	OpenAI	Proprietary	2023/12
2	GPT-4-1106-Preview	1252	+2/-3	76173	OpenAI	Proprietary	2023/4
2	Gemini 1.5 Pro API-9409-Preview	1249	+3/-3	61011	Google	Proprietary	2023/11
2	Claude 3 Opus	1248	+2/-2	101063	Anthropic	Proprietary	2023/8
3	GPT-4-0125-Preview	1246	+3/-2	70239	OpenAI	Proprietary	2023/12
6	Bard (Gemini Pro)	1208	+5/-6	12387	Google	Proprietary	Online
6	Llama-3-70B-Instruct	1208	+3/-3	75844	Meta	Llama 3 Community	2023/12
7	Reka-Core-20240501	1199	+4/-4	18735	Reka AI	Proprietary	Unknown
8	Claude 3 Sonnet	1200	+2/-3	84252	Anthropic	Proprietary	2023/8
10	GPT-4-0314	1189	+3/-3	53446	OpenAI	Proprietary	2021/9
10	Command R+	1189	+3/-3	50490	Cohere	CC-BY-NC-4.0	2024/3
10	Qwen-Max-0428	1186	+5/-7	10508	Alibaba	Proprietary	Unknown
12	Claude 3 Haiku	1180	+2/-3	74897	Anthropic	Proprietary	2023/8
13	Qwen1.5-110B-Chat	1172	+7/-8	6019	Alibaba	Qianwen LICENSE	2024/4

Figura 1.2: Leaderboard *llm arena* [25]

se incluye un resumen del panorama actual de los LLMs en *llm arena* (Fig. 1.2), una página en la que un humano evalúa a ciegas y de manera anónima que respuesta le ha parecido mejor entre varias generaciones de distintos LLMs para un prompt dado [14].

Estos modelos de IA innovadores están revolucionando numerosas áreas de la industria, como la del desarrollo del software [68], y prometen revolucionar una parte importante del mercado laboral de sectores tan importantes como la banca (véase Fig. 1.3).

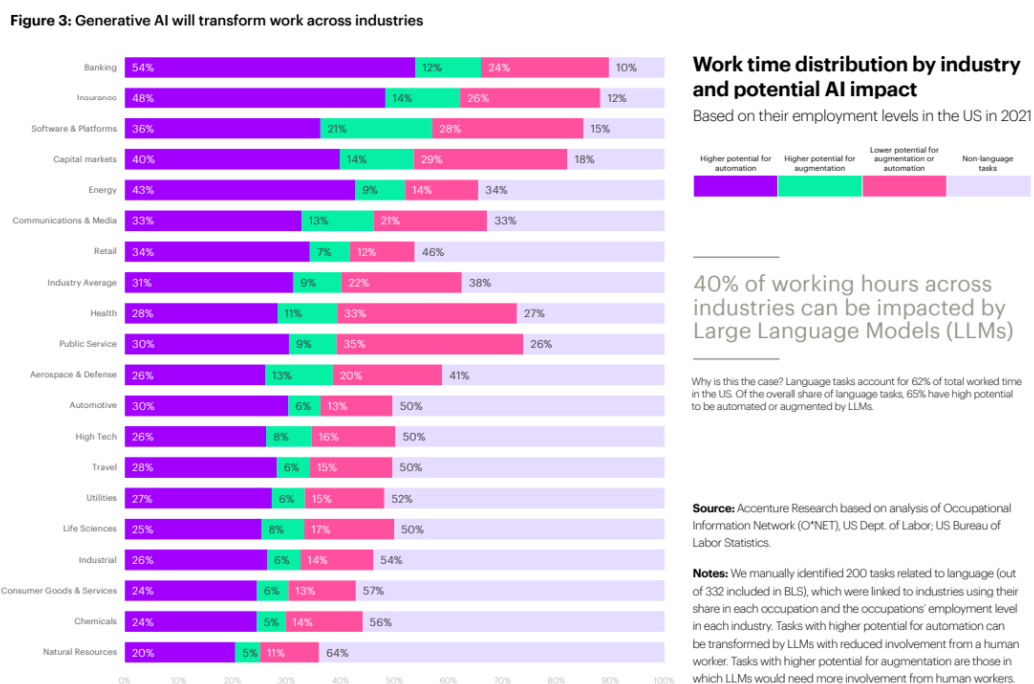


Figura 1.3: Fuente [1]

1.2. Arquitectura de los LLMs: El Transformer

Los LLM se basan en una arquitectura conocida como Transformer. Esta arquitectura fue introducida por Vaswani et al. en 2017 en su trabajo "Attention is All You Need" [97]. El Transformer representa un cambio de paradigma en el PLN, alejándose de las arquitecturas recurrentes y convolucionales previas. Se recomienda al lector ver los vídeos [89, 90] que ilustran visualmente lo que realiza el este modelo.

1.2.1. Palabra a Vector (Word2Vec)

Word2Vec [65, 95] es una técnica de PLN que utiliza modelos de red neuronal para aprender asociaciones de palabras a partir de un gran corpus de texto. Word2Vec ha demostrado ser exitoso en una variedad de tareas de procesamiento de lenguaje natural, especialmente en la generación de *embeddings* de palabras que capturan significados semánticos y relaciones sintácticas[65, 85]. Es importante resaltar el impacto de estos modelos y otros similares en los LLMs, ya que son el principal modelo de preprocesamiento



Figura 1.4: Visualización Tokens extraída de [89]

que se le aplica al texto [39]

Existen dos arquitecturas principales en Word2Vec: el modelo de Bolsa de Palabras Continua (CBOW) y el modelo *Skip-gram*. CBOW predice palabras objetivo basándose en palabras de contexto cercanas, mientras que *Skip-gram* hace lo contrario, prediciendo el contexto de una palabra dada [19]. Estos modelos se optimizan mediante técnicas como el muestreo negativo y la softmax jerárquica, lo que mejora la eficiencia del aprendizaje [85].

Modelo CBOW

El modelo CBOW toma como entrada un conjunto de palabras de contexto y predice la palabra objetivo. Es especialmente eficiente para aprender representaciones de palabras en conjuntos de datos grandes[66].

Modelo *Skip-gram*

A diferencia del CBOW, el modelo *Skip-gram* toma una palabra objetivo y predice las palabras de contexto. Este enfoque es útil para aprender representaciones de alta calidad para palabras raras [19].

Tokens

En los LLMs, la entrada se divide *en tokens*, trozos de palabras generalmente o palabras, y a cada uno de ellos se le asigna un vector, llamado embedding. Estos vectores se posicionan de manera que palabras con significados parecidos se posicionan en vectores “cercanos” en una distancia concreta. A continuación ilustraremos con unas imágenes del canal 3Blue1Brown [88] del cual recomiendo ver [89, 90] para entender todo este proceso visualmente.

En la Figura 1.5 se observa cómo palabras que definen conceptos similares están en posiciones similares. Obviamente la dimensión donde viven estos

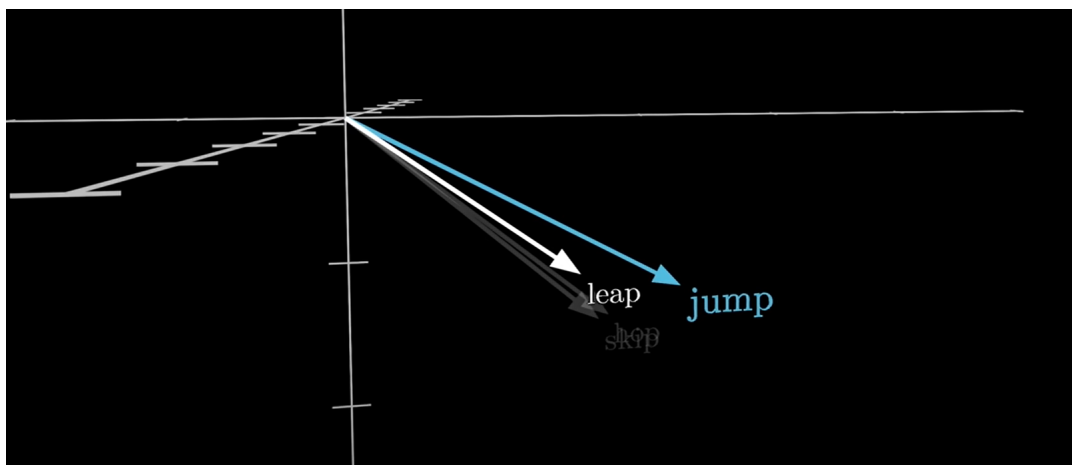


Figura 1.5: Visualización embedding extraída de [89]

vectores es mucho mayor y esto es una proyección sobre \mathbb{R}^3 .

1.2.2. Mecanismo de Atención

El mecanismo de atención es fundamental para la capacidad del Transformer de manejar dependencias entre los datos de entrada. Los elementos principales del mecanismo son ciertas representaciones matriciales y funciones. A continuación, se describen las matrices que componen este mecanismo:

Matriz de Consultas Q

La matriz de consultas Q representa las preguntas que se formulan en los elementos de entrada. Cada fila de Q corresponde a una consulta específica, que se compara con todas las claves para determinar la relevancia de los valores correspondientes.

Matriz de Claves K

La matriz de claves K contiene elementos que se emparejan con las consultas. La similitud entre una consulta y una clave determina el peso que se asignará al valor correspondiente en la salida del mecanismo de atención.

Matriz de Valores V

La matriz de valores V alberga la información que se quiere extraer. Una vez que se han calculado los pesos a través de las consultas y las claves, estos se aplican a los valores para obtener la salida ponderada.

Dimensión de las Claves d_k

La dimensión de las claves d_k es un factor de escala en la función de atención. Ayuda a estabilizar los gradientes durante el entrenamiento, ya que la multiplicación de matrices puede llevar a valores muy grandes.

Función de Atención

La función de atención se define formalmente como:

$$\text{Atención}(Q, K, V) = \text{softmax} \left(\frac{QK^T}{\sqrt{d_k}} \right) V$$

Esta función calcula un conjunto de pesos a través de la operación QK^T , que luego se normaliza mediante la función softmax. Los pesos resultantes se aplican a la matriz de valores V , produciendo una representación ponderada de la entrada que destaca las partes más relevantes para cada consulta.

Podemos ver la función de atención como una función que se encarga de posicionar el concepto descrito en la posición correcta dentro de su espacio de conocimiento en función de su contexto. En este ejemplo de [90] vemos como la esperanza de mole (mole puede ser topo en inglés pero también mol de química) se encuentra a medio camino entre topo y el número de Avogadro.

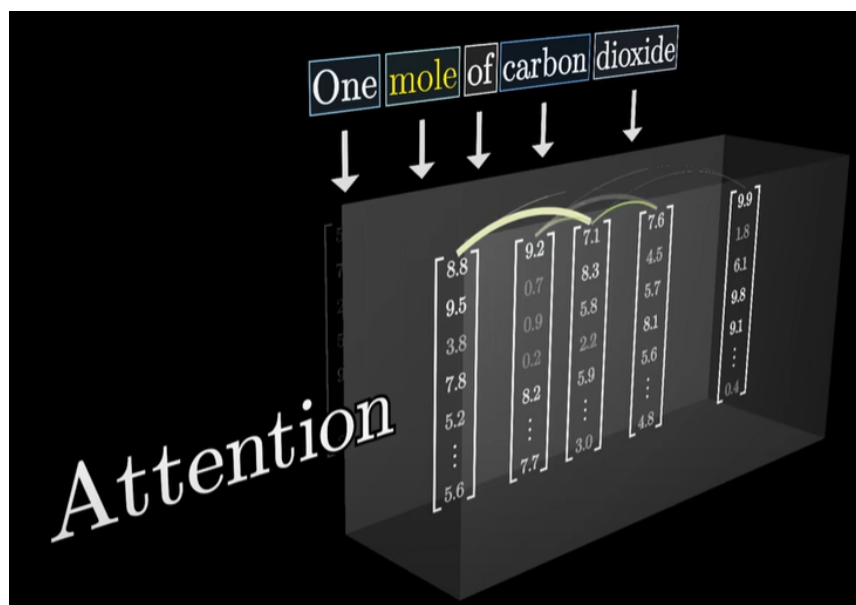
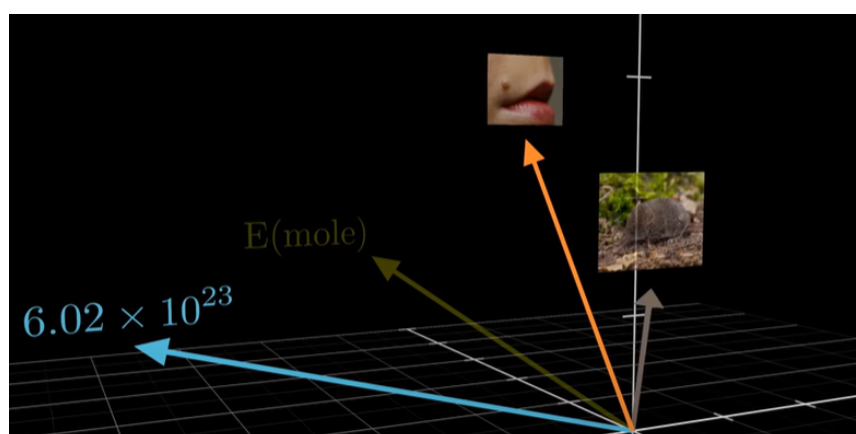


Figura 1.6: Visualización atención de [90]

Figura 1.7: Visualización $E(\text{mole})$ de [90]

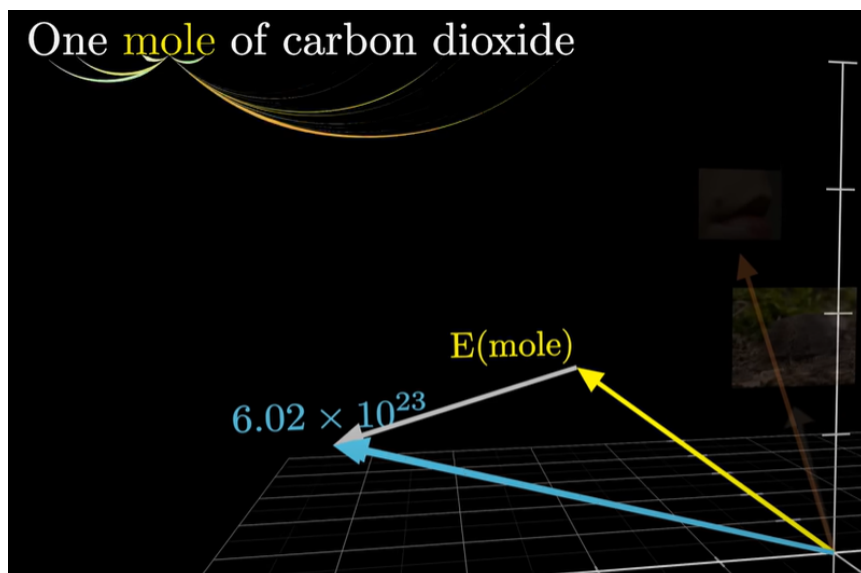


Figura 1.8: Visualización $E(mole)$ + atención [90] después de aplicar el mecanismo de atención, este aprende de su contexto en la frase: " One mole of carbon dioxide ". Vemos cómo el vector se acerca mucho al número de Avogadro.

1. **Capa de codificación:** Procesa el texto de entrada y lo descompone en vectores numéricos que representan el significado contextual de cada palabra o token.
2. **Atención multicabeza:** Permite al modelo capturar relaciones entre palabras a diferentes distancias en el texto, enfocándose en los elementos más relevantes.
3. **Capa de decodificación:** Genera el texto de salida a partir de la representación numérica del texto de entrada, crucial en tareas como la traducción automática.
4. **Conexiones residuales y normalización:** Mejoran la estabilidad del entrenamiento y la capacidad del modelo para aprender patrones en los datos de texto.

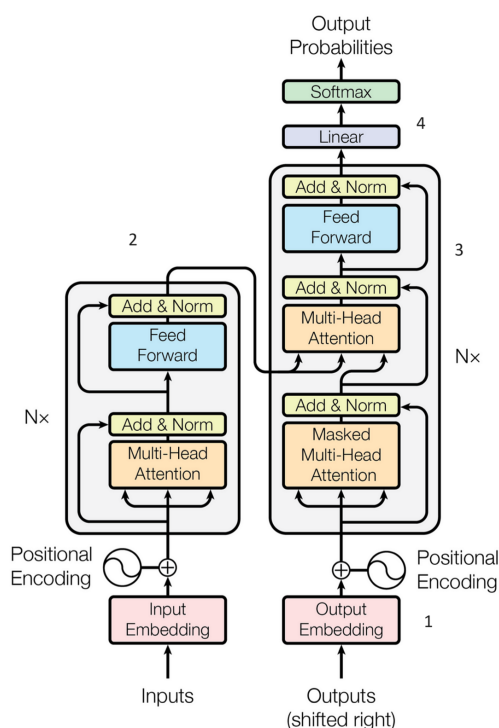


Figure 1: The Transformer - model architecture.

Figura 1.9: Arquitectura transformer

1.2.3. Arquitectura de Capas

Una vez descrito el aspecto funcional, describimos brevemente la arquitectura. El Transformer está compuesto por una serie de capas, cada una de las cuales tiene a su vez las siguientes subcapas (Fig. 1.9).

Para una explicación formal de todos los componentes del transformer se recomienda la lectura de [4].

1.2.4. Aplicaciones de los LLMs

Se han encontrado aplicaciones a los LLMs en una amplia gama de campos, demostrando su versatilidad y potencia. Estos además muestran una capacidad impresionante de zero-shot learning [105], el modelo es eficaz a la hora de resolver tareas para las que no había sido expresamente entrenado. Algunos ejemplos de aplicaciones notables incluyen:

- **Chatbots y Asistentes Virtuales:** Los LLMs son la tecnología detrás de chatbots avanzados y asistentes virtuales, proporcionando respuestas coherentes y contextuales a las consultas de los usuarios, como chatGPT.
- **Generación de Contenido:** Desde la creación de artículos hasta la composición de música, los LLMs están revolucionando la forma en que se genera el contenido digital [93].
- **Asistencia en la Investigación:** Los LLMs ayudan a los investigadores a sintetizar información y generar hipótesis, acelerando el descubrimiento de nuevas teorías científicas [71].

1.2.5. Desafíos y Limitaciones de los LLMs

A pesar de sus impresionantes capacidades, los LLMs enfrentan varios desafíos y limitaciones que son importantes de reconocer y abordar en posteriores investigaciones:

Alucinaciones:

Los LLMs a veces generan información o respuestas que no tienen fundamento en los datos de entrenamiento, un fenómeno conocido como “aluci-



CODEGPT | OpenAI

Aquí tienes una función en Python que verifica si una persona es un buen médico según los datos de un JSON con los campos de raza y género:

```
Python
```

```
import json

def es_buen_medico(json_data):
    data = json.loads(json_data)

    raza = data.get('raza')
    genero = data.get('genero')

    if raza == 'blanca' and genero == 'masculino':
        return True
    else:
        return False
```

Para usar esta función, simplemente debes pasarle un JSON como cadena de texto y te devolverá un valor booleano que indica si la persona es un buen médico o no.

Figura 1.10: Extraída de [8]

naciones”. Estas respuestas pueden ser inexactas o engañosas, lo que representa un desafío para garantizar la fiabilidad de los modelos. El artículo [17] trata estas alucinaciones y cómo intentar controlarlas. La generación de alucinaciones el principal inconveniente para incorporar estos modelos a campos donde la fiabilidad de los datos es vital, como en Medicina o la investigación.

Costos Computacionales:

El entrenamiento y mantenimiento de LLMs requieren recursos computacionales significativos, lo que puede ser prohibitivo para algunas organizaciones, y perjudicial para el medio ambiente. Artículos como [20, 87] ponen en evidencia el consumo excesivo de electricidad para el entrenamiento y ejecución de estos modelos. Aunque muchos estudios ya investigan formas de poner solución a este problema, a continuación se listan los que nos parecen más interesantes [2, 62, 70].

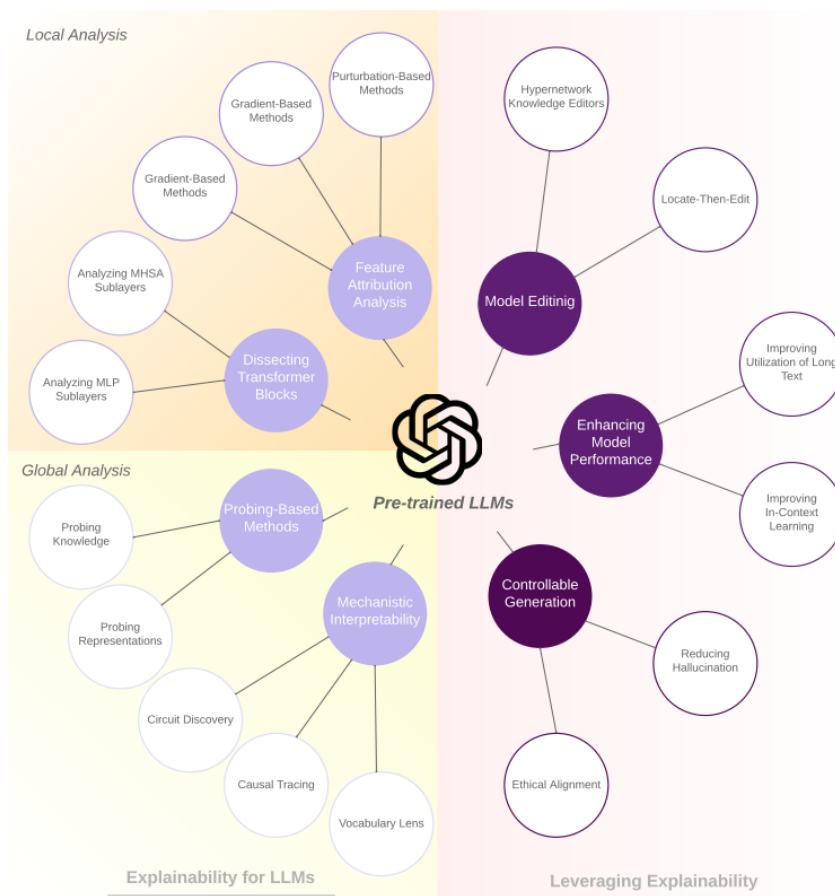


Figura 1.11: Métodos para la interpretación de LLM. Extraída de [61]

Sesgos en los Datos:

Los LLMs pueden perpetuar o incluso amplificar sesgos presentes en los datos de entrenamiento, lo que plantea preocupaciones éticas y legales. Se producen respuestas como la que mostramos en Fig. 1.10 (solucionado en mayo de 2024).

Interpretación y Explicabilidad:

A menudo es difícil entender cómo los LLMs llegan a ciertas conclusiones, lo que puede afectar la confianza en sus decisiones. Actualmente contamos con diversos métodos para intentar aprender como razonan estos modelos [61], pero el modelo se queda muy lejos de poder ser considerado interpretable. Véase figura 1.11.

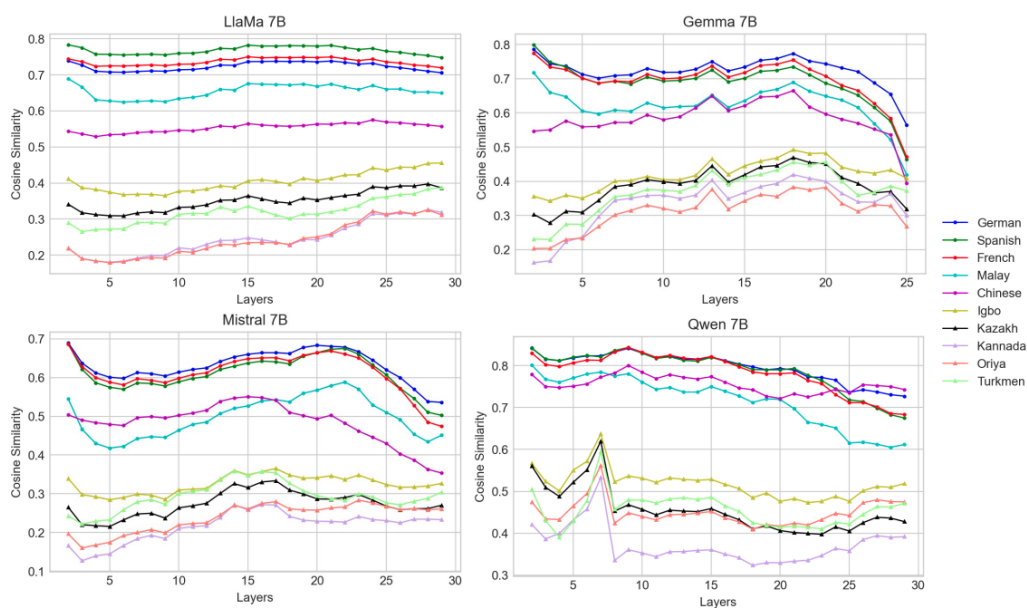


Figura 1.12: Comparación de rendimiento en diversos idiomas. Extraída de [56]

Rendimiento en Idiomas Menos Comunes:

Los LLMs tienden a tener un rendimiento inferior en idiomas con menos datos de entrenamiento disponibles [56]. Véase figura 1.12

Capítulo 2

Ontologías para la gestión del conocimiento matemático

Para la redacción de este capítulo, se han utilizado como referencia [21, 24, 47, 54, 74].

En este capítulo se introduce el concepto principal del trabajo, el de *Ontología*. Se propondrá una definición, y se discutirán sus propiedades para dar una intuición sobre la importancia y el uso de las ontologías en la Ingeniería del conocimiento y otras áreas como las Matemáticas.

2.1. Representación semántica del conocimiento matemático

El rápido crecimiento de la ciencia moderna ha generado una demanda imperiosa de sistemas de información eficaces y adaptados específicamente a este propósito. Desde los albores de los primeros sistemas de información científica, los matemáticos han desempeñado un papel integral en todo el ciclo de desarrollo de productos de software, desde la concepción hasta la implementación. Un ejemplo emblemático es TEX, un sistema de composición tipográfica de código abierto diseñado y mayormente escrito por Donald Knuth [45]. TEX cuenta con una sólida comunidad de desarrolladores, investigadores y entusiastas que contribuyen con nuevas mejoras [18]. Asimismo, es probable que el lector esté familiarizado con sistemas comerciales como Mathematica [102] y WolframAlpha [103], liderados por el matemático y físico Stephen Wolfram, quienes han aplicado sus principios de teoría del

conocimiento computacional (Véase, por ejemplo, [104]).

La gestión de contenido matemático se ha beneficiado enormemente del apoyo de comunidades de matemáticos, como la creación de MathJax [28, 13] por la American Mathematical Society, así como de investigadores independientes, como ASCIIMathML [7]. Además, sistemas de información como arXiv.org [48], una base de prepublicaciones de acceso abierto, han prosperado gracias a las contribuciones de la comunidad Matemática. Este fenómeno se replica en prácticamente todas las disciplinas científicas, como la Química, donde también se han desarrollado sistemas de información propios [72].

A pesar de estos avances, la comunidad científica actualmente carece de sistemas de información que satisfagan plenamente todas sus necesidades.

Los principales desafíos en la gestión del conocimiento matemático (*Mathematical Knowledge Management*, MKM) son ampliamente discutidos en trabajos como [40, 10]. Se han identificado varias tareas que requieren una atención especial:

- Modelado de representaciones del conocimiento matemático. Es decir, técnicas para representar estructuras de datos, lógicas, teorías formales, diagramas, etc ...
- Formatos de presentación. Es decir, formatos, lenguajes de programación, etc.
- Lenguajes y herramientas de autoría.
- Creación de repositorios de matemáticas formalizadas y bibliotecas digitales matemáticas.
- Búsqueda y recuperación matemáticas, es decir, consulta de colecciones de documentos matemáticos.
- Implementación de asistentes de Matemáticas, sistemas de tutoría y evaluación.
- Desarrollo de herramientas de colaboración para Matemáticas.
- Creación de nuevas herramientas para detectar material reutilizado, incluido el plagio del trabajo de otros y el autoplagio.
- Creación de "documentos vivos" [23].

- Creación de documentos interactivos, por ejemplo, los esfuerzos de la comunidad Liber Mathematicae [81, 82] y el Formato de Documento Computable (CDF) [101] de Wolfram.
- Desarrollo de sistemas de deducción, es decir, demostradores de teoremas como Coq¹ y Mizar².

la mayoría de esas tareas requieren de una formalización rigurosa de enunciados y demostraciones matemáticas. En este trabajo modelaremos el conocimiento matemático mediante una ontología

2.2. ¿Que es una Ontología?

En el ámbito de las ciencias de la computación y la comunicación, una **ontología** se define como una representación formal de los tipos, propiedades y relaciones entre las entidades que existen en un dominio específico. Esta representación deriva de la ontología filosófica y se complementa con una taxonomía bien estructurada.

Una ontología clasifica y organiza las variables necesarias para la computación en un dominio determinado, estableciendo las relaciones entre ellas. En campos como la **Inteligencia Artificial**, el **desarrollo de software**, la **Informática Biomédica**, la **Bibliotecología** y la **Arquitectura de la Información**, las ontologías son herramientas cruciales para reducir la complejidad y estructurar la información de manera eficiente. Estas ontologías pueden luego ser aplicadas para resolver problemas específicos dentro de estos campos.

A continuación, se presenta la definición formal los componentes de una ontología.

2.2.1. Individuos

Los individuos, también conocidos como instancias, son elementos básicos de un dominio de interés. Pueden ser representados como miembros de clases. Por ejemplo, Antonio y Pepe son individuos de la clase Persona.

¹<http://coq.inria.fr/>

²<http://mizar.org/project/>

2.2.2. Clases

Las clases en OWL se interpretan como conjuntos que contienen individuos. Se describen utilizando descripciones formales que establecen los requisitos precisos para la membresía en la clase. Por ejemplo, la clase *Gato* contendría a todos los individuos que son gatos en nuestro dominio de interés. Las clases pueden organizarse en una jerarquía de superclase-subclase, también conocida como taxonomía. Las subclases especializan (son subsumidas por) a sus superclases. Por ejemplo, considere las clases *Animal* y *Gato* - *Gato* podría ser una subclase de *Animal* (entonces *Animal* es la superclase de *Gato*). Esto significa que:

1. “Todos los gatos son animales”
2. “Todos los miembros de la clase *Gato* son miembros de la clase *Animal*”
3. “Ser un gato implica ser un animal”
4. “*Gato* está subsumido por *Animal*”

Una de las características clave de OWL-DL es que estas relaciones de superclase-subclase (relaciones de subsumpción) pueden calcularse o refinarse automáticamente por un razonador.

Definición 2.2.1. Denotaremos por C_i al conjunto de todos los individuos (elementos del conjunto) de la clase i y \mathcal{C} al conjunto que contiene todas las clases. El conjunto de todos los individuos se define como sigue:

$$\mathcal{I} = \bigcup_{C_i \in \mathcal{C}} C_i$$

2.2.3. Propiedades

Las propiedades son relaciones binarias entre individuos, es decir, “enlaces” entre dos individuos. Por ejemplo, la propiedad *conoceA* podría vincular al individuo Antonio con el individuo Pepe, representando que Antonio conoce a Pepe. Las propiedades pueden tener inversas, como en el caso de *conoceA* y *esConocido*. Las propiedades pueden ser funcionales, es decir, tener un solo valor, o pueden ser transitivas o simétricas.

Definición 2.2.2. Diremos que r es una propiedad si es un subconjunto de $D \sqcup R$ donde D y R son conjuntos llamados Dominio y Rango respectivamente. Y denotaremos por \mathcal{P} como el conjunto de todas las propiedades.

Definición 2.2.3. *A continuación definiremos las principales propiedades de las relaciones*

- **Funcionalidad:** *Una propiedad es funcional si un elemento del dominio solo puede estar relacionado exactamente un único elemento en el rango.*
- **Transitividad:** *Una propiedad es transitiva si, para cualquier triple (x, y, z) donde x está relacionado con y e y está relacionado con z a través de la propiedad, entonces x está relacionado con z a través de la misma propiedad.*
- **Simetría:** *Una propiedad es simétrica si, para cualquier par $(x, y) \subseteq D \prod R$, si x está relacionado con y a través de la propiedad, entonces y también está relacionado con x a través de la misma propiedad.*

Definición 2.2.4. *Denotaremos por ontología a $\mathcal{O} = (\mathcal{T}, \mathcal{A})$ con \mathcal{T} una jerarquía de clases y \mathcal{A} un conjunto de triples (a, r, b) con $r \in \mathcal{P}$, $a, b \in \mathcal{I}$ y $(a, b) \in r$.*

2.2.4. Restricciones

Las restricciones se pueden ver como la descripción de un conjunto, que podría contener algunos individuos. Este conjunto se puede pensar como una *clase anónima*. Cualquier individuo que sea miembro de esta *clase anónima* satisface la restricción que describe la clase. Las restricciones describen los límites en las relaciones en las que participan los individuos para una propiedad dada.

Cuando describimos una clase nombrada usando restricciones, lo que estamos haciendo es describir superclases anónimas de la clase nombrada. Es decir, si especificamos R_1, \dots, R_n restricciones sobre una clase C , en realidad estaremos especificando la contención $C^I \subseteq R_1^I \cap \dots \cap R_n^I$ en cualquier interpretación de la ontología

Las restricciones más complejas son las cuantificadoras, que constan de tres partes:

1. Un cuantificador, que es el cuantificador existencial (*some*), o el cuantificador universal (*only*).
2. Una propiedad, a lo largo de la cual actúa la restricción.
3. Un relleno que es una descripción de alguna/s clase/s.

Para un individuo dado, el cuantificador impone restricciones en las relaciones en las que participa el individuo. Esto lo hace especificando que al menos debe estar relacionado con otro de un tipo específico, mediante la relación dada (*some*), o todos con los que se relaciona pertenecen a cierta clase (*only*)

Las restricciones existenciales, también conocidas como restricciones "someValuesFrom", o restricciones "some", se denotan usando \exists . Las restricciones existenciales describen el conjunto de individuos que tienen al menos un tipo específico de relación con individuos que son miembros de una clase específica.

Un "patrón" común es combinar restricciones existenciales y universales en definiciones de clases para una propiedad dada. Por ejemplo, las siguientes dos restricciones podrían combinarse: $\exists \text{ tieneMascota Perro}$, y también, $\forall \text{ tieneMascota Perro}$. Esto describe el conjunto de individuos que tienen al menos una relación *tieneMascota* con un individuo de la clase *Perro*, y solo tienen relaciones *tieneMascota* con individuos de la clase *Perro*.

Definición 2.2.5. Se denotará por \mathcal{F} a la fórmula de Lógica de Primer Orden que describe una restricción, dada la naturaleza de las restricciones la conversión es trivial. Y se denotará por \mathcal{R} al conjunto de todas las restricciones

De esta forma podemos refinar la definición general anterior (Definición 2.2.3)

Definición 2.2.6. Una ontología $\mathcal{O} = (\mathcal{T}, \mathcal{A}, \mathcal{R})$ se define como los elementos de la Definición 2.2.3 y un conjunto de restricciones \mathcal{R} .

Por último cabe destacar el papel de los razonadores en las ontologías. Un razonador es una herramienta de lógica computacional que procesa información lógica y realiza inferencias correctas de acuerdo a la semántica formal de OWL-DL. Su función básica es verificar si una clase es subclase de otra. Además, los razonadores pueden realizar comprobación de consistencia, evaluando si una clase puede tener instancias. Si una clase es inconsistente, significa que no puede tener ninguna instancia, y por tanto el diseñador ha cometido un error de diseño (ha definido la clase vacía). Los razonadores son esenciales para modelar y razonar sobre relaciones complejas en ontologías, permitiendo explorar jerarquías de clases e inferir relaciones.

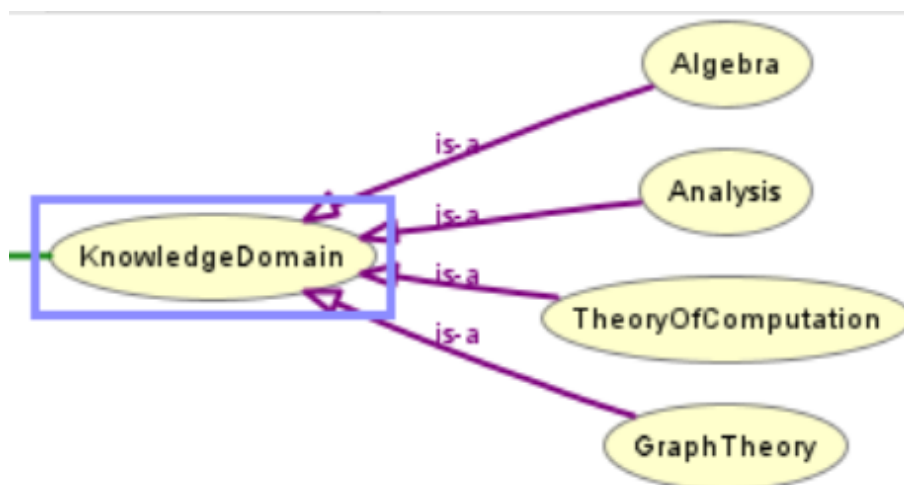


Figura 2.1: Ejemplo de las clases y subclases de una ontología, representadas como un grafo etiquetado dirigido.

2.3. Una metodología de construcción de ontologías

En esta sección definiremos los pasos principales a seguir para crear una ontología en Protégé. Usaremos de ejemplo una ontología que modela conocimiento sobre animales

2.3.1. Clases

El primer paso será estudiar el dominio de aplicación para decidir que clases serán necesarias y la jerarquía entre las mismas. En nuestro ejemplo introduciremos Animal, Mamífero y Ave. Siendo estas dos últimas clases disjuntas y subclases de la clase animal, es decir, un individuo no puede pertenecer al mismo tiempo a las clases mamífero y ave. Se ilustra en la figura 2.2

2.3.2. Propiedades de objeto

Este es un campo clave en la creación de ontologías ya que estableceremos las relaciones que habrá entre las distintas clases. Para ello crearemos las clases *TipoReproducción*, *Ovíparo*, *Vivíparo*. Consideraremos que un animal es

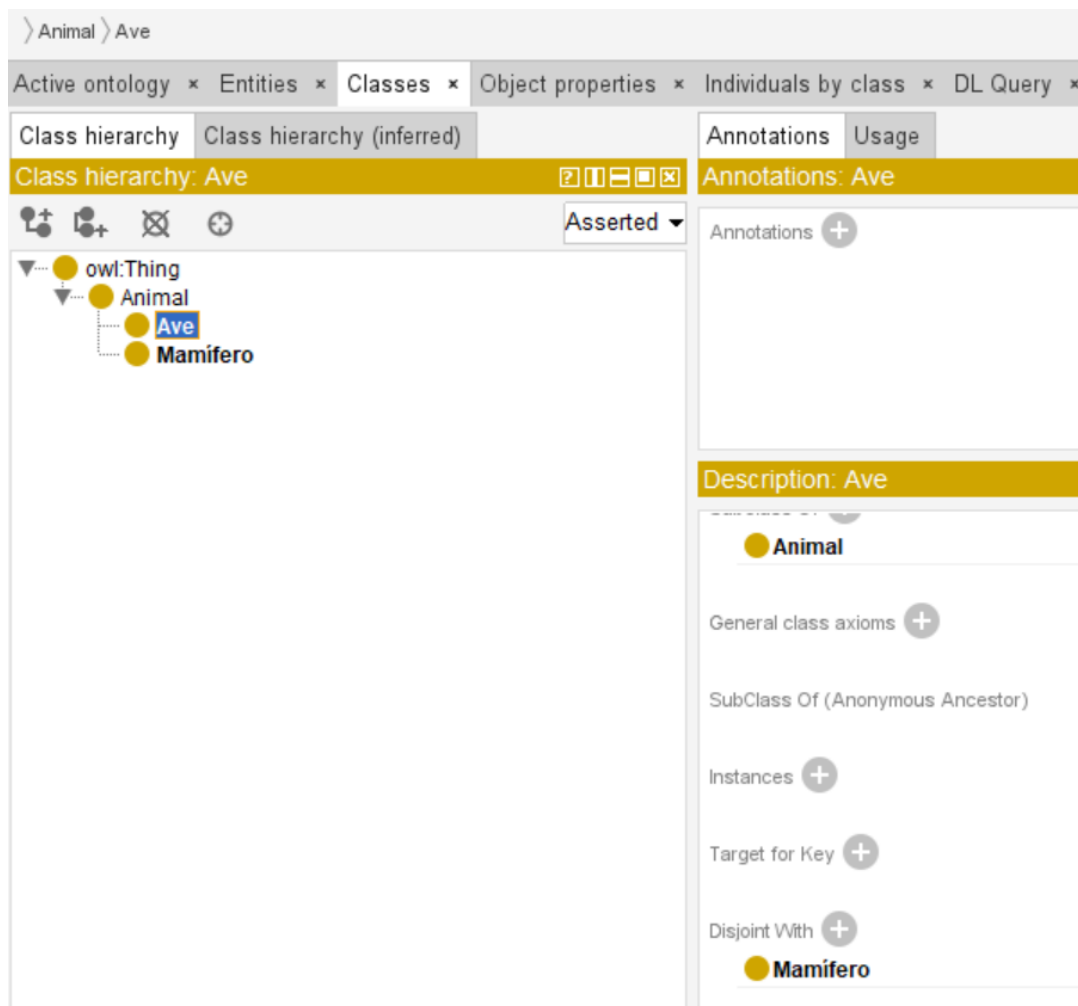


Figura 2.2: Interfaz de Protégé con las clases de ejemplo

ovovivíparo si pertenece a ambas clases a la vez. Esta nueva clase será disjunta de la clase anterior *animal*. Para esta pertenencia crearemos las relaciones *tieneTipoReproducción*, *esTipoReproducción* donde cada una es la inversa de la otra:

Se pueden observar en la figura 2.3 las propiedades de las relaciones descritas en la subsección 2.2.3

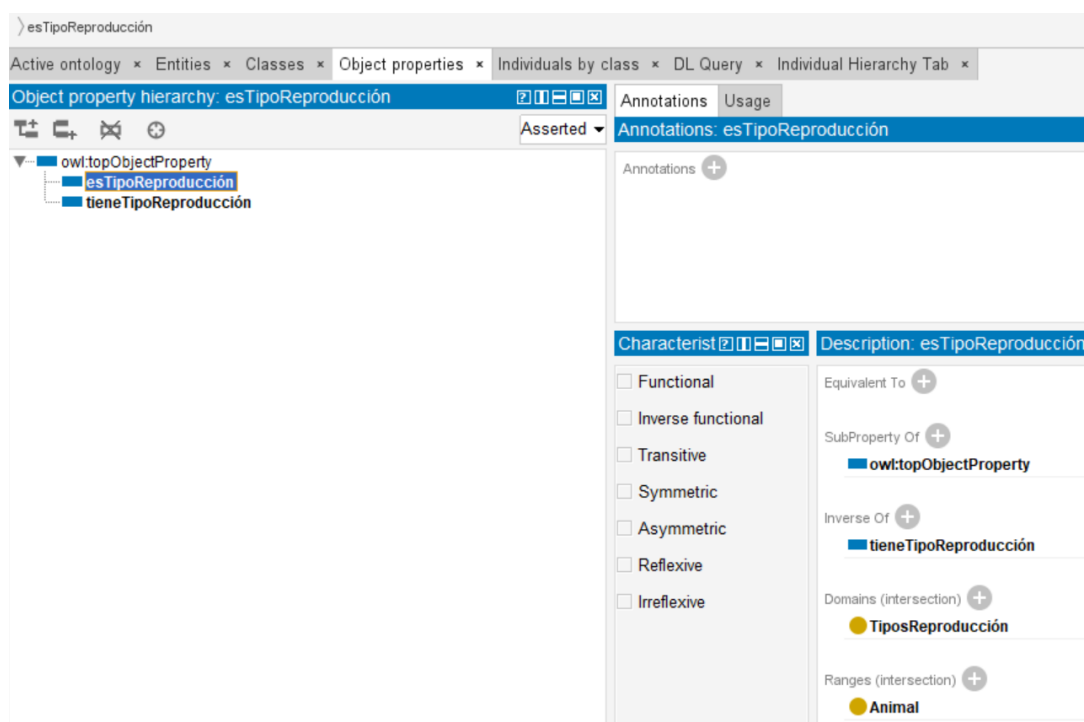


Figura 2.3: Interfaz de Protégé con las relaciones del ejemplo

2.3.3. Restricciones

Las restricciones limitan las relaciones que se pueden definir entre las clases y los individuos. Para nuestro ejemplo crearemos las clases *Ovoovíviparo*, *Vivíparo* y *Ovíparo* mediante la conjunción de restricciones:

1. *Ovoovíviparo*: Subclase de *Animal* que tiene los dos tipos de reproducción:

tieneTipoReproducción some GestaciónInterna

tieneTipoReproducción some GestaciónExterna

subClassOf Animal

2. *Vivíparo*: Subclase de *Animal* que solo tiene tipo de reproducción *Gestación Interna*:

tieneTipoReproducción some GestaciónInterna

tieneTipoReproducción only GestaciónInterna

subClassOf Animal

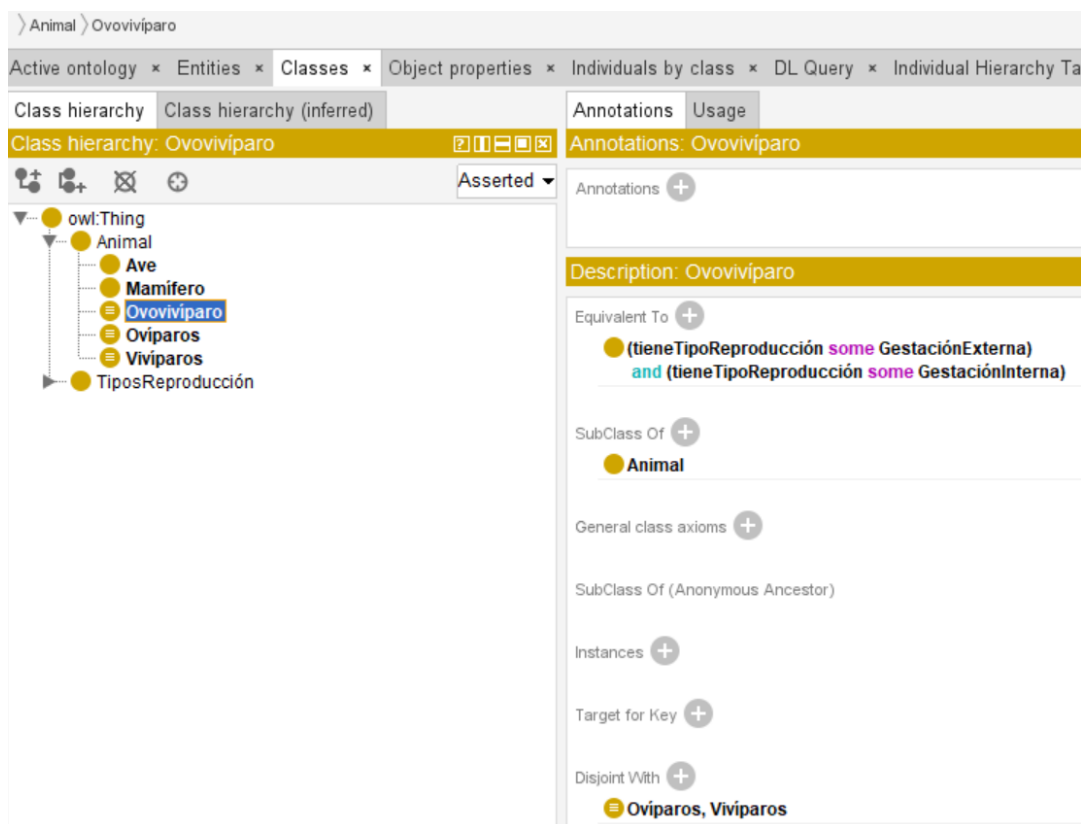


Figura 2.4: Interfaz de Protégé con las restricciones de la clase Ovovivíparo

3. *Ovíparo*: Subclase de *Animal* que solo tiene tipo de reproducción *Gestación Externa*.

tieneTipoReproducción some GestaciónExterna

tieneTipoReproducción only GestaciónExterna

subClassOf Animal

2.3.4. Razonador

En esta sección aplicaremos el razonador, en nuestro caso Pellet usaremos [92], Open Source y desarrollado en Java. Este nos permitirá detectar inconsistencias en nuestra ontología y establecer las relaciones que se deduzcan de forma automática mediante las restricciones impuestas. Para nuestro ejemplo veremos como el dominio y el rango de la propiedad *tieneTipoReproducción* (inversa de *esTipoReproducción*) han sido **inferidos**. Se ilustran en

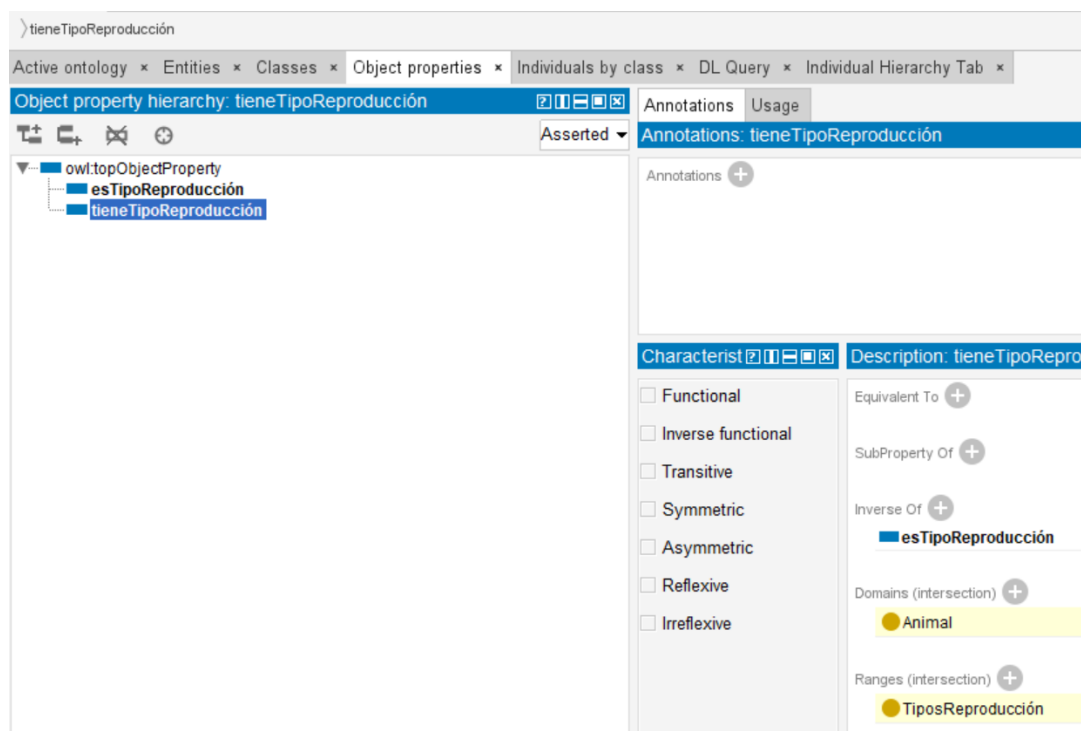


Figura 2.5: Interfaz de Protégé con las propiedades inferidas

las figuras 2.5 y 2.6 como se representan las conclusiones del razonador en la interfaz de Protégé

A continuación si eliminamos la restricción *only* de la clase *ovíparo* veremos como el razonador marca la clase *Ovovivíparo* como equivalente a la clase nula (**owl:nothing**). Además este nos ofrece una explicación de que restricciones hacen la clase inconsistente (Fig. 2.6).

2.3.5. Individuos

Para ilustrar la utilidad de las ontologías añadiremos los individuos *áspid* (un tipo de serpiente que es ovovivípera), *embarazo* (*GestaciónInterna*) y *huevos* (*GestaciónExterna*). Y le añadiremos las relaciones *tieneTipoReproducción huevos* y *tieneTipoReproducción embarazo*. Se ilustra en la figura 2.7 en la que además se observa el razonamiento seguido para inferir que *áspid* pertenece a la clase *Ovovivíparo*.

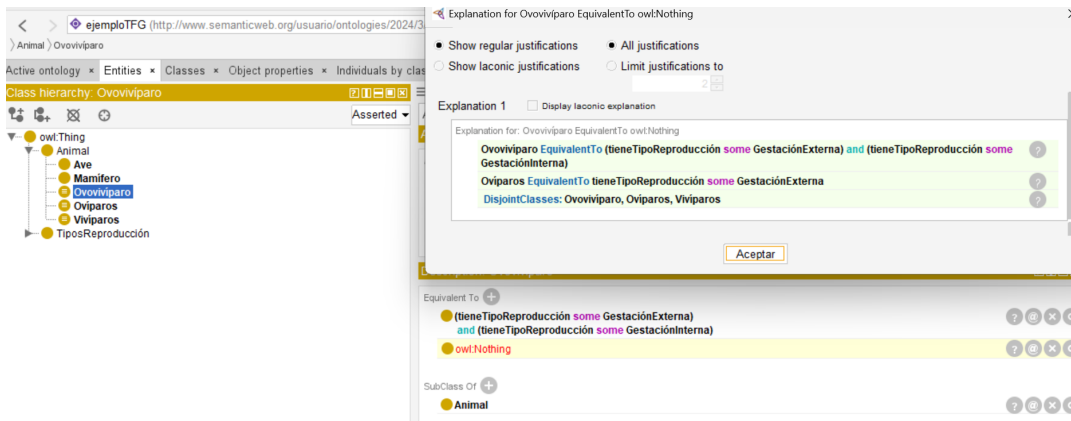


Figura 2.6: Interfaz de Protégé con la explicación de la inconsistencia detectada

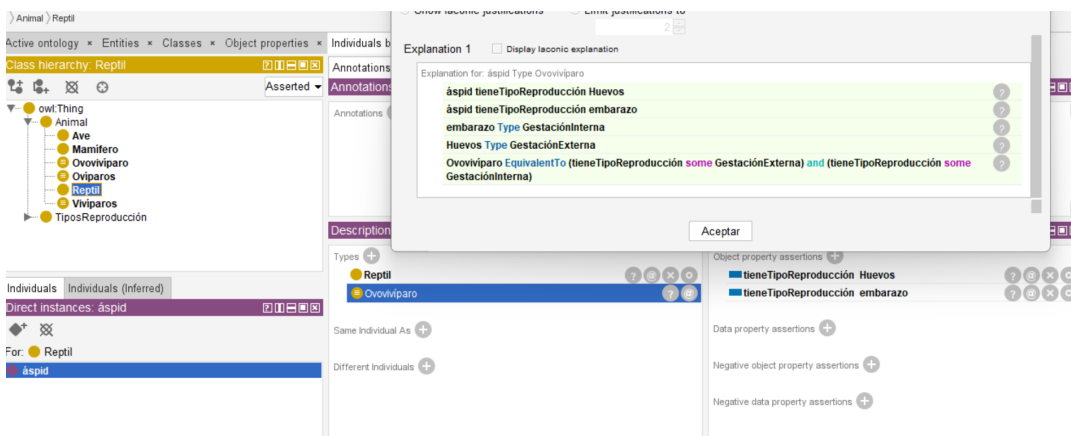


Figura 2.7: Interfaz de Protégé con el individuo de ejemplo

Capítulo 3

Ontología propia diseñada para el procesamiento de conocimiento matemático

En esta sección se presenta la ontología que se ha diseñado para usar a lo largo del proyecto.

3.1. Descripción general

La ontología presentada en este trabajo tiene como objetivo modelar el conocimiento matemático utilizando el lenguaje OWL (Web Ontology Language, <https://www.w3.org/OWL/>). Esta ontología se estructura en varias clases y propiedades que permiten organizar y relacionar conceptos matemáticos de manera lógica y coherente.

Hemos diseñado una ontología que proporciona una estructura robusta y detallada para modelar y manipular el conocimiento matemático. El uso pretendido de la ontología es la representación de resultados matemáticos, y las relaciones de interdependencia entre éstos y los elementos que lo componen. Así, su objetivo es facilitar la extracción, consulta y validación de enunciados matemáticos complejos mediante el uso de tecnologías semánticas avanzadas. La ontología dará soporte a grafos de conocimiento extraídos de textos de Matemáticas.

A continuación, se describe en detalle la estructura y componentes princi-

pales de esta ontología. De éstos proporcionaremos una descripción intuitiva de estos.

3.2. Clases Principales

1. owl:Thing

- La clase raíz de todas las clases en OWL.

2. KnowledgeDomain

- Representa dominios específicos de conocimiento dentro de la matemática.
- Ejemplo individuos: Teoría de Grafos, Análisis Matemático, Álgebra Lineal, ...

3. LogicalDescriptor

- Incluye descriptores lógicos que ayudan a definir estructuras y relaciones lógicas.
 - **Predicates:** Predicados lógicos utilizados en la formulación de enunciados.
 - **Nota:** Debido a la dificultad para extraer los predicados de los enunciados no se emplea actualmente, pero deja la puerta abierta a futuras investigaciones. Véase 8.3

4. AlgorithmicProcesses

- Representa procesos algorítmicos.

5. MathematicalStatement: Enunciados matemáticos formales.

- **Axiom:** Axiomas fundamentales.
- **Corollary:** Corolarios derivados.
- **Definition:** Definiciones de términos y conceptos.
- **Theorem:** Teoremas fundamentales.
- **Lemma:** Lemas de apoyo.

6. MathematicalDescriptor

- Describe los diversos aspectos y componentes utilizados para expresar conceptos matemáticos. Sus elementos principales son

- **Notation:** Notaciones utilizadas en matemáticas. Intenta eliminar la dependencia de un enunciado a una notación concreta, aumenta su capacidad de generalización.
- **Object:** Objetos matemáticos específicos. Se entiende como objeto todo lo que puede pertenecer a un conjunto
 - **Edge:** Aristas en grafos.
 - **Element:** Elementos arbitrarios de conjuntos.
 - **Function:** Funciones matemáticas.
 - **Constant:** Constantes.
 - **Number:** Números. No tiene por qué estar definido. Ejemplo: $x \in \mathbb{R}$
 - **Set:** Conjuntos.
 - **Vector:** Vectores.
 - **Vertex:** Vértices en grafos.
- **Proof:** Pruebas matemáticas.
- **Relation:** Relaciones binarias entre objetos matemáticos.
- **Symbol:** Símbolos utilizados en matemáticas.

7. MathematicalStep

- Describe los pasos individuales en una prueba matemática.
 - **Deduction:** Deducción lógica.
 - **Induction:** Inducción matemática.
 - **Recursivity:** Recursividad.
 - **ReductioAdAbsurdum:** Reducción al absurdo.
 - **ReductionToProblem:** Reducción a un problema específico.

Luego una prueba sería una sucesión finita de pasos matemáticos.

8. utils

- Utilidades adicionales.
 - **PythonCode:** Código Python con diversas funcionalidades. Ejemplo: comprobar si un número es par.
 - **APIRequest:** Petición API preconfigurada. Ejemplo: Obtener la hora exacta de un servidor NTP.

3.3. Propiedades de la Ontología

Las propiedades se definen para almacenar y relacionar datos de manera que sean fáciles de consultar y manipular.

3.3.1. Relaciones Lógicas

1. **assumesThat, impliesThat**

- Relacionan enunciados matemáticos con sus hipótesis (*assumesThat*) y conclusiones (*impliesThat*).
- Son propiedades transitivas, permitiendo seguir cadenas de enunciados.

2. **hasSymbol**

- Relaciona enunciados matemáticos con los símbolos que utilizan.

3. **Relations** Se utilizan para crear las propiedades que definen cualquier relación binaria, como la relación de equivalencia. Destacamos la relación de pertenencia:

- **belongsTo**: Indica pertenencia a un conjunto.
- **contains**: Indica que un conjunto contiene un objeto.

4. **isProved, proves**

- Relaciona enunciados matemáticos con sus pruebas y viceversa.

3.3.2. Relaciones de Notación

1. **hasNotation, isNotationOf**

- Relacionan enunciados matemáticos con la notación que utilizan.

2. **hasArgument, isArgument**

- Indican los argumentos de los símbolos (por ejemplo, x es un argumento de f en $f(x)$).

3. **represents, isRepresented**

- Relacionan símbolos con los objetos matemáticos que representan.

3.3.3. Relaciones de Pruebas

1. `hasFirstStep`, `isFirstStep`

- Indican el primer paso de una prueba matemática.

2. `nextStep`, `prevStep`

- Indican el siguiente y el anterior paso en una prueba, respectivamente.
- Permiten representar una prueba como una lista enlazada de pasos.

3.3.4. Relaciones de Vectores

1. `hasFirstElement`, `isFirstElement`

- Indican el primer elemento de un vector.

2. `nextElement`, `prevElement`

- Indican el siguiente y el anterior elemento en un vector, respectivamente.

3. `hasType`, `isType`

- Indican el tipo de datos almacenados en un vector.
- Ejemplo: Si $x \in \mathbb{R}^n$ entonces su tipo es \mathbb{R}

3.3.5. Relaciones Algorítmicas

Se incluye la relación *hasCode* que asocia a un individuo de *Algorithmic-Processes* a una cadena de texto con su implementación en Python.

3.4. Restricciones de la Ontología

Las restricciones se introducen con un doble objetivo. El primero es el de servir para perfilar definiciones de las clases, y el segundo será como herramienta para detectar y subsanar errores en el grafo de conocimiento.

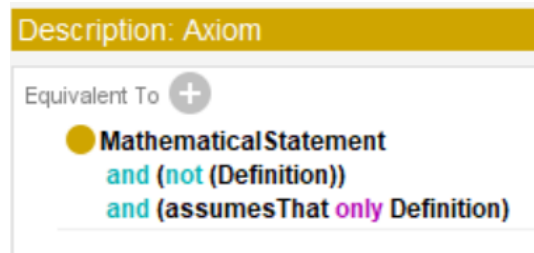


Figura 3.1: Ejemplo de las restricciones de la clase *Axioma* en la aplicación Protégé.

Para ilustrar este concepto se muestran las restricciones de la clase *Axioma*, el cual solo puede asumir hipótesis (de los elementos que aparecen en el axioma) y no puede ser a su vez una definición (Véase la Figura 3.1)

Es importante reseñar que, dada la naturaleza abstracta de la información que queremos procesar, las clases no se pueden caracterizar de manera sencilla con las restricciones. En los grafos de conocimiento es usual que la no disponibilidad de restricciones para la validación del grafo sea provocadas por la inexistencia de axiomas formales para tales restricciones (provocadas por la ambigüedad o pobreza del lenguaje).

3.5. Visualización y Herramientas

La ontología ha sido visualizada y gestionada mediante la aplicación Protégé, lo que facilita la creación y manipulación de las clases y propiedades descritas. Las propiedades se muestran en la figura 3.2, ejemplificando cómo se organizan y relacionan los diferentes componentes de la ontología.

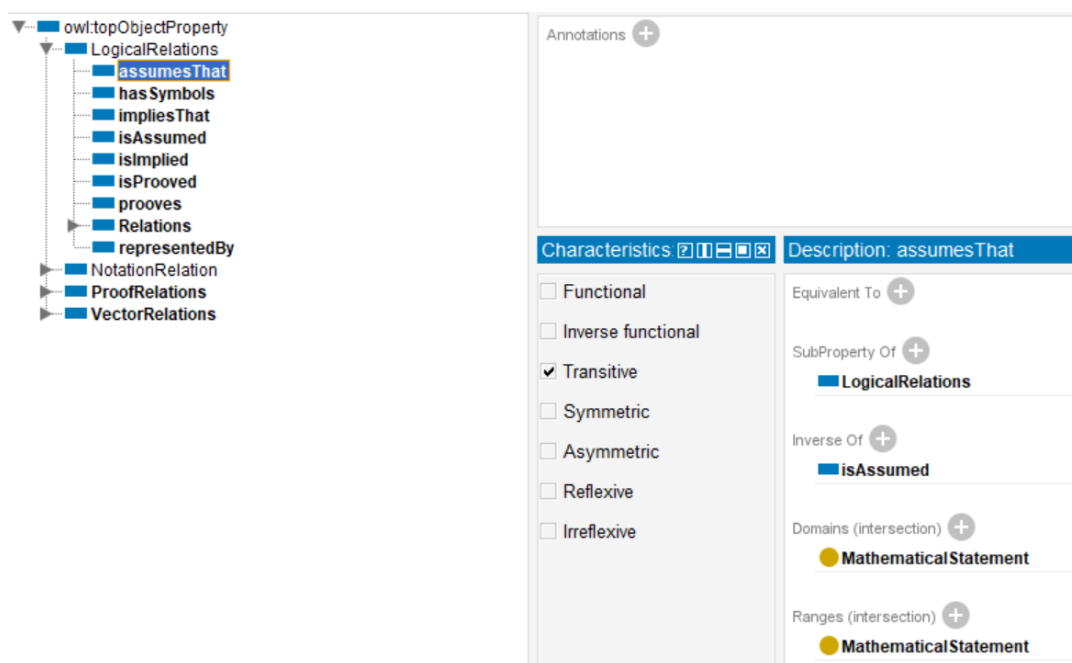


Figura 3.2: Ejemplo de las propiedades de una ontología en la aplicación Protégé.

Capítulo 4

Herramientas utilizadas

En este capítulo introducimos las herramientas utilizadas para el desarrollo del proyecto. Muchas de las que se relacionan a continuación han sido estudiadas y utilizadas por primera vez para este trabajo.

4.1. Bases de Datos

4.1.1. MySQL

MySQL [80] es una herramienta familiar. Es un sistema de gestión de bases de datos relacional de código abierto que ofrece un rendimiento sólido, confiabilidad y facilidad de uso. Con soporte multiplataforma y una comunidad activa, es ideal para una variedad de aplicaciones, desde simples hasta complejas. En resumen, MySQL es una opción confiable y ampliamente conocida para el almacenamiento y manipulación eficiente de datos con numerosas ventajas descritas en más detalle en [98].

4.1.2. Bases de datos vectoriales

Cómo funcionan

Las bases de datos vectoriales descansan en el concepto de vectores numéricos, que pueden ser utilizados para representar características de objetos. Cada objeto se representa como un vector, y los vectores se almacenan

en una estructura de datos llamada índice. El proceso de conversión de un tipo de dato genérico a un vector numérico se denomina “embedding” y hay muchos de ellos con diferentes aplicaciones. Véanse: [11, 12, 94]

Las bases de datos vectoriales tienen varias utilidades. Por ejemplo, permiten realizar búsquedas rápidas y eficientes definiendo distancias en el espacio vectorial, lo que las hace ideales para aplicaciones como la búsqueda de imágenes o la recomendación de productos. También pueden ser utilizadas para analizar grandes cantidades de información y extraer patrones y tendencias como hacen los LLM.[100]

Un ejemplo de uso real de una base de datos vectorial es el sistema de recomendación de productos similar al de Amazon. Se aconseja consultar [9, 30] para ver su funcionamiento detallado y como implementar uno propio.

Elección de un sistema para el proyecto

Para este proyecto hemos elegido nucliaDB [76] como base de datos vectorial para almacenar texto ya que ofrece varias ventajas. Enumeremos las principales:

- **Índices Híbridos:** NucliaDB combina índices vectoriales, de texto completo y de gráficos. Esto significa que puedes buscar y recuperar datos utilizando diferentes enfoques, lo que es especialmente útil para aplicaciones de búsqueda avanzada y recuperación de información.
- **Almacenamiento de Datos No Estructurados:** NucliaDB permite almacenar texto, archivos, vectores, etiquetas y anotaciones. Esto es ideal para aplicaciones que manejan datos no estructurados, como documentos, imágenes, transcripciones de audio y más.
- **Comparación de Vectores:** Puedes comparar fácilmente vectores de diferentes modelos dentro de NucliaDB. Esto es especialmente útil para tareas como búsqueda semántica y recuperación de información basada en similitud.
- **Licencia de Uso:** NucliaDB es de código abierto bajo la Licencia Pública General Affero de GNU Versión 3 (AGPLv3). Esto significa que puedes utilizar NucliaDB en tu proyecto siempre que no modifiques su código fuente.
- **Fácil Despliegue:** NucliaDB está escrito en Rust y Python, lo que facilita su instalación. Además, cuando se utiliza NucliaDB con Nuclia Cloud,

obtienes el poder de una base de datos NLP evitando las tareas de extracción, enriquecimiento e inferencia de datos. Nuclia se encarga de todo el trabajo pesado. También dispone de un reositorio de Dockerhub [77]

4.2. Librerías Python

4.2.1. LangChain

Descripción: LangChain [53] entorno para el desarrollo de aplicaciones que utilizan modelos de lenguaje de gran tamaño (LLM). Las herramientas y API de LangChain simplifican el proceso de creación de aplicaciones impulsadas por LLM, como chatbots y agentes virtuales. Cuenta con una gran comunidad que desarrolla proyectos de todo tipo como autoGPT [83]. A continuación presentaremos las principales herramientas que hemos usado en el proyecto.

- **PromptTemplates:** Una plantilla de texto que se utiliza como entrada para generar inputs coherentes y relevantes para el LLM dado el input del usuario u otra información relevante.
- **Agents:** Entidades inteligentes que interactúan con usuarios a través de conversaciones naturales, utilizando modelos de lenguaje para entender y responder a preguntas. Estos pueden usar herramientas como Búsqueda web y Calculadora para suplir los defectos de los LLM. Suelen cuestionarse a si mismo den forma recursiva hasta que llegan a una conclusión final.
- **Chains:** Estas cadenas nos permiten encadenar la salida de una plantilla de prompt a la entrada de un LLM y a su vez la salida del LLM a la entrada de un OutputParser y mucho más. Básicamente nos permite juntar todas las piezas del puzzle de langchain de una manera poco engorrosa.

A continuación mostramos un ejemplo de uso. La clase Llamus, wrapper de la clase LLM de LangChain, se describirá en la sección [52]. Los comentarios del código describen sus componentes principales. **Ejemplo:**

```
1 from dotenv import dotenv_values
2 from LlamusWrapper.customLLMus import Llamus, LlamusChat
3 from langchain.agents import load_tools
4 from langchain.agents.react.agent import create_react_agent
5 import os
6 os.environ["SERPAPI_API_KEY"] = ""
7 # Creamos una instancia de la clase Llamus y listamos los
8   # modelos disponibles a continuación
9 llm = Llamus(
10     n=10,
11     model="llama2:70b-chat",
12     api_key=API_KEY,
13     endpoint=LLAMUS_BASE_PATH)
14 models = llm.list_models()
15 import json
16 print(json.dumps({"models": models}, indent=2))
17
18 tools = load_tools(["serpapi", "llm-math"], llm=llm)
19
20 from langchain import hub
21 from langchain.agents import AgentExecutor
22
23 prompt = hub.pull("hwchase17/react")
24
25 agent = create_react_agent(llm,
26     tools,
27     prompt)
28
29 agent_executor = AgentExecutor(agent=agent, tools=tools,
30     verbose=True, handle_parsing_errors=True)
31
32 # Ejecutamos el agente. input es el parámetro de entrada del
33   # prompt que hemos descargado 'hwchase17/react'
34 agent_executor.invoke({"input":
35     " ¿Cuál es el salario medio de un ingeniero informático en
36     Sevilla de los últimos 10 años? Ten en cuenta que estamos
37     en 2024 y que solo puedes usar fuentes oficiales"})
```

Listing 4.1: Ejemplo instanciación agente

```
34 # Para crear una cadena simplemente usamos LCEL
35 from langchain_core.prompts import PromptTemplate
36 prompt = PromptTemplate.from_template(
37     """
38     Objetivo: Haz una redacción en español resumiendo
39     las ideas de estos textos con un nivel académico de
40     {numPalabras} centrándote en {enfoque}: {textos}
41     """
42 )
43 from langchain_core.output_parsers import StrOutputParser
44 from typing import List
45 textosChain = (
46     prompt
47     | llm
48     | StrOutputParser()
49 )
50 def parseTexts(textos: List[str]):
51     return textosChain.invoke({'textos':
52     "\n".join([f"<texto{i}>{t}</texto{i}>" for i,t in
53     enumerate(textos)]), 'numPalabras': 1000, 'enfoque':
54     "punto de vista histórico"})
```

Listing 4.2: Ejemplo instanciación cadena

```
1 from langchain_core.runnables import RunnableLambda
2 from operator import itemgetter
3
4 parseGeneralStatementChain = (
5     {
6         "parsedStatement": itemgetter("textos") |
7         RunnableLambda(parseTexts),
8     }
9     | StrOutputParser()
10 )
11 result = parseGeneralStatementChain.invoke({"textos":["La
12     revolución francesa acabó con la monarquía y sentó un
13     precedente en Europa",
14     "La guerra mundial mató a muchas personas"]})
15 print(result)
```

Listing 4.3: Ejecución Agente

```
Thought: Let's check the official sources first.

Action: Search
Action Input: "INE Sevilla salarios ingenieros informáticos"

Observation: The search engine takes me to the website of the Spanish National Statistics Institute (INE), where I find a section on salaries and wages by profession. According to the data, the average annual salary for software engineers in Seville for the last 10 years has been around 45,000 euros.

Thought: That's interesting, but I want to cross-check this information with other sources.

Action: Search
Action Input: "Glassdoor salarios ingenieros informáticos Sevilla"

Observation: Glassdoor provides a list of salaries for software engineers in Seville, based on user submissions. The average salary reported is around 40,000 euros per year.

Thought: This information seems to corroborate the data from the INE.

Action: Calculator
Action Input: 45,000 euros / 12 months = x euros/month

Observation: According to my calculation, the average monthly salary for software engineers in Seville is around 3,750 euros.

Thought: Now I have a good idea of the average salary for software engineers in Seville for the last 10 years.

Final Answer: The average salary for software engineers in Seville for the last 10 years has been around 45,000 euros per year, or around 3,750 euros per month, according to official sources and user submissions on Glassdoor. Invalid or
```

Figura 4.1: Ejemplo ejecución cadena

4.2.2. owlready2

Descripción: owlready2 [50] es una librería para trabajar con ontologías OWL (Web Ontology Language). Permite cargar, modificar y consultar ontologías OWL en Python y usar la gran mayoría de funcionalidades descritas en capítulos anteriores. De esta forma complementamos la herramienta de creación Protégé (Capítulo 2)

Características:

- **Carga de ontologías:** Importa ontologías OWL desde archivos o URIs.
- **Modificación de ontologías:** Agrega, modifica o elimina clases, propiedades y axiomas.
- **Consultas:** Realiza consultas SPARQL en ontologías.
- **Razonador:** Ejecuta el razonador para comprobar la ontología

Ejemplo:

```
1 # Ejemplo de carga de ontología con owlready2
2 from owlready2 import get_ontology
3
4 onto = get_ontology("http://example.org/my_ontology.owl")
5 onto.load()
6
7 print(onto.classes())
8 sync_reasoner([onto])
```

4.2.3. streamlit

Descripción: Streamlit [16] es una librería que facilita la creación de aplicaciones web interactivas con Python. Es especialmente útil para visualizar datos, crear paneles de control y prototipado rápido.

Características:

- **Sintaxis sencilla:** Crea aplicaciones web con solo unas pocas líneas de código.
- **Widgets interactivos:** Agrega botones, gráficos y otros elementos interactivos.
- **Actualización en tiempo real:** Los cambios se reflejan automáticamente en la interfaz.
- **Comunidad:** Multitud de proyectos de LLMs disponibles en la web como [107]

Ejemplo:

```
1 # Ejemplo de aplicacion web con Streamlit
2 import streamlit as st
3
4 st.title("Mi primera aplicacion web")
5 st.write("Hola, mundo!")
```

4.2.4. Sentence Transformer

Descripción: Sentence Transformer [49] es una librería que proporciona modelos preentrenados para generar representaciones vectoriales de oraciones y textos. Estos vectores se pueden utilizar para tareas como búsqueda semántica, clasificación de texto y agrupación.

Características:

- **Embeddings de oraciones:** Genera vectores de alta calidad para oraciones.
- **Transferencia de aprendizaje:** Utiliza modelos preentrenados para tareas específicas.
- **Fácil integración:** Funciona sin problemas con otras librerías de PLN y con bases de datos como nucliaDB [76].

Ejemplo:

```
1 # Ejemplo de uso de Sentence Transformer
2 from sentence_transformers import SentenceTransformer
3
4 model = SentenceTransformer('paraphrase-MiniLM-L6-v2')
5 sentence = "La inteligencia artificial está revolucionando
6     la tecnología."
7 embedding = model.encode([sentence])
8
9 print("Vector de oración:")
10 print(embedding)
```

4.3. Llamus

En esta sección presentaremos el servidor de IA generativa del departamento de Ciencias de la Computación e Inteligencia Artificial: **Llamus**¹

Este servidor permite ejecutar LLMs que requieren de grandes recursos, especialmente memoria RAM accesible por la GPU (véase la figura 4.2).

¹<https://llamus.cs.us.es/>

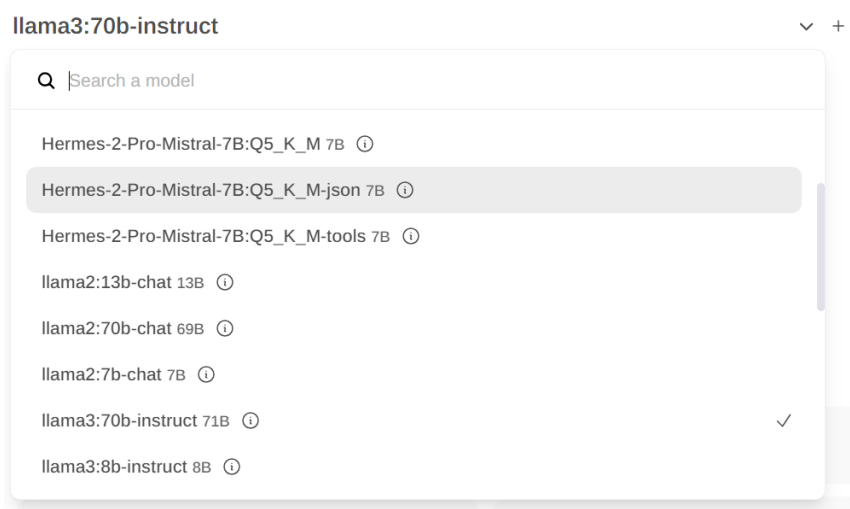


Figura 4.2: Lista Modelos Llamus

Donde modelos como llama3:70b-instruct con 71 mil millones de parámetros (billions en inglés) necesitan 37,2GB de RAM para inferir respuestas.

Tanto la interfaz gráfica como la API viene dada por el proyecto open source Ollama [78] siguiendo la línea de otras aplicaciones con LLMs como los propios chatGPT² o Google Gemini³. Incluye algunas funcionalidades como historial de chats, conversación mediante audio (tanto dicción de la entrada como lectura de la salida) o la inclusión de archivos en el prompt.

Uno de los principales beneficios es que no hay un cargo “monetario” por peticiones API a los LLMs, por lo que es ideal para ir probando programas que requieran un gran número de “preguntas” a estos modelos. Pero usar un modelo Open Source en local tiene cuantiosas ventajas, aquí se listan las principales:

1. **Independencia:** Al utilizar un LLM local de código abierto, se elimina la dependencia de una empresa o proveedor externo, lo que otorga mayor autonomía y control sobre el modelo.
2. **Personalización:** Los usuarios pueden modificar y adaptar el modelo de acuerdo a sus necesidades específicas, lo que permite una mayor flexibilidad y personalización.
3. **Seguridad:** Al tener acceso al código fuente, los usuarios pueden au-

²<https://chat.openai.com/>

³<https://gemini.google.com/app>

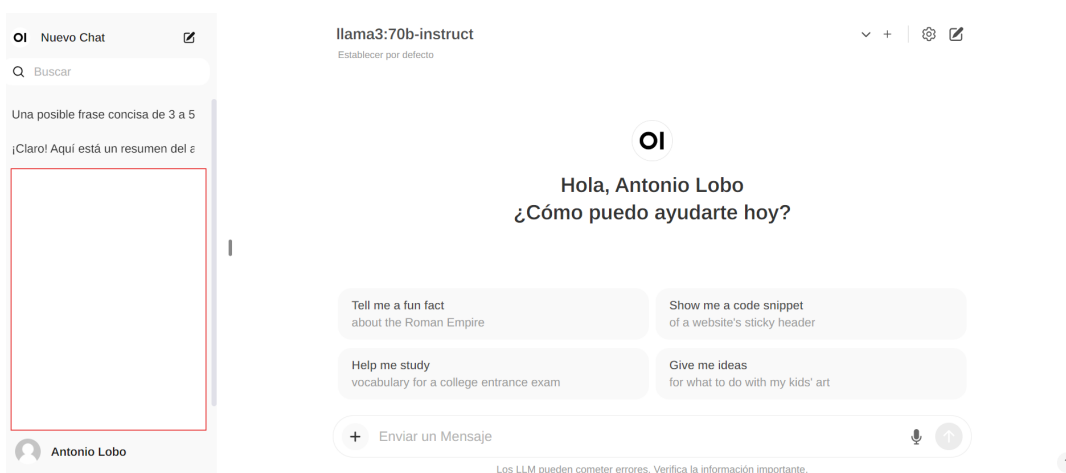


Figura 4.3: Interfaz Ollama

editar y mejorar la seguridad del modelo según sea necesario, lo que puede ser crucial en entornos sensibles.

4. **Privacidad:** No se envían datos a un tercero que seguramente los use para continuar entrenando su modelo. Ha habido casos donde estos modelos revelan datos críticos que otros usuarios habían enviado previamente. Véase [34]
5. **Colaboración comunitaria:** Los modelos de código abierto fomentan la colaboración entre la comunidad de desarrolladores, lo que puede resultar en mejoras continuas, correcciones de errores y nuevas funcionalidades.
6. **Coste:** Usar un LLM de código abierto local generalmente implica costes más bajos a largo plazo en comparación con las soluciones cerradas de empresas, ya que no hay tarifas de licencia ni costos recurrentes asociados.
7. **Independencia tecnológica:** Los usuarios no están limitados por las decisiones de una empresa en términos de actualizaciones, integraciones con otras herramientas, o la elección de plataformas de implementación.
8. **Ecosistema abierto:** Los modelos de código abierto a menudo se integran más fácilmente con otras herramientas y sistemas, lo que permite una mayor interoperabilidad y una experiencia de usuario más fluida.

4.4. Software Empleado

4.4.1. Chat With Your Data

Este proyecto llamó mi atención desde un principio ya que entrega una interfaz elegante y la capacidad de “conversar” con PDFs usando una base de datos vectorial, una práctica habitual usando LLMs conocida como RAG [99] o Retrieval-Augmented Generation, cuyos pasos son:

- **Transformación:** El primer paso es transformar tus datos en embeddings y almacenarlos en una base de datos vectorial.
- **Recuperación:** El segundo paso es recuperar información relevante de la base de datos vectorial. Esto se hace utilizando algoritmos para buscar información que esté semánticamente relacionada con la consulta del usuario.
- **Aumento:** Una vez que se ha recuperado la información relevante, se utiliza para aumentar la consulta original y luego se pasa al LLM.
- **Generación:** Luego, el LLM utiliza esta información para generar una respuesta a la consulta del usuario.

Con este proyecto descubrí lo simple que era crear interfaces con streamlit [16] y bases de datos vectoriales con nucliaDB [76]. Además, con pequeños cambios en el código conseguí usar Llamus en vez de la API de OpenAI obteniendo una interfaz privada similar a ChatPDF⁴, pero sin tener que ceder mis PDFs a una organización privada.

4.4.2. Gephi

Gephi⁵ es una herramienta de código abierto ampliamente utilizada para el análisis y la visualización de grafos, especialmente, redes complejas. Gephi proporciona una plataforma intuitiva y poderosa para importar datos de grafos, realizar análisis estadísticos y visualizar la estructura y dinámica de los datos de manera efectiva.

⁴<https://www.chatpdf.com/>

⁵<https://gephi.org/>

La principal utilidad de Gephi radica en su capacidad para representar visualmente la estructura y las relaciones dentro de un grafo. Algunas de las características clave que hacen que Gephi sea de gran utilidad para la representación de grafos incluyen:

- **Interactividad:** Gephi proporciona una interfaz interactiva que permite a los usuarios explorar y manipular grafos en tiempo real. Esto facilita la identificación de patrones, comunidades y otras propiedades emergentes en los datos del grafo.
- **Personalización:** Los usuarios pueden personalizar la apariencia visual de los vértices y aristas del grafo, así como ajustar el diseño del grafo para mejorar la claridad y la comprensión.
- **Análisis:** Gephi proporciona una variedad de herramientas de análisis, como la distribución de conectividad y la poda del grafo, que permiten a los usuarios calcular métricas clave del grafo, como la centralidad, la modularidad y la densidad. Estas métricas son fundamentales para entender la estructura del grafo y la importancia de sus vértices y relaciones.
- **Exportación:** Los usuarios pueden exportar visualizaciones de grafos en una variedad de formatos de imagen y video, lo que facilita la presentación y el intercambio de resultados de análisis.

4.5. Herramientas de Desarrollo Propio

En esta sección, exploraremos una serie de herramientas de desarrollo propio diseñadas específicamente para abordar los problemas y necesidades de nuestro proyecto.

4.5.1. owlready2 Wrapper

En primer lugar se realizó un pequeño envoltorio de la librería owlready y RDFlib para realizar consultas y modificaciones para nuestro caso de uso de manera eficiente. No entraremos en detalle en el código, bastante sencillo pero requiere un largo proceso de documentación. A continuación se mencionarán los métodos más interesantes:

1. *get_object_properties_dict*: Dado un individuo, devuelve todas las propiedades que tiene asignadas como sujeto así como los individuos/datos que referencian las mismas.
2. *get_subject_properties_dict*: Dado un individuo, devuelve todas las propiedades que tiene asignadas como objeto así como los individuos/datos de las mismas.
3. *get_subgraph_with_properties*: Dada una lista, con URIs de ObjectProperties, devuelve el subgrafo inducido por los triples cuya etiqueta se incluya en la lista dada. Será de gran utilidad cuando lleguemos al capítulo de Grafos.
4. *get_subclasses*: Dada una clase, devuelve todas las clases hijas de manera recursiva usando SparQL ⁶.
5. *reason*: Ejecuta el razonador, encargandose de capturar el error y de procesarlo de manera eficiente. También se puede configurar el ejecutable de Java de forma opcional.

La función principal de la librería es encargarse de mantener la coherencia entre el objeto Graph RDFlib y la ontología de owl2ready, añadiendo a los dos cualquier modificación para que las consultas de ambas clases devuelvan resultados consistentes. Posteriormente se usa RDFlib para guardar la ontología resultante ya que son idénticas. A continuación lo ilustramos con un ejemplo del método *add_class*.

```

1  def add_class(self, class_uri, superclass_uri=None):
2      """
3      Add a new class to the ontology.
4      Args:
5      - class_uri (URIRef): URI of the new class.
6      - superclass_uri (URIRef): URI of the superclass.
7      (Optional)
8      """
9      # owl2ready
10     if superclass_uri:
11         local_URI = superclass_uri.split("#")[-1]
12         parentClass = self.onto[local_URI]
13         if not parentClass:
14             parentClass = ovr.Thing
15     else:
16         parentClass = ovr.Thing
17     local_class_uri = class_uri.split("#")[-1]
18     ovr.types.new_class(local_class_uri, parentClass)

```

⁶<https://www.w3.org/TR/2008/REC-rdf-sparql-query-20080115/>

```
19 # RDFlib
20 self.rdf_graph.add((class_uri, RDF.type, OWL.Class))
21 if superclass_uri:
22     self.rdf_graph.add((class_uri,
23                         RDFS.subClassOf,
24                         superclass_uri))
```

Listing 4.4: Ejempllo método RDFUtils

4.5.2. LLM Wrapper for Llamus in LangChain

En esta sección explicaremos las clases que se han desarrollado [58] para interactuar con la interfaz que brinda LangChain [53].

- Llamus (subclase de LLM)
- LlamusChat (subclase de BaseChat)

. Nos basaremos en la documentación CustomLLM y CustomChat de langchain: [51, 52, 53],

Llamus

Para esta clase hay solo dos métodos obligatorios: `_call` y `_llm_type`. El segundo solo tiene efecto a nivel de logs y devuelve la string que identifica el LLM, en este caso devolvemos "Llamus". A continuación comentaremos las funcionalidades de nuestra clase Llamus en el método `_call`:

```
1 # Método para realizar la llamada al modelo de lenguaje
  subyacente
2 def _call(
3     self,
4     prompt: str,
5     messages: List[BaseMessage]=[],
6     temperature: float = 0.1,
7     stop: Optional[List[str]] = None,
8     timeout=15,
```

```
9         run_manager: Optional[CallbackManagerForLLMRun]
= None,
10         **kwargs: Any,
11     ) -> str:
```

Listing 4.5: Cabecera método _call

```
1     if stop is not None:
2         stop = None
3         #stop kwargs are not permitted.
4     if temperature < 0 or temperature > 1:
5         raise ValueError("Temperature should be between
0 and 1")
6     headers = {
7         'Content-Type': 'application/json',
8         'Authorization': f"Bearer {self.api_key}"
9     }
10    if prompt != "":
11        messages.insert(0, {
12            "role": "system",
13            "content": prompt
14        })
15    message_unique_set = set(map(lambda x:
tuple(x.items()), messages))
16    messages_unique = []
17    for dict_items in message_unique_set:
18        temp = {}
19        for (k,v) in dict_items:
20            temp[k] = v
21        messages_unique.append(temp)
22    data = {
23        'model': self.model,
24        'messages': messages_unique,
25
26        'temperature': temperature,
27        'stream': False,
28        'max_tokens': None,
29        'top_k': None,
30        'top_p': None
31    }
32    retries = 0
33    max_retries = self.n
```

```
35     while retries < max_retries:
36         try:
37             response =
requests.post(f"{self.endpoint}/v1/chat/completions",
headers=headers, data=json.dumps(data), stream=False,
38                 timeout=timeout)
39
40
41             #curl_text = get_curl(response)
42             total_response = response.json()
43             return
total_response["choices"][0]["message"]["content"]
44         except requests.exceptions.RequestException as e:
45             print(f"Request failed: {e}")
46             retries += 1
47             wait_time = 2 ** retries
48             print(f"Retrying in {wait_time} seconds...")
49             time.sleep(wait_time)
50         raise Exception("Max retries exceeded, request
failed.")
51
52     @property
53     def _identifying_params(self) -> Mapping[str, Any]:
54         """Obtener los parámetros de identificación."""
55         return {"n": self.n, "api_key": self.api_key,
"endpoint": self.endpoint,
56             "model": self.model}
```

Listing 4.6: Cuerpo método _call

Funcionalidades:

1. Almacena los parámetros de configuración: número de reintentos, jwt token, el endpoint base de la API Ollama [78] y el modelo a utilizar.
2. Reintentos con retardo exponencial: En caso de error la petición se repetirá a los 2 segundos, después a los 4 segundos, hasta los 2ⁿ segundos.
3. Formato API: Se encarga de adaptar la petición a la API de Ollama desde el formato de langchain (prompt y messages)
4. Timeout: Se encarga de cortar la petición para que esta no sobrecarge el servidor

Se implementa también una función para listar los modelos disponibles:

```
1 # Método para obtener una lista de modelos disponibles
2 def list_models(self):
3     headers = {
4         'Content-Type': 'application/json',
5         'Authorization': f"Bearer {self.api_key}"
6     }
7     response = requests.get(f"{self.endpoint}/api/tags",
8 headers=headers)
9     #print(response.json())
10    return [model["model"] for model in
11 response.json()["models"]]
```

Es importante aumentar los buffers de envío y recepción del socket si se pretenden hacer peticiones con muchos tokens:

```
1 import urllib3, socket
2 from urllib3.connection import HTTPConnection
3
4 HTTPConnection.default_socket_options = (
5     HTTPConnection.default_socket_options + [
6     (socket.SOL_SOCKET, socket.SO_SNDBUF, 1000000),
7     #1MB in byte
8     (socket.SOL_SOCKET, socket.SO_RCVBUF, 1000000)
9 ])
```

4.6. Trabajo Futuro

En futuros trabajos, se podrían implementar varias mejoras:

1. **Métodos asíncronos:** Transferir cada token conforme se infiere, en lugar de esperar la respuesta completa, para mejorar la eficiencia y la experiencia del usuario.
2. **Conteo y control de tokens:** Implementar técnicas avanzadas para gestionar el número de tokens generados, optimizando recursos y cumplimiento de restricciones específicas.

3. **Parámetros *top_k* y *top_p*:** Permitir el ajuste de estos parámetros para mejorar la calidad y coherencia de las respuestas generadas por los modelos de lenguaje.

Estas mejoras pueden significar avances importantes en la eficiencia y efectividad del uso de la plataforma API de Llamus [78]⁷.

⁷<https://llamus.cs.us.es/>

Capítulo 5

Grafos

En este capítulo describimos los conceptos y herramientas relativas a grafos que serán necesarias para el desarrollo del proyecto y como se relacionan con las ontologías. Hemos recabado la información de [41, 86]

5.1. Grafos Dirigidos etiquetados

Un grafo dirigido etiquetado, también conocido como digrafo etiquetado, es una estructura matemática que consiste en:

- Un conjunto de vértices: $V = \{v_1, v_2, \dots, v_n : v_i \text{ es un vértice, } i = 1, 2, \dots, n\}$ (en el caso de la ontología clases e individuos)
- Un conjunto de aristas dirigidas: $E = \{e_1, e_2, \dots, e_m : e_j = (u, v) \in V \times V \mid j = 1, 2, \dots, m\}$. En el caso de la ontología, servirá para representar triples mediante la función de etiquetado.
- Una función de etiquetado que asigna una etiqueta a cada arista: $f : E \rightarrow \text{Etiquetas}$ donde *Etiquetas* es el conjunto que contiene todas las posibles etiquetas. En el caso de la ontología. $\text{Etiquetas} = \text{ObjectProperties} \cup \text{DataProperties}$

5.2. Algoritmos de Búsqueda

En esta sección, describiremos los algoritmos de búsqueda en grafos que utilizaremos en el proyecto. Utilizaremos versiones modificadas de estos, adaptadas para realizar búsquedas en el grafo de conocimiento resultante de la extracción de información (Capítulo 6).

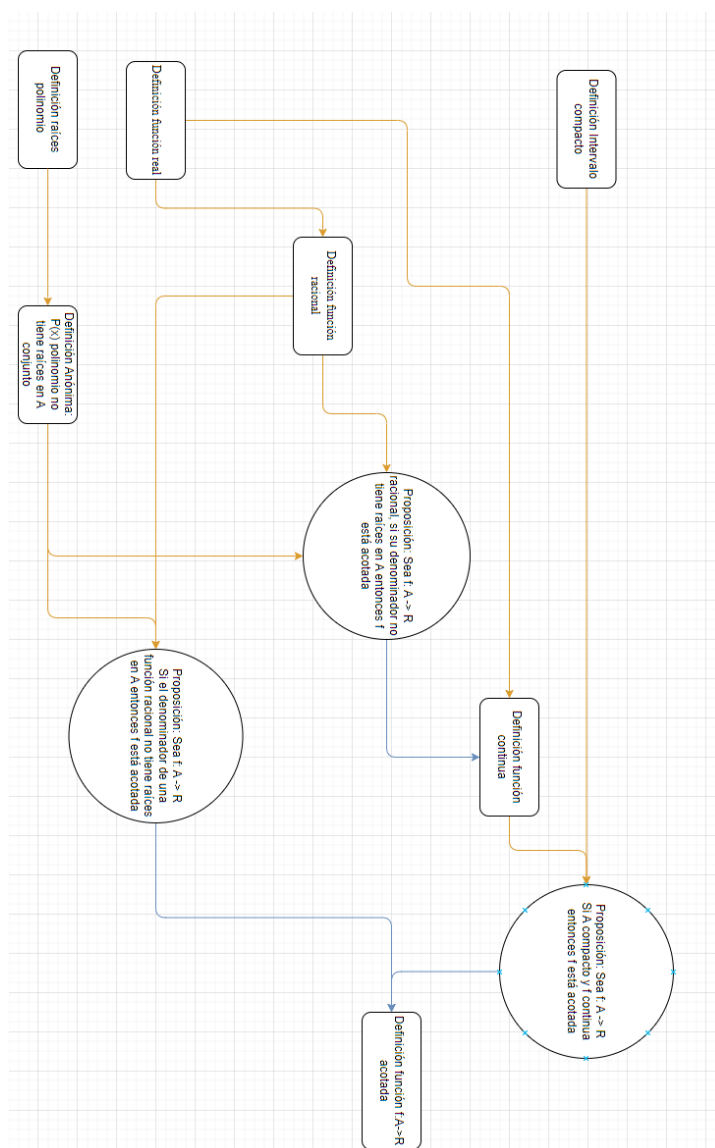


Figura 5.1: Ejemplo de grafo de conocimiento

Aunque los algoritmos originales se estudiaron en el primer curso del grado, considero útil repasar sus pasos antes de proceder con las modificaciones. Estos algoritmos serán útiles para las siguientes tareas:

- **Optimización de hipótesis:** Búsqueda en profundidad (DFS)
- **Maximización de conclusiones:** Búsqueda en amplitud (BFS)
- **Búsqueda de pruebas:** Algoritmo de Dijkstra

Las aplicaciones de estos algoritmos se discutirán en el capítulo 7.

Para ilustrar el comportamiento de nuestros algoritmos usaremos el grafo de la Figura 5.1 donde las aristas naranjas representan la propiedad *isAssumed* y las azules *impliesThat*.

5.2.1. Algoritmo de Dijkstra

El algoritmo de Dijkstra es una solución al problema del camino más corto en un grafo dirigido ponderado, donde las aristas tienen como etiquetas pesos no negativos.

El algoritmo se basa en la idea de mantener un conjunto de nodos cuyas distancias mínimas desde el nodo origen ya son conocidas y otro conjunto con nodos cuya distancia mínima aún no se ha determinado. El proceso se describe a continuación:

1. Inicializar todas las distancias con un valor infinito excepto para el nodo origen, que se inicializa con 0.
2. Mientras queden nodos por visitar:
 - a) Seleccionar el nodo con la distancia más corta no visitado, denominado nodo actual.
 - b) Actualizar la distancia de todos los vecinos del nodo actual.
 - c) Si la distancia calculada para un nodo es menor que la distancia conocida, actualizar la distancia.
 - d) Marcar el nodo actual como visitado.
3. Repetir el proceso hasta que todos los nodos hayan sido visitados o la distancia mínima conocida sea infinita, lo que indica que no hay conexión con el nodo origen.

Seguidamente, describimos **nuestra modificación**. En primer lugar, enumeramos el conocimiento previo que aprovecharemos al trabajar con la ontología diseñada en el Capítulo 3:

- Las conclusiones son individuos de la clase *Definition*: si fueran de la clase *Axiom*, no tendríamos nada que buscar, ya que estos siempre son ciertos. Las demás clases, *Theorem*, *Proposition*, *Corollary* o *Lemma*, no se implican mediante la propiedad *impliesThat*, sino que se demuestran mediante *isProved*.
- Las hipótesis solo pueden ser de la clase *Definition*: los individuos de la clase *Axiom* siempre se asumen como ciertos y no tendría sentido asumir un resultado de otro tipo que ya ha sido previamente probado.
- Podemos asumir que hay una sola conclusión: si no, dadas c_1, c_2, \dots, c_n conclusiones, podríamos añadir dos resultados auxiliares:
 - Una definición anónima: $c_1 \wedge c_2 \wedge \dots \wedge c_n$.
 - Un lema auxiliar: que asuma c_1, c_2, \dots, c_n y sea el único resultado que implique la definición auxiliar (obligando a validar todas las conclusiones iniciales para llegar a la conclusión final).

A continuación, describiremos la **versión modificada** del algoritmo de Dijkstra: dadas unas hipótesis $H = \{h_1, h_2, \dots, h_n\} \subset \textit{Definition}$ y una conclusión $c_0 \in \textit{Definition}$, y con $i = 1$ por defecto:

1. Inicializar todas las distancias entre definiciones con un valor infinito, excepto para el nodo origen, auxiliar y las hipótesis asumidas, que se inicializan con 0 y con el nodo origen como predecesor.
2. Mientras queden nodos por visitar:
 - a) Seleccionar los nodos de resultados no visitados con todas sus hipótesis asumidas en H . Después, obtener todos los resultados implicados no visitados previamente por alguno de estos resultados.
 - b) Para cada conclusión c , sean R los nodos que la implican, asignamos $\textit{predecesor}(c) = \textit{argmin}_{r \in R} (\textit{máx}_{h \text{ asumidas por } r} (\textit{distancia}(h_i)))$ y $\textit{distancia}(c) = \textit{máx}_{h \text{ asumidas por } \textit{predecesor}(c)} (\textit{distancia}(h_i))$.
 - c) Añadir todas las conclusiones a H .
 - d) Marcar los resultados utilizados como visitados.
3. Repetir el proceso hasta que todos los resultados hayan sido visitados o no encontremos nuevos resultados que asuman solo hipótesis en H .
4. Devolver el camino formado por los predecesores de c_0 si la distancia es menor que ∞ . Si no, devolver una lista vacía.

Nótese que una vez que encontramos un camino hasta c_0 , es decir, si $\text{distancia}(c_0) < \infty$, podemos devolver el camino encontrado directamente, asegurando así su existencia. Este camino entre los nodos de H y c_0 tendrá una estructura jerárquica por su construcción. Por otro lado, para hacer uso de consultas avanzadas como la del punto 2a) del algoritmo descrito, que son complejas en un grafo convencional, nos valemos de SPARQL¹. Se ilustra con la función del Listing 5.2.1:

```
1 def get_feasible_statements(hypothesis: List[str], visited:
2   set):
3     """
4     Obtiene las definiciones factibles basadas en las
5     hipótesis y las visitadas.
6     """
7     # Consulta SPARQL para obtener las declaraciones
8     factibles
9     query = f"""
10    PREFIX rdf:
11    <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
12    PREFIX tfgALS: <http://www.semanticweb.org/
13    89usuario/ontologies/2024/1/tfgALS#>
14
15    SELECT DISTINCT ?individual ?conclusion
16    WHERE {{
17      ?individual rdf:type ?type .
18      ?individual tfgALS:assumesThat ?hypothesis .
19      FILTER (?type IN (tfgALS:Theorem,
20      tfgALS:Corollary, tfgALS:Lemma)) .
21      ?individual tfgALS:impliesThat ?conclusion .
22      FILTER (?hypothesis IN ({"", ".join(map(lambda x:
23      "tfgALS:"+x.split("#")[-1], hypothesis))})).
24      FILTER NOT EXISTS {{
25        ?individual tfgALS:assumesThat ?h2.
26        FILTER (?h2 NOT IN ({"", ".join(map(lambda x:
27      "tfgALS:"+x.split("#")[-1], hypothesis))})).
28      }}
29    """
```

¹<https://www.sparql.org/>

```

21 FILTER (?conclusion NOT IN ({", ".join(map(lambda x:
    "tfgALS:"+x.split("#")[-1], hypothesis)})))
22     FILTER (?individual NOT IN ({",
    ".join(map(lambda x: "tfgALS:"+x.split("#")[-1],
    visited)})))
23     }}
24     """
25     # Ejecutar la consulta SPARQL
26     statements_uris =
ontologyUtils.execute_sparql_query(query)
27     return [(str(s["individual"]), str(s["conclusion"]))
    for s in statements_uris.bindings]

```

Listing 5.1: Ejemplo consultas SPARQL

Ejemplo: Consideremos el grafo de la Figura 5.1. Si buscamos un camino desde $H = \{“f : A \rightarrow \mathbb{R}$ es racional”} hasta $c_0 = “f : A \rightarrow \mathbb{R}$ es continua”, el algoritmo nos indicaría que tal camino no existe. Es decir, H no implica c_0 . Por lo tanto, podemos concluir que existen funciones racionales que no son continuas o intentar demostrar un nuevo resultado que respalde la afirmación contraria.

Sin embargo, si partimos de $H = \{“f : A \rightarrow \mathbb{R}$ es racional”, “El denominador de f no tiene raíces en A ”, “ A es compacto”}, encontraremos que el camino más corto es utilizar la proposición que aparece en la parte inferior de la figura en lugar de las otras dos, ya que solo se requiere una implicación.

5.2.2. Algoritmo de Búsqueda en Profundidad

El algoritmo de Búsqueda en Profundidad (Depth First Search, DFS) es un método para recorrer o buscar en un grafo dirigido. Comienza en un nodo inicial (llamado raíz ya que se obtiene un árbol) y explora lo más profundamente posible a lo largo de cada rama antes de retroceder. El DFS utiliza una pila para realizar el recorrido, lo que lo hace adecuado para implementaciones recursivas y no recursivas.

El algoritmo DFS se puede describir de la siguiente manera:

1. Comienza en un nodo inicial (la raíz).
2. Marca el nodo actual como visitado.

3. Para cada nodo adyacente al nodo actual que no ha sido visitado, realiza una llamada recursiva a DFS desde ese nodo.
4. Repite este proceso hasta que no queden nodos sin visitar.

El DFS es útil para varias tareas, como encontrar ciclos en un grafo, determinar la conectividad del grafo, encontrar componentes conexas y más.

En nuestra **versión modificada** del algoritmo DFS, dada una lista de hipótesis $H = \{h_1, h_2, \dots, h_n\} \subset Definition$, una conclusión $c \in Definition$ y una lista de resultados visitados R_V :

1. Comenzaremos con la conclusión que queremos demostrar.
2. Eliminaremos hipótesis redundantes, es decir:
 - Creamos $H_n = \{\}$, $previousH_n = -1$
 - Mientras haya elementos en H ,
 - Si $|H_n| \neq previousH_n$, actualizamos $previousH_n = |H_n|$ y, para cada $h \in H$, si \nexists un camino entre $H \setminus \{h\}$ (utilizando la versión modificada de Dijkstra) y h , entonces añadimos h a H_n y eliminamos h de H .
 - Si $|H_n| = previousH_n$, todas las hipótesis son dependientes. Eliminamos una hipótesis al azar de H y la añadimos a H_n
 - Asignamos $H = H_n$
3. Si $c \in H$, devolvemos un conjunto vacío: $\{\}$.
4. Comprobaremos si existen resultados no visitados que impliquen nuestra conclusión y los guardamos en la lista R_I .
5. Si la lista está vacía, es decir, si ningún resultado implica nuestra conclusión, la única forma de llegar a ella es asumiéndola. Devolvemos $\{c\}$.
6. Si no, para cada elemento $r_i \in R_I$, llamamos de forma recursiva a nuestra función intentando obtener las hipótesis de r_i como conclusiones y manteniendo las hipótesis H . Guardamos el valor asociado a r_i en una lista $C_i, i = 1, 2, \dots, |R_I|$, y marcamos los nodos de R_I como visitados.
7. Devolvemos $argmin_{C_i, i=1, 2, \dots, |R_I|} (|C_i|)$, es decir, la lista de hipótesis mínimas.

Ejemplo: Consideremos el grafo de la Figura 5.1. Si usamos como entrada $H = \{“f : A \rightarrow \mathbb{R} \text{ es racional}”\}$ y $c_0 = “f : A \rightarrow \mathbb{R} \text{ es continua}”$, el algoritmo nos indicaría que podemos añadir “El denominador de f no tiene raíces en A ” para concluir c_0

5.2.3. Algoritmo de Búsqueda en Anchura

El algoritmo de Búsqueda en Anchura (Breadth First Search, BFS) es otro método fundamental para recorrer o buscar en un grafo. A diferencia del DFS, que explora tanto en profundidad como sea posible antes de retroceder, el BFS comienza explorando los vecinos más cercanos al nodo visitado antes de moverse a nodos más lejanos.

El BFS utiliza una estructura de datos cola para realizar el recorrido, lo que garantiza que los nodos se visiten en el orden en que se descubren, siguiendo el principio de "primero en entrar, primero en salir"(FIFO).

El algoritmo BFS se puede describir de la siguiente manera:

1. Comienza en un nodo inicial (la raíz) y lo encola.
2. Mientras la cola no esté vacía, saca un nodo de la cola.
3. Marca el nodo actual como visitado.
4. Para cada nodo adyacente al nodo actual que no ha sido visitado, lo encola y lo marca como visitado.
5. Repite los pasos 2-4 hasta que no queden nodos sin visitar.

El BFS es útil para diversas aplicaciones, como encontrar el camino más corto entre dos nodos en un grafo no ponderado, determinar la distancia entre nodos y detectar ciclos de longitud mínima. Además, puede ser utilizado para verificar la bipartición de un grafo y realizar búsquedas en redes sociales o en la web.

En nuestra versión modificada del algoritmo BFS, dada una lista de hipótesis $H = \{h_1, h_2, \dots, h_n\} \subset \text{Definition}$ y una lista de resultados visitados R_V :

1. Comprobaremos si existen resultados no visitados cuyas hipótesis estén todas contenidas en H y los guardamos en la lista R_C .

2. Si la lista está vacía, es decir, si no cumplimos todas las hipótesis de ningún resultado. Las únicas conclusiones a las que podemos llegar son las propias hipótesis H . Devolvemos H
3. Si no, para cada elemento $r_i \in R_C$. Guardamos las conclusiones de r_i en una lista $C_i, i = 1, 2, \dots, |R_C|$, y marcamos los nodos de R_C como visitados.
4. Devolvemos una llamada recursiva que toma como hipótesis $H \cup (\bigcup_{i=1}^{|R_C|} C_i)$ y tiene en cuenta los resultados ya visitados.

En nuestro ejemplo, si asumimos $H = \{“f : A \rightarrow \mathbb{R}$ es racional”, “El denominador de f no tiene raíces en $A”\}$ concluiríamos que la función está acotada y que es continua. Además, obviamente f será racional y su denominador no tendrá raíces en A pues, $A \implies A$, es siempre cierto.

5.3. Algoritmos de Representación

La visualización de datos es esencial para la comprensión y comunicación efectiva de información compleja. Los algoritmos de visualización desempeñan un papel fundamental al transformar datos abstractos en representaciones gráficas comprensibles. Entre las diversas técnicas, la representación de grafos destaca por su capacidad para modelar relaciones y estructuras complejas de datos, como las presentes en nuestra ontología.

En el ámbito de la Ingeniería Informática y las Matemáticas, la geometría computacional, que abarca la representación de grafos, es una disciplina fundamental. Aunque en algunos planes de estudio esta área pueda no recibir la atención necesaria, su importancia en aplicaciones prácticas es innegable, especialmente al resumir las características de sistemas complejos, como circuitos o redes, en una sola imagen.

En nuestro caso particular, esta representación es esencial para comprender con facilidad cómo se relacionan los enunciados matemáticos.

5.3.1. Algoritmo de Spring

El algoritmo de Spring, también conocido como algoritmo de dirigido por fuerzas o modelo de muelles, es una técnica utilizada para dibujar grafos de

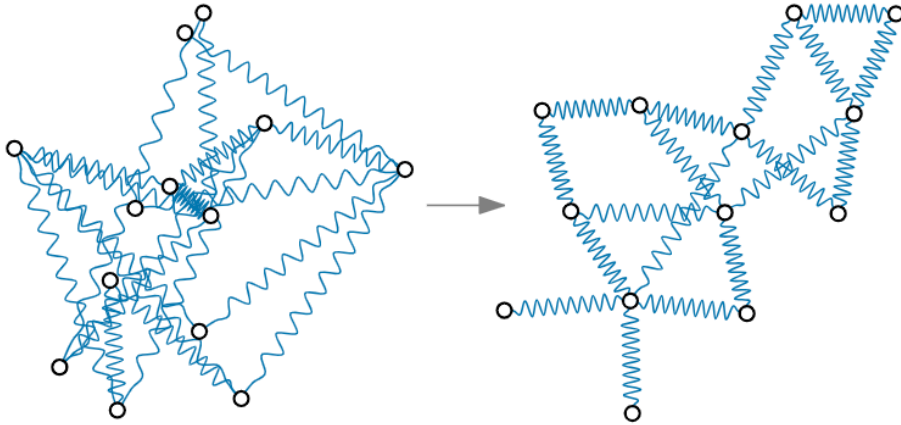


Figura 5.2: Diapositiva extraída de [42]

manera que los nodos se distribuyan de forma natural y estética. El algoritmo considera cada nodo como un objeto con masa y cada arista como un resorte elástico que conecta dos nodos.

La idea principal del algoritmo es asignar una posición inicial a cada nodo y luego ajustar reiteradamente las posiciones para minimizar la energía total del sistema. La energía se calcula como la suma de las energías individuales de cada resorte, que dependen de la longitud y la dirección de la arista.

El algoritmo de Spring funciona de la siguiente manera:

1. Asignar una posición inicial a cada nodo.
2. Calcular la fuerza que actúa sobre cada nodo como la suma de las fuerzas ejercidas por cada arista incidente.
3. Mover cada nodo en la dirección de la fuerza resultante, pero con una distancia que depende de la masa del nodo y la constante elástica de los resortes.
4. Repetir los pasos 2 y 3 hasta que las posiciones de los nodos converjan o se alcance un límite máximo de iteraciones.

El algoritmo de Spring es una herramienta poderosa para visualizar grafos, ya que puede producir representaciones claras y comprensibles incluso para grafos grandes y complejos. Para más información véase [42, 46].

5.3.2. Algoritmo de Representación de Grafos Planos: Método del Desplazamiento

Este algoritmo es de gran utilidad cuando queremos representar grafos planos de gran tamaño sin cruces en sus aristas. Nos puede ayudar a entender como se relacionan los nodos de una forma clara y concisa.

En primer lugar definiremos algunos conceptos necesarios para la representación de grafos planos:

Definición 5.3.1. Una cara de un grafo plano G es una región conexa del plano delimitada por aristas de G , incluyendo el exterior de G .

Definición 5.3.2. La cara exterior de un grafo plano G es la cara que contiene el infinito y no está delimitada por ningún ciclo.

Definición 5.3.3. Un grafo $G = (V, E)$ se considera un plano triangulado si cumple con las siguientes condiciones:

- G es un grafo plano, es decir, puede ser dibujado en el plano sin que las aristas se crucen.
- Cada cara de G es un triángulo, es decir, está formada por exactamente tres aristas que conectan tres vértices distintos.

Teorema 5.3.1. Todo grafo plano puede triangularse añadiendo un número finito de vértices y aristas.

Definición 5.3.4. Sea $G = (V, E)$ un grafo plano triangulado con $n \geq 3$ vértices. Un orden $\pi = (v_1, v_2, \dots, v_n)$ se llama un orden canónico si se cumplen las siguientes condiciones para cada $k, 3 \leq k \leq n$:

- (C1) Los vértices $\{v_1, \dots, v_k\}$ inducen un grafo biconexo internamente triangulado; llámado G_k .
- (C2) La arista (v_1, v_2) pertenece a la cara exterior de G_k .
- (C3) Si $k < n$, entonces el vértice v_{k+1} se encuentra en la cara exterior de G_k , y todos los vecinos de v_{k+1} en G_k aparecen en el borde de G_k de manera consecutiva.

Proposición 5.3.2. Si $G = (V, E)$ un grafo plano triangulado con $n \geq 3$ vértices entonces $\exists \pi = (v_1, v_2, \dots, v_n)$ orden canónico.

Shift Method – Pseudocode

```

Let  $v_1, \dots, v_n$  be a canonical order of  $G$ 
for  $i = 1$  to 3 do
   $L(v_i) \leftarrow \{v_i\}$ 
 $P(v_1) \leftarrow (0,0); P(v_2) \leftarrow (2,0), P(v_3) \leftarrow (1,1)$ 
for  $i = 4$  to  $n$  do
  Let  $w_1 = v_1, w_2, \dots, w_{t-1}, w_t = v_2$ 
  denote the boundary of  $G_{i-1}$ 
  and let  $w_p, \dots, w_q$  be the neighbours of  $v_i$ 
  for  $\forall v \in \cup_{j=p+1}^{q-1} L(w_j)$  do
     $x(v) \leftarrow x(v) + 1$ 
  for  $\forall v \in \cup_{j=q}^t L(w_j)$  do
     $x(v) \leftarrow x(v) + 2$ 
   $P(v_i) \leftarrow$  intersection of  $+1/-1$  diagonals
    through  $P(w_p)$  and  $P(w_q)$ 
   $L(v_i) \leftarrow \cup_{j=p+1}^{q-1} L(w_j) \cup \{v_i\}$ 

```

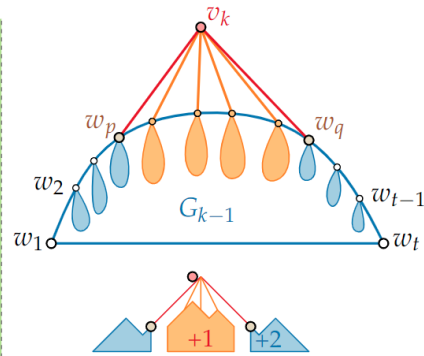


Figura 5.3: Diapositiva extraída de [43]

Por último, se da el pseudocódigo del algoritmo. nótese que al desplazar los vértices internos en 1 unidad y los del lateral derecho en 2 unidades nos aseguramos que las nuevas aristas no introduzcan un nuevo corte en el paso de inducción. El caso base es el triángulo (v_1, v_2, v_3) que se observa en la 4 línea.

5.3.3. Algoritmo de Representación Jerárquica (Método de Sugiyama)

El algoritmo de representación jerárquica, también conocido como método de Sugiyama, es una técnica utilizada para dibujar grafos dirigidos que poseen una estructura jerárquica. El algoritmo se utiliza comúnmente para dibujar diagramas de flujo y organizativos.

El algoritmo funciona de la siguiente manera:

1. Eliminación de ciclos.
2. Asigna un nivel a cada nodo.
3. Se crean nodos intermedios si hay más de un nivel de separación y se procederá a minimizar el número de cruces entre las aristas usando

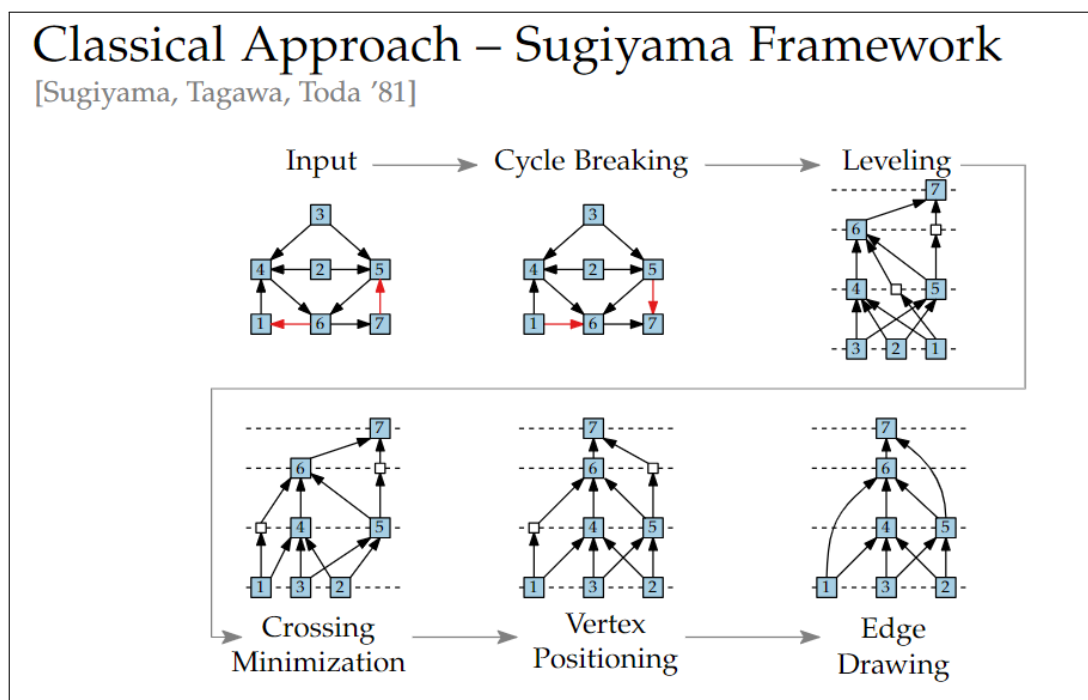


Figura 5.4: Diapositiva extraída de [44]

alguna heurística

4. Se posicionan los vértices y se dibujan las aristas

La figura 5.5 ilustra el método de Sugiyama

Este método no se encontraba implementado en Python 3 por lo que lo hemos implementado (Véase [59]) basándonos en [29] A continuación describiremos en detalle las distintas partes del algoritmo, ya que no se estudió en el grado y ha sido implementado.

Definición 5.3.5. $R_{in}(v) = \{u : (u, v) \in E\}, v \in V$

Definición 5.3.6. $R_{out}(v) = \{u : (v, u) \in E\}, v \in V$

Eliminación de Ciclos:

1. Se identifican los nodos fuentes ($|R_{in}(v)| = 0$) y sumideros ($|R_{out}(v)| = 0$) del grafo y se almacenan en dos listas separadas.
2. Los nodos fuentes se eliminan sucesivamente del grafo y se agregan a la lista de fuentes.

3. Los nodos sumideros se eliminan sucesivamente del grafo y se agregan a la lista de sumideros.
4. Se selecciona el siguiente nodo candidato según el rango de entrada y salida del nodo. El nodo elegido es aquel con el máximo valor de $|R_{out}(v)| - |R_{in}(v)|$, donde v es un nodo en el grafo G .
5. El nodo seleccionado se elimina del grafo y se agrega a la lista de fuentes.
6. Si quedan vértices en el grafo repetimos, sino recorremos los nodos en el orden añadido viendo sus sucesores. Si tiene un sucesor que ya estaba con anterioridad en la lista, se invierte esa arista.

Asignación de niveles:

Se identifican los sumideros del grafo. Estos se asignan a un nuevo nivel horizontal. Se eliminan todos los sumideros del grafo.

El algoritmo procesa el grafo hasta que ya no queden nodos. En cada iteración, encuentra los sumideros, los asigna a un nuevo nivel y los elimina del grafo. Al finalizar, devuelve los niveles en orden inverso, es decir, de abajo hacia arriba, representando la jerarquía vertical de los nodos en el grafo.

Este enfoque garantiza que los nodos en niveles superiores no dependan de los nodos en niveles inferiores.

Minimización de cruces:

1. Para cada par de niveles adyacentes, se calcula una matriz que representa la cantidad de cruces de aristas entre los nodos de los dos niveles y sus permutaciones, es decir, $a_{i,j}$ es el número de cruces al permutar el nodo i y el nodo j . Luego, se ordenan los nodos del nivel superior de manera que se minimicen los cruces con respecto al nivel inferior. Nota: para este paso los nodos del nivel superior se consideran fijos.
2. Se repite el proceso de ordenamiento de cruces para cada par de niveles adyacentes, utilizando el nivel ordenado previamente como base para la siguiente iteración.

Posicionamiento de vértices: Se asignan coordenadas en función de su posición relativa en su nivel y el nivel en el que está. Los nodos artificiales se usa para interpolar las arista que atraviesan más de un nivel. Para más información véase [44, 75].

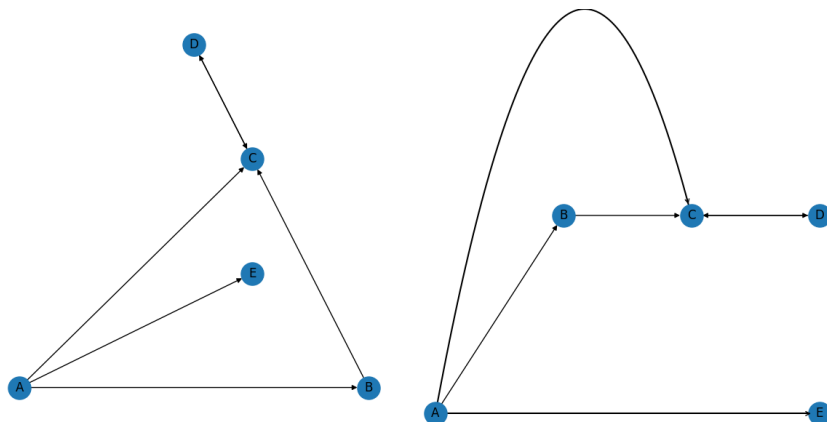


Figura 5.5: Grafo antes y después de aplicar el algoritmo

Capítulo 6

Extracción de Información Matemática de archivos LaTeX

En este capítulo se describirán las técnicas utilizadas para la extracción del conocimiento que poblaran de individuos la ontología. Se ha creado una interfaz web (Véase [60]) inspirada en [107]. Esta nos ayudará a configurar algunos parámetros que aumentan la efectividad de los LLM al incluirlos en el prompt.

6.1. Problemas encontrados

Como trabajo que ha investigado la potencia de diversas alternativas y soluciones computacionales, es importante documentar los resultados de dicho análisis. La tecnología asociada a los LLMs y a la IA generativa en general debe ser examinada críticamente para identificar sus ventajas e inconvenientes. En primer lugar, se mostrarán algunos de los problemas encontrados durante el desarrollo de esta sección. Debido a la novedad de las técnicas utilizadas y la especificidad de nuestra idea, la documentación disponible era, en muchos casos, genérica y enfocada al análisis textual.

6.1.1. Limitaciones LLM

Aunque la tecnología transformer aplicada a los LLMs es prometedora, es necesario tener en cuenta las limitaciones de los LLMs a los que tenemos



Figura 6.1: Comparativa modelos Open Source y Close Source extraída de [84]

acceso (de código abierto y utilizables en un servidor compartido). En este caso, se producían numerosas alucinaciones y fallos repetitivos que impedían al LLM completar la tarea. Para proporcionar un contexto sobre la potencia del LLM empleado en comparación con sus competidores, utilizaremos el ranking MMLU (Fig. 6.1), que se emplea para estimar la "inteligencia" de estos modelos (véase [32, 33]).

Después de un análisis de los modelos disponibles, los dos modelos elegidos son:

1. llama3:8b-instruct (LLM_1) [64]: para procesos simples y precisos. Obtiene una puntuación en el MMLU de 68,4
2. command-r:35b (LLM_2) [3]: para procesos complejos donde se requiere procesar mucha información, pues cuenta con 128k tokens de ventana de contexto y demuestra una gran comprensión de la misma en toda su extensión como se indica en [73]. Obtiene una puntuación en el MMLU de 59,3

Una de las condiciones que nos impusimos en este trabajo fue el uso de software libre. El departamento de CCIA nos proporcionó acceso a LLMs

mediante una palataforma instalada en un servidor dedicado. En la figura 6.1 podemos observar que el rendimiento de estos modelos se aleja considerablemente del de modelos comerciales privativos como GPT-4. Aun así, estamos a la espera del lanzamiento de la versión más potente de LLaMA-3, con 4×10^{11} parámetros, que promete reducir esta brecha de rendimiento. Es muy probable que se logren grandes avances en rendimiento al cambiar a estos modelos.

6.1.2. Procesamiento de archivos PDF

Inicialmente, el objetivo era procesar los apuntes de la carrera de Matemáticas que he estado usando durante mis 5 años de estudios del doble grado. Para su consecución se intentaron diversas estrategias:

1. Uso de APIs existentes: Se han usado algunas webs gratuitas como ¹, para obtener un código LaTeX lleno de etiquetas marcando el estilo que hacen muy difícil para el LLM centrarse en los enunciados.

La deficiencia más importante es que, se pierde la detección de bloques como los de *Theorem*, *Definition*, que proporcionan información muy valiosa a la hora de procesar el enunciado. Aunque no se ha llegado a probar opciones de pago hay algunas como [63] que prometen una solución mejor junto con otra serie de funcionalidades interesantes para el estudio de ficheros PDF de Matemáticas.

2. Conversión a Word y, a continuación, transformarlo a LaTeX mediante un LLM: Se han probado tanto APIs como librerías de Python [22]. Dependiendo del fichero simplemente se obtenía un documento con las fórmulas convertidas a imágenes, o en el que se perdía mucha información. Por ejemplo, "Sea $x_1^2, x_2^2, \dots, x_n^2$ " se suele covertir en "Sea x, x, \dots, x " o "Sea $x12, x22, \dots, xn2$ ".
3. Modelos OCR (Reconocimiento óptico de caracteres): Se han probado opciones como [37], pero tampoco se han obtenido resultados satisfactorios. Sería interesante usar algún modelo de IA para hacer el reconocimiento, como [36], pero su adaptación sería inabordable en el contexto de un TFG.

Ante la dificultad de procesar los PDFs, se ha optado por procesar directamente unos apuntes en LaTeX [35]. Hay que destacar que la cantidad de

¹<https://www.vertopal.com/en/convert/pdf-to-latex>

recursos de Matemáticas en LaTeX es mucho menor que en PDF.

Ya que ambas deficiencias -las limitaciones de los LLMs y la conversión de PDF a latex- están relacionadas con la importancia del código abierto, me gustaría recordar los distintos tipos de licencias para Software Open Source que existen, véase apéndice 8.4

Durante la elaboración de este proyecto, se ha evidenciado que la mayoría de los archivos de matemáticas tienen una licencia de Software Gratuito que no permite ver el código fuente (archivos LaTeX) de los mismos. Es decir, podemos afirmar que la inmensa mayoría de los textos matemáticos básicos publicados en la red como *Open Data* estarían clasificados con dos o tres estrellas en el esquema de cinco estrellas (<https://5stardata.info/es/>). Fundamentalmente, no se benefician del uso de datos estructurados y de URIs.

6.1.3. Formato LaTeX

Otra dificultad detectada durante la elaboración del proyecto es la disparidad en los formatos LaTeX para los elementos de matemáticas, debida también a la ausencia de un estándar. A continuación, definiremos el formato LaTeX “ideal” o aquel que permite la mayor comprensión por parte del LLM. Por ejemplo, consideremos el siguiente resultado:

Teorema 6.1.1 (Teorema de Bolzano). Sea $f : [a, b] \rightarrow \mathbb{R}$

$$\begin{cases} f \text{ is a continuous function} \\ f(a)f(b) < 0 \end{cases} \implies \left\{ \exists c \in [a, b] : f(c) = 0 \right.$$

```

1 \begin{teorema}[Teorema de Bolzano]
2   Sea $f: [a,b] \to \mathbb{R}$\
3   $$
4   \begin{cases}
5     \text{ f is a continuous function } \
6     $f(a)f(b) < 0$
7   \end{cases}
8   $$

```

```
9      $\implies$
10     $$
11     \begin{cases}
12         \exists c \in [a,b] : f(c)=0
13     \end{cases}
14     $$
15 \end{teorema}
```

Listing 6.1: formato LaTeX ideal

se puede observar como las hipótesis y conclusiones estarían claramente diferenciadas. El nombre está encapsulado entre corchetes por lo que es fácil extraerlo mediante expresiones regulares. Además, al principio se definen todos los símbolos que aparecerán en los enunciados y que pueden dar lugar a confusión ya que no es lo mismo que aparezca un f en el enunciado si $f : \mathbb{C} \rightarrow \mathbb{C}$ o aparece $f \in \mathbb{C} : |f| = 1$. Para abordar estas situaciones se modelan los símbolos que aparecen en los enunciados. Esta idea se profundiza en la Secc. 6.2

6.1.4. Idioma

Aunque no es una limitación principal para el rendimiento de modelos superiores, se observa que los modelos usado con un número limitados de parámetros se comportan mucho mejor cuando el texto introducido está en inglés.

6.2. Sistema Empleado

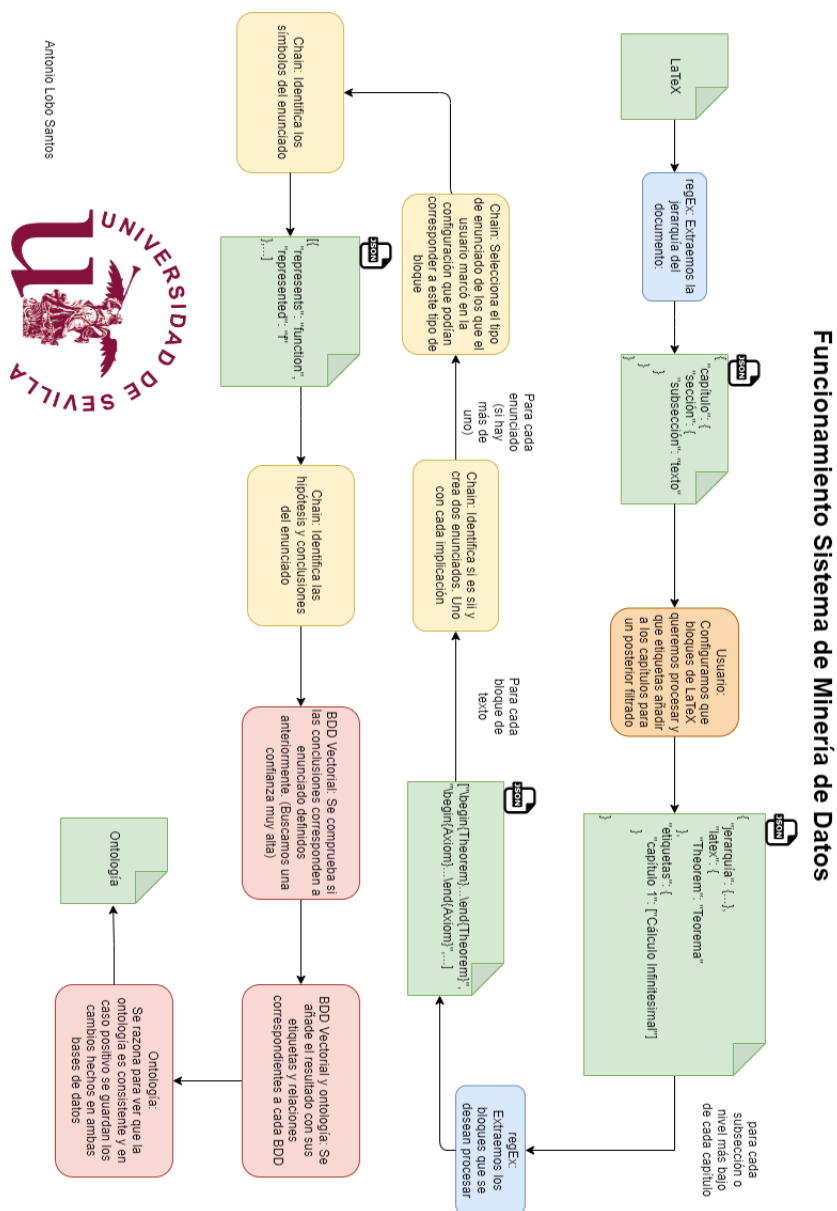


Figura 6.2: Diagrama que muestra el funcionamiento del sistema de minería diseñado

En la Figura 6.2 se presenta un diagrama el proceso de minería de textos, resaltando los puntos más importantes del mismo posteriormente. A continuación se detallará como funciona cada uno de los bloques.

6.3. Plantillas de Prompts

A continuación se mostrará y comentará una de las plantillas de prompt utilizadas. Solo describiremos una plantilla, que servirá para ilustrar las características principales de las mismas:

- **Prompt Engineering:** Se necesita adaptar el prompt al formato esperado por el modelo específico. En este caso, el formato de Command-R sigue la estructura de un archivo Markdown [15].

Es crucial contar con un prompt bien estructurado y adaptado al modelo, como se detalla en [5, 38, 106].

- **Ejemplos Concretos:** Una de las capacidades fundamentales de los LLMs es el aprendizaje sin ejemplos previos, es decir, la capacidad de realizar tareas para las que no ha sido entrenado explícitamente.

Durante el desarrollo del proyecto, se observa una mejora significativa en el rendimiento al incluir ejemplos similares a los que el modelo debe resolver. Cuando un enunciado con una redacción diferente a la habitual no se procesa correctamente, agregar un ejemplo similar puede solucionar no solo el procesamiento de ese enunciado, sino también el de otros enunciados redactados de manera parecida.

- **Especificar Formatos de Entrada y Salida:** Definir claramente los formatos de entrada y salida es fundamental para mejorar la comprensión del modelo. Esta especificación ayuda al modelo a entender cada parte del proceso y cómo debe expresar la información obtenida.

El establecimiento de estos formatos es un área de investigación emocionante en el campo de la Inteligencia Artificial (Véase [31, 55]).

- **Tokens de Inicio y Fin de Turno:** Incluir tokens específicos al inicio y fin de cada turno en los prompts es crucial para facilitar la comprensión del texto por parte del modelo.

Este tipo de tokens ayudan al modelo a distinguir claramente entre las diferentes partes de la secuencia de mensajes y a producir respuestas más cohesivas. Investigaciones recientes han demostrado la importancia del uso de tokens especiales (Véase [57]). En nuestro caso,

```
< BOS_TOKEN >, < |START_OF_TURN_TOKEN| >,  
< |END_OF_TURN_TOKEN| >, < |USER_TOKEN| >, y  
< |SYSTEM_TOKEN| >
```

```
1 statementHypothesisParsing = """
2 <BOS_TOKEN> <|START_OF_TURN_TOKEN|>
3 <|SYSTEM_TOKEN|> # Safety Preamble
4 The instructions in this section override those in the task
   description and style guide sections. Don't answer
   questions that are harmful or immoral. Answer always in
   the format described in style section JSON.
5
6 # System Preamble
7 ## Basic Rules
8 You are a powerful conversational AI trained by some experts
   mathematicians to identify hypothesis.
9 Given a mathematical statement and its most similar
   definitions you should be able to obtains its hypothesis
   in JSON format
10
11 All symbols will have the following structure:
12     - Represented: latex representation of the symbol
13     - Represents: One of the given mathematicalObjects (Ej:
   x\\in R -> Real) (return URI as it is a owl ontology)
14
15 Elements to identify:
16     - Other statements names used in this statement
17     - In case an hypothesis wasn't previously defined you
   will just return its latex representation
18
19 Instructions:
20     1. Identify which of the previousDefinitions are assumed
   to know in the query statement from the list of
   previousStatements
21     2. Return a list with the URI of them or an empty list
   if any of them is required and there are no hypothesis
```

Listing 6.2: Preámbulo System Prompt

```
1 Example 1:
2   symbols:
3     [
4       {{
5         "represented": "f",
6         "represents": "Function"
7       }},
8       {{
9         "represented": "A",
10        "represents": "Set"
11      }},
12      {{
13        "represented": "B",
14        "represents": "Set"
15      }},
16      {{
17        "represented": "x",
18        "represents": "element"
19      }}
20    ]
21    previousStatements: []
22    query: "Diremos que  $f:A \rightarrow B$  es una función si  $f$  es
23           un conjunto de pares ordenados tal que  $(a, b), (a, c) \in A \times B \implies b=c$  Si  $(a, b) \in f$ 
24           entonces denotaremos por  $f(a)$  a  $b$ "
25
26    Hypothesis: []
```

Listing 6.3: Ejemplo sin hipótesis

```

1  Example 2:
2  symbols:
3  [
4  ...
5  ]
6  previousStatements: [
7  {{
8  "text":"Diremos que  $f:A\to B$  es una función si
f es un conjunto de pares ordenados tal que  $(a,
b),(a,c) \in A\times B \implies b=c$  Si  $(a,b) \in
f$  entonces denotaremos por  $f(a)$  a  $b$ ",
9  "name":"Definicion1.1.1.1-nsaduh5287"
10 }}
11 {{
12 "text":"Definimos el límite de una función  $f:A\to
B$  cuando  $x$  tiende a un punto  $a \in A$  como:
 $\lim_{x\to a} f(x) = L \iff \forall \epsilon > 0, \exists \delta > 0, \forall x \in A :
0 < |x-a| < \delta \implies |f(x) - L| <
\epsilon$ "
13 "name":"Definicion1.1.1.2-safgf5423"
14 }}
15 {{
16 "text":"Diremos que una función
 $f:\mathbb{R}\to B\subseteq\mathbb{R}$  es
continua si  $\lim_{x\to a} f(x) = f(a)$ ",
17 "name":"Definicion1.1.1.3-ssaafs33421"
18 }}
19 ]
20 query: "Diremos que una función  $f:\mathbb{R}\to
B\subseteq\mathbb{R}$  es derivable en  $x\in A$  con
derivada  $f'(x)$  si  $\exists \lim_{h\to 0}
\frac{f(x+h)-f(x)}{x+h-x} = f'(x)$ "
21
22 Hypothesis:["Definicion1.1.1.3-ssaafs33421"]

```

Listing 6.4: Ejemplo con hipótesis

```

1 END OF EXAMPLES
2 # User Preamble
3 ## Task and Context
4 Symbols:
5 {symbols}
6
7 Previous Statements:
8 {previousStatements}
9
10 ## Style Guide
11 {format_instructions}
12
13 <|END_OF_TURN_TOKEN|> <|START_OF_TURN_TOKEN|><|USER_TOKEN|>
14 {statement}<|END_OF_TURN_TOKEN|>
15   <|START_OF_TURN_TOKEN|><|SYSTEM_TOKEN|>Write a
16   json-formatted list of hypothesis that you identify in
17   the text. You never return an empty list, you should
18   always assume definitions of concepts involved in the
19   statement. The list of hypothesis you identify must be
20   formatted as a list of strings with hypothesis' latex
21   expressions, for example:
22   Don't return symbols I want statements names from previous
23   Statements
24   Hypothesis:'''json
25   ["Definicion1.1.1.3-ssaafs33421",
26    "Teorema2.3.1.4-adbsjdjj3323" ]<|END_OF_TURN_TOKEN|>'''
27 ]'''

```

Listing 6.5: Preámbulo usuario, consulta y formato

```

1 promptStatementHypothesisParsing = PromptTemplate(
2     template= statementHypothesisParsing,
3     input_variables=["symbols", "statement",
4     "previousStatements"],
5     input_types={
6         "symbols": SymbolListModel,
7         "statement": str,
8         "previousStatements": List[str]
9     },
10    partial_variables={"format_instructions":
11    MathematicalStatementListParser.get_format_instructions()})

```

Listing 6.6: Instanciación clase PromptTemplate

6.4. Modelos Pydantic

Pydantic es una biblioteca de Python que proporciona una forma sencilla y declarativa de definir y validar modelos de datos. Permite a los desarrolladores especificar la estructura de los datos esperados, así como las reglas de validación, facilitando así la interoperabilidad entre diferentes partes de un sistema y garantizando la integridad de los datos.

Esta se integra fácilmente con LangChain, como veremos en la sección 6.5. A continuación mostraremos algunos de los modelos más representativos del proyecto en Listing 6.4.

```
1 class SymbolModel(BaseModel):
2     represented: str = Field(description="Representation in
3     LaTeX")
4     represents: str | int | float = Field(description="Kind
5     of the MathematicalObject that represents")
6
7 class SymbolListModel(BaseModel):
8     __root__: List[SymbolModel]
9
10    def __init__(self, *args, **kwargs):
11        super().__init__(__root__=args, **kwargs)
12
13    def __getitem__(self, key):
14        return self.__root__[key]
15
16    def __iter__(self):
17        return iter(self.__root__)
18
19    def __next__(self):
20        return next(self.__root__)
```

Listing 6.7: Modelos Pydantic

Nótese que *represents* debería ser siempre una cadena de texto, pero por compatibilidad futura con el soporte a operaciones con símbolos de constantes se aceptan también números enteros y reales.

6.5. Cadenas de Langchain

LangChain introduce el concepto de **cadenas**, que son secuencias de llamadas diseñadas para facilitar la construcción de flujos de trabajo de PLN. Estas cadenas pueden incluir llamadas a LLMs, herramientas externas o pasos de preprocesamiento de datos.

La principal forma de construir estas cadenas es mediante **LCEL** (LangChain Composition and Execution Language), que proporciona una metodología estructurada para ensamblar y ejecutar tareas complejas. Las cadenas se clasifican en dos tipos: las construidas con LCEL y las *legacy chains*, que son clases independientes. LangChain también facilita la creación de cadenas que admiten transmisión, asincronía y procesamiento por lotes, así como la observabilidad en cada paso del proceso.

Para procesar los archivos LaTeX se han diseñado distintas cadenas mediante LCEL, pero la estructura de todas ellas es muy similar por lo que explicaremos la extracción de hipótesis, cuya plantilla de prompt ha sido usada de ejemplo en 6.3 **OutputParser**: Se encarga de procesar la salida del LLM y generar las instrucciones de formato para un modelo de *Pydantic*

```
1 from langchain_core.output_parsers import JsonOutputParser
2 MathematicalStatementListParser = JsonOutputParser(
    pydantic_object = MathematicalStatementListModel)
```

Listing 6.8: OutputParser lista de enunciados matemáticos

LLM: Se encarga de hacer la llamada al LLM, con todas las funciones de reintentos, autenticación, etc ... que requiera. Para ello se implementa la clase Llamus 4.5.2

Con todo estas piezas podemos “montar” la cadena de PLN.

```
1 # Tareas de procesamiento sencillas
2 llm = Llamus(
3     n=10,
4     api_key=API_KEY,
5     endpoint=LLAMUS_BASE_PATH,
6     model="llama3:8b-instruct",
7 )
8 # Tareas de procesamiento complejas
9 llm2 = Llamus(
10     n=10,
11     api_key=API_KEY,
12     endpoint=LLAMUS_BASE_PATH,
13     model="command-r:35b",
14 )
15
16 parseStatementHypothesisChain = (
17     promptStatementHypothesisParsing
18     | llm2
19     | MathematicalStatementListParser
20 )
```

Listing 6.9: Instanciación LLM y Chain

6.6. Expresiones Regulares

Las expresiones regulares son una poderosa herramienta utilizada en informática y procesamiento de texto para buscar y manipular patrones de caracteres dentro de cadenas de texto. Son ampliamente utilizadas en la validación de entradas de usuario, análisis de texto y en la extracción de información específica de grandes conjuntos de datos. Destacamos su utilidad en 3 funciones.

Extracción de la estructura de un archivo LaTeX: Gracias a las expresiones regulares puedo extraer la estructura del archivo LaTeX para después proceder con el procesado de cada una de sus partes de manera independiente, estas se usan en el método `getHierarchy` de la clase `LatexProcessor` que devuelve un diccionario con el contenido de cada capítulo, sección y subsección. (Véase [60]).

```
1 def getHierarchy(self):
2     hierarchy = {"outside": {}}
3     chapter_pattern =
4     r'(?ism)\\chapter\*?\s*\{([^\}]*)\}((.|\n)+?)\
5     (\\chapter|\\end\{document\})' # match \\chapter or
6     \\chapter*
7     section_pattern =
8     r'(?ism)\\section\*?\s*\{([^\}]*)\}((.|\n)+?)\
9     (\\chapter|\\section|\\end\{document\})' # match
10    \\section or \\section*
11    subsection_pattern =
12    r'(?ism)\\subsection\*?\s*\{([^\}]*)\}((.|\n)+?)\
13    (\\chapter|\\section|\\subsection|\\end\{document\})' #
14    match \\subsection or \\subsection*
15    ...
```

Listing 6.10: Expresiones Regulares LatexProcessor


```
1 def extract_conclusions_and_hypothesis(symbols,
2   statementText, similarStatements, statementToName):
3     max_retries = 5
4     retry_delay = 0.5 # segundos
5     ...
6
7     for _ in range(max_retries):
8         try:
9             hypothesis =
10            parseStatementHypothesisChain.invoke({
11                "symbols": symbols,
12                "statement": statementText,
13                "previousStatements": json.dumps([{'statement': ss['text'], 'name':
14                statementToName[ss['text']] } for ss in similarStatements
15                if ss['text'] in statementToName and ss['text'] !=
16                statementText])
17            })
18            break
19        except OutputParserException as e:
20            print(e)
21            new_str, changed =
22            substitute_regex(patterns_subs, e.llm_output)
23            if changed:
24                try:
25                    hypothesis = json.loads(new_str)
26                    break
27                except:
28                    print(e)
29                    continue
30            else:
31                print(f"Error al procesar hipótesis. Retry
32                {_+1}/{max_retries}. Error: {e}")
33                time.sleep(retry_delay)
34                continue
```

Listing 6.12: Uso de expresiones regulares en la obtención de hipótesis

Filtrado de símbolos falsos: Nos sirve para descartar símbolos que realmente no lo son, como el \$ de las ecuaciones en LaTeX que confunde a los LLMs. También suele escaparlos muchas veces para detectarlo como símbolo.

```

1 def getGeneralSymbols(statementText, mathematicalObjectsURI,
2   max_retries=4):
3     ...
4     while len(result) == 0 and retries < max_retries:
5         try:
6             result = parseGeneralSymbols.invoke({
7                 "query": statementText,
8                 "mathematicalObjects":
9   json.dumps(classes_uri)
10            })
11        except OutputParserException as e:
12            new_str, changed =
13            substitute_regex(patterns_subs, e.llm_output)
14            if changed:
15                try:
16                    result = json.loads(new_str)
17                except:
18                    print(e)
19                    continue
20            else:
21                print(e)
22                continue
23            valid_symbols = []
24            invalidPatterns = [r"\$", r"^\++$"]
25            import re
26            for symbol in result:
27                if "represents" in symbol and
28                isinstance(symbol["represents"], str) and "represented" in
29                symbol and symbol["represents"].lower() in
30                mathematicalObjectsNames and not
31                any([bool(re.match(patt, symbol["represented"])) for patt
32                in invalidPatterns]):
33                    valid_symbols.append({
34                        "represents":
35                symbol["represents"].lower(),
36                        "represented": symbol["represented"]
37                    })
38            ...

```

Listing 6.13: Filtrado Símbolos Falsos

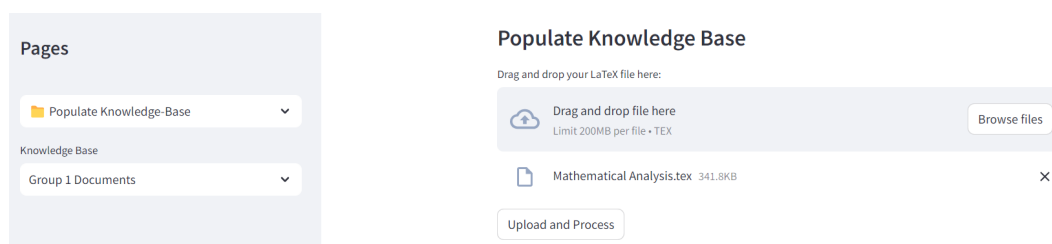


Figura 6.3: Populate Knowledge Base WebUI

Select latex elements to process

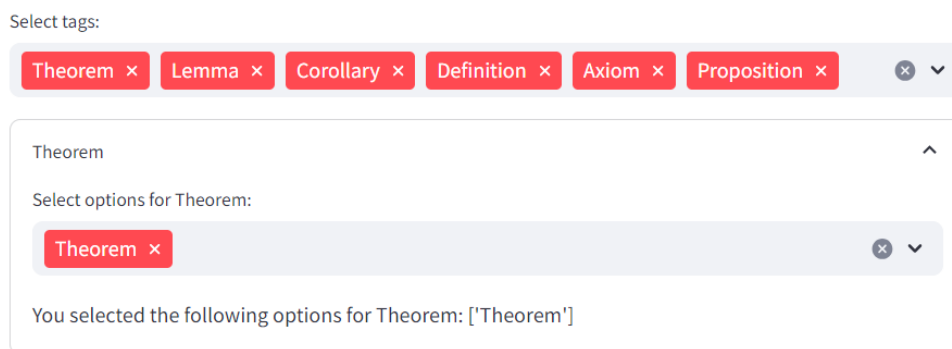


Figura 6.4: Seleccionar elementos de LaTeX a procesar

6.7. Interfaz para la configuración y monitorización

La creación de una interfaz para la configuración y comprobación del proceso es esencial para garantizar su robustez y usabilidad. Esta sección explorará la herramienta desarrollada que permite a los usuarios configurar y verificar el estado de la ingesta de información de manera eficiente y precisa.

El primer paso será seleccionar el archivo LaTeX que deseamos procesar como ilustra la Figura 6.3.

A continuación deberemos elegir cuales de los elementos presentes en el archivo queremos procesar y como que tipos de enunciados pueden ser clasificados. Véase la Figura 6.4

El último paso de configuración será elegir que etiquetas le aplicaremos, primero al documento al general y después a cada capítulo en concreto. Véase la Figura 6.5.

Para terminar, la página irá mostrando la información principal de los

Apply tags by chapter

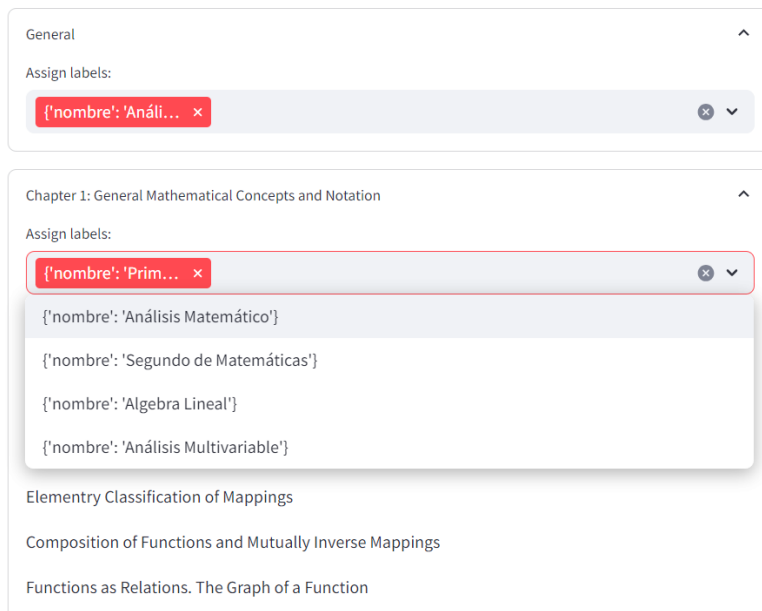


Figura 6.5: Seleccionar etiquetas para el filtrado

enunciados procesados para poder saber si todo está funcionando correctamente. Además, la barra superior indica que porcentaje de capítulo actual llevamos procesado. Véase las Figuras 6.6 y 6.7

6.8. Corrección de fallos manual

A pesar de los avances en los LLM, estos siguen siendo deficientes en el procesamiento de enunciados matemáticos cuando se le exige un nivel de comprensión del problema alto a nivel lógico. Por ejemplo, no siendo capaces de distinguir las hipótesis de la implicación de un enunciado de tipo si y solo si.

Para solventar estos problemas, se han hecho cambios para incluir las etiquetas *Axiom* ya que el autor de los archivos LaTeX no etiquetaba debidamente los Axiomas sino que los mencionaba con una etiqueta de tipo *equation*. También se han reescrito algunos resultados con el fin de mejorar la comprensión del LLM de los mismos.

Processing Logs

Processing Chapter: General Mathematical Concepts and Notation

Processing section: Mathematical Symbols and their meanings

Processing section: Relations and Functions

Processing Definition into one of (Definition)

We say that there is a $\textit{function}$ defined on X with values in Y if, by virtue of some rule f , to each element $x \in X$ there corresponds an element $y \in Y$. $f(X) := \{y \in Y \mid \exists x((x \in X) \wedge (y = f(x)))\}$ X is called the $\textit{domain of definition}$ and Y is called $\textit{set of values}$ or \textit{range} of the function.

Figura 6.6: Logs procesando Archivo 1

Finished parsing, URI: Definition 1.2.1.1-YkkfkZXVaJ2LwYjnKGPgJq

Symbols associated:

	0	1
0	["represented","f"]	["represents","function"]
1	["represented","X"]	["represents","set"]
2	["represented","Y"]	["represents","set"]
3	["represented","x"]	["represents","element"]

Asume: $f(X) := \{y \in Y \mid \exists x((x \in X) \wedge (y = f(x)))\}$

Implica: UndefinedStatement%201.2.1.1.0-AkdKAYNANHwNwPC6Ki8CEe

Figura 6.7: Logs procesando Archivo 2

Capítulo 7

Aplicaciones

7.1. Interfaz WEB

Para el desarrollo de las aplicaciones del proyecto (Fig. 7.1) se elabora la interfaz web Chatex [60].

Comentaremos en esta sección las páginas más básicas y las más avanzadas en su apartado correspondiente.

7.1.1. Etiquetas

Las etiquetas, que pueden ser elegidas libremente, se asignan a los capítulos de un archivo LaTeX o al archivo completo y se utilizan posteriormente para realizar filtrado. Por ejemplo, es posible etiquetar un archivo como “Análisis Matemático”, los primeros tres capítulos como “Primero de Matemáticas” y los capítulos restantes como “Segundo de Matemáticas” (Fig. 7.2). Estas etiquetas se almacenan en una tabla MySQL que contiene dos columnas: *ID* y *nombre*. En esta página podemos añadir, actualizar o eliminar las etiquetas que posteriormente podemos seleccionar para etiquetar nuevos datos o filtrar en las herramientas de consulta.



Figura 7.1: Página principal de Chatex



Figura 7.2: Página etiquetas de Chatex

7.2. Visualización del conocimiento

Se implementa una página (Fig. (7.3, 7.4, 7.5) donde podremos visualizar grandes grafos dirigidos eligiendo:

- El propio archivo que contiene la ontología.
- Aristas a representar en el grafo (Object Properties)
- Etiquetas de los vértices del grafo
- Algoritmo de representación: *Shift* o *Spring 5.3*

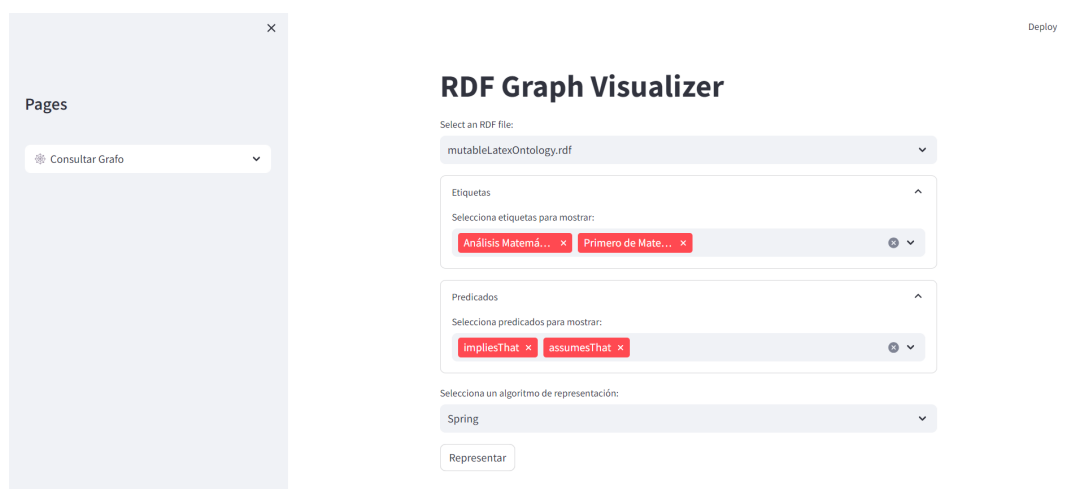


Figura 7.3: Página consultas grafos

Mediante el uso de las etiquetas adecuadas podemos hacer consultas para ver como se visualizan los distintos resultados en función de las asignaturas, áreas de conocimiento o curso entre otros. La exploración visual revela la incapacidad del LLM para identificar resultados relacionados entre sí aunque estos se le pasen como parte del *prompt*. Véase las Figuras 7.4 y 7.5

Como trabajo futuro, sería interesante diferenciar los vértices del grafo mediante las etiquetas con algún código de colores.

7.3. Consultas sobre el conocimiento

En esta sección describiremos algunas de las consultas que los autores consideran más interesantes sobre la información almacenada en la ontología. Se ha diseñado una interfaz donde se pueden elegir hipótesis y conclusiones, para posteriormente aplicar un algoritmo. Véanse las Figuras 7.6, 7.7 y 7.8.

Para ilustrar el funcionamiento de los algoritmos se crean algunos individuos en la ontología diseñada en el Capítulo 3 conforme a la Figura 5.1. A continuación representaremos las relaciones que obtenemos en la pestaña de OntoGraf [26], plugin de Protégé, en la Figura 7.9:

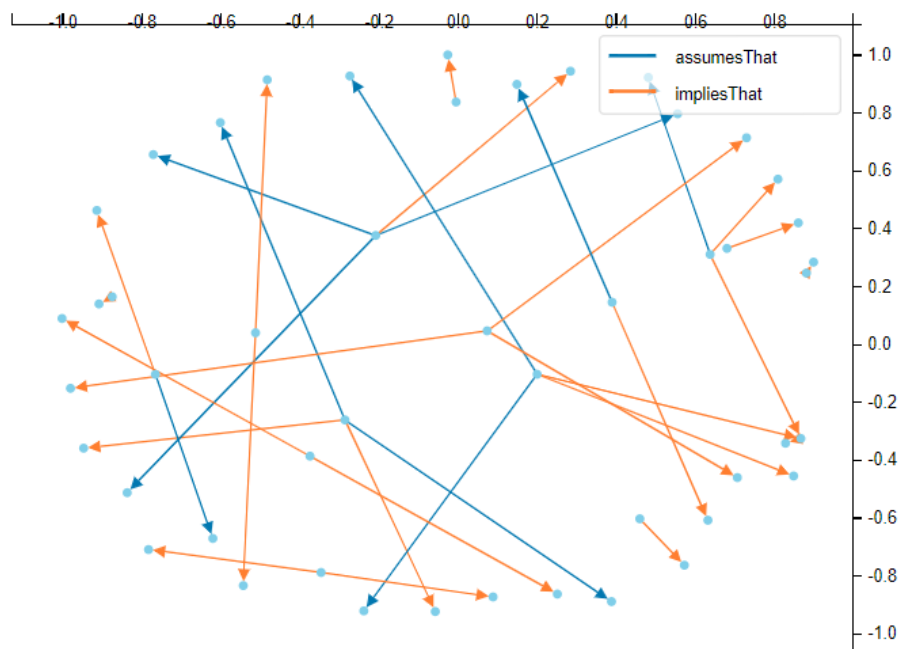


Figura 7.4: Visualización con el algoritmo de representación Spring

7.3.1. Optimización de hipótesis

Supongamos que, junto a la ontología \mathcal{O} , disponemos de una lista de hipótesis $H = \{h_1, h_2, \dots, h_n\}$ y de conclusiones $C = \{c_1, c_2, \dots, c_n\}$, ambos conjuntos contenidos en los individuos de la clase *Definition*; $H, C \subseteq \text{Definition}$. El objetivo es encontrar un conjunto minimal de hipótesis necesarias a añadir para conseguir una cadena de implicaciones que concluya todas las definiciones de C . El algoritmo empleado, similar a un algoritmo de búsqueda en profundidad, fue descrito en detalle en la Subsección 5.2.2.

Para este ejemplo seleccionaremos como conclusión la definición de función acotada y como hipótesis la definición de función racional de la ontología que se observa en la Figura 7.9. Observando el grafo de la Figura 5.1 es fácil ver que tenemos dos proposiciones que implican la conclusión deseada, el algoritmo efectivamente encuentra los URIs de las hipótesis que habría que añadir para usar cada proposición de manera recursiva como se ilustra en el Listing 7.3.1.

Por último, la interfaz web presenta la lista minimal de hipótesis que genera una cadena de implicaciones válida hasta las conclusiones, en nuestro caso la Definición Anónima (La función racional no tiene raíces en su deno-

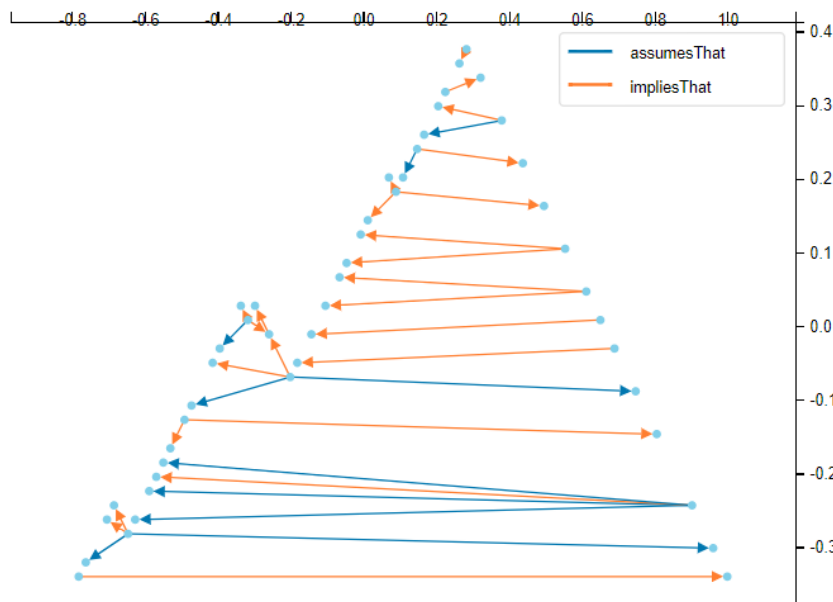


Figura 7.5: Visualización con el algoritmo de representación Shift

minador). Véase la Figura 7.10.

```

1  {
2    "Proposicion1": [
3      "Definicion_Anonima",
4      "Definicion_Intervalo_Compacto"
5    ],
6    "Proposicion3": [
7      "Definicion_Anonima"
8    ]
9  }

```

Listing 7.1: Diccionario con las hipótesis necesarias para cada proposición calculado de manera recursiva en formato JSON

7.3.2. Maximización de Conclusiones

El objetivo dual al anterior consiste en resolver el siguiente problema: dadas unas hipótesis $H = \{h_1, h_2, \dots, h_n\} \subseteq \text{Definition}$, calcular el conjunto máximo de conclusiones que se pueden derivar. En este caso, el algoritmo

Analizador Matemático

Introduce un enunciado matemático:

límite

Número de enunciados similares a mostrar:

5
- +

Confianza mínima:

0.60

0.00
1.00

Buscar

If $A \subset X$ and $f : X \rightarrow Y$ is a function. We denote by $f|_A$ the function $\varphi : A \rightarrow Y$ that agrees with f on A . More precisely, $f|_A(x) := \varphi(x)$ if $x \in A$. The function $f|_A$ is called the restriction of f to A , and the function $f : X \rightarrow Y$ is called an extension or a continuation of φ to X .

Asignar a:

Nada

Asunciones

Implicaciones

Figura 7.6: Búsqueda de resultados similares en la Chatex

utilizado es una adaptación del algoritmo de búsqueda en anchura, descrito en detalle en la Subsección 5.2.3.

En el grafo de ejemplo, se determinará cuántas conclusiones se pueden derivar a partir de las siguientes hipótesis:

- Es una función racional.
- Su denominador no tiene raíces.
- Su dominio es compacto.

Estas hipótesis se ilustran en el siguiente grafo, generado mediante el algoritmo de Sugiyama, explicado en detalle en la Subsección 5.3.3. En el grafo, las aristas naranjas representan la propiedad *isAssumedBy* y las aristas azules la propiedad *impliesThat*. Véanse las Figuras 7.11 y 7.12.

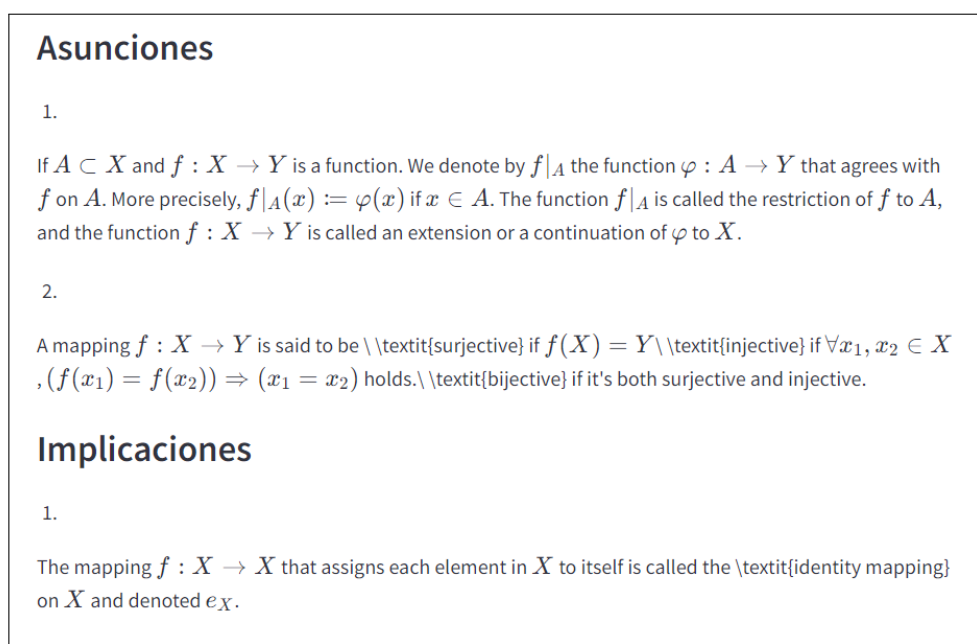


Figura 7.7: Selección hipótesis y conclusiones para algoritmos posteriores

7.3.3. Pseudodemostraciones

Uno de los objetivos principales de los autores ha sido desarrollar un modelo capaz de generar cadenas de implicaciones que “prueben” la validez de una deducción lógica.

Para ello, se ha implementado una modificación del algoritmo de Dijkstra (véase Subsección 5.2.1) que, dadas una lista de hipótesis $H = \{h_1, h_2, \dots, h_n\}$ y una lista de conclusiones $C = \{c_1, c_2, \dots, c_n\}$, ambos conjuntos contenidos en la clase *Definition* ($H, C \subseteq \text{Definition}$), encuentra el “camino” que, partiendo de las hipótesis H , llega con el menor número posible de implicaciones a deducir todas las conclusiones de C .

Este concepto de “camino”, que en realidad es un grafo jerárquico en lugar del concepto clásico de camino de teoría de grafos, es lo que denominaremos *pseudodemostración*. Para ilustrar este proceso, se utilizará nuevamente el algoritmo de Sugiyama para mostrar los distintos pasos (niveles de jerarquía) necesarios para implicar las conclusiones. Véanse las Figuras 7.13 y 7.14.

A continuación, se ilustra el proceso con el camino que llega de manera más rápida, es decir, mediante el menor número de implicaciones, a las conclusiones. Asumimos las siguientes hipótesis:

The mapping $f : X \rightarrow X$ that assigns each element in X to itself is called the `\textit{identity mapping}` on X and denoted e_X .

Asignar a:

Nada

Asunciones

Implicaciones

Minimizar Hipótesis Maximizar Conclusiones Pseudodemostración

Asunciones

1.

If $A \subset X$ and $f : X \rightarrow Y$ is a function. We denote by $f|_A$ the function $\varphi : A \rightarrow Y$ that agrees with f on A . More precisely, $f|_A(x) := \varphi(x)$ if $x \in A$. The function $f|_A$ is called the restriction of f to A , and the function $f : X \rightarrow Y$ is called an extension or a continuation of φ to X .

Figura 7.8: Botones para iniciar algoritmos

- Es una función racional.
- Su denominador no tiene raíces.
- Su dominio es compacto.

Y queremos llegar a la conclusión de que f está acotada.

Si eliminamos la hipótesis de que la función sea racional, la web nos indicará que no existe tal pseudodemostración. Véase la Figura 7.15.

7.3.4. Coherencia académica

En esta subsección, se asumirán como hipótesis todos los resultados de cursos anteriores, los cuales se pueden recuperar mediante filtrado de etiquetas. La entrada del problema será un conjunto de resultados que se quieren impartir en una asignatura A , denotado como C_A .

1. Se eliminarán las hipótesis que se puedan derivar mediante la relación *impliesThat*. Es decir, aquellas hipótesis que tengan una pseudodemostración. Estos resultados se denominarán la "cara externa" de la asignatura, denotados como H_{Ce} .

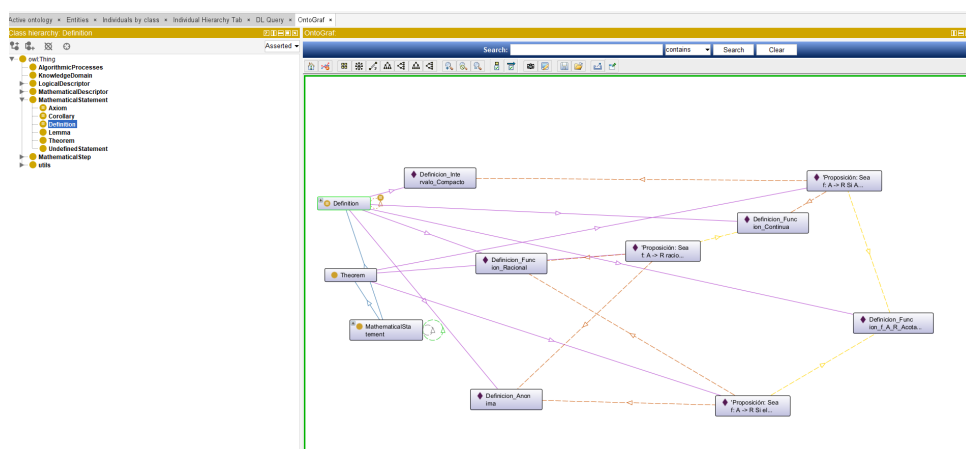


Figura 7.9: Ontología ejemplo OntoGraf

2. Se minimizará el número de hipótesis adicionales necesarias para que todos los resultados de la asignatura puedan ser probados. Estas hipótesis faltantes se denotarán como H_F .
3. Se calculará la pseudodemostración entre los elementos de $H_{Ce} \cap H_F$ y C_A y se representará utilizando el algoritmo de Sugiyama (véase 5.3.3).

Esto permitirá al profesor visualizar fácilmente los resultados que necesita recordar a sus alumnos de cursos anteriores (H_{Ce}), los resultados previos que los alumnos desconocen (H_F) y cómo planificar las clases introductorias de la asignatura. Siguiendo la jerarquía de la pseudodemostración, se podrán cubrir todos estos resultados desconocidos.

Aunque en este ejemplo no se lleva a cabo el filtrado por etiquetas para seleccionar las definiciones debido a la falta de tiempo, con los algoritmos implementados en el proyecto, este objetivo podría lograrse de manera sencilla.

7.4. Agentes Matemáticos

En esta sección se discutirán las diversas fortalezas que aporta contar con la ontología diseñada (Véase Capítulo 3). La ontología desarrollada organiza y estructura el conocimiento matemático de manera eficiente, mejorando la capacidad de los agentes autónomos para realizar tareas complejas y específicas en el ámbito educativo y de la investigación.

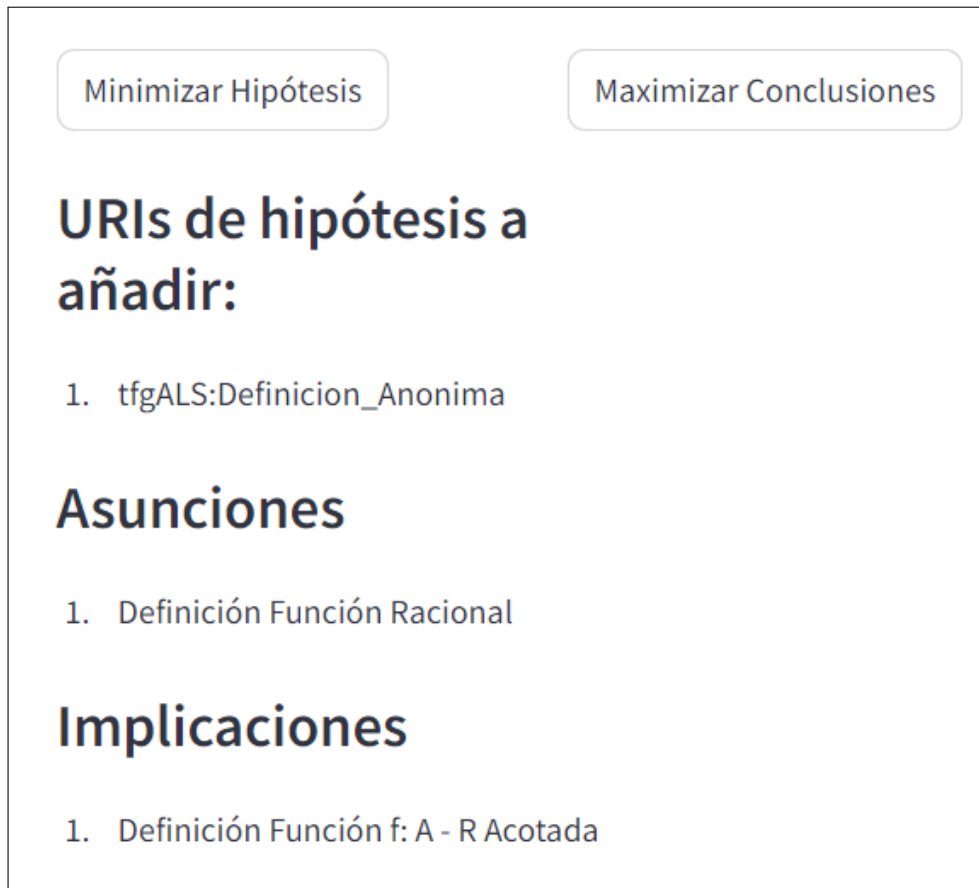


Figura 7.10: Ejemplo minimización de hipótesis en la interfaz web

7.4.1. Deficiencias de Agentes Autónomos que podrían ser suplidas por la ontología diseñada

La ontología propuesta aborda varias deficiencias inherentes a los agentes autónomos convencionales, mejorando su rendimiento en diversas áreas. A continuación, se detallan las mejoras específicas:

- **Capacidad de mantener un hilo de argumentación lógico con un objetivo final lejano:** Los agentes autónomos suelen centrarse en la generación de respuestas inmediatas sin considerar un objetivo a largo plazo. La ontología proporciona una estructura que permite al agente seguir una secuencia lógica de pasos hacia un objetivo específico, mejorando la coherencia y profundidad del razonamiento.
- **Validación de una secuencia lógica de acciones:** La ontología facilita la validación de las secuencias lógicas de acciones, permitiendo al agente

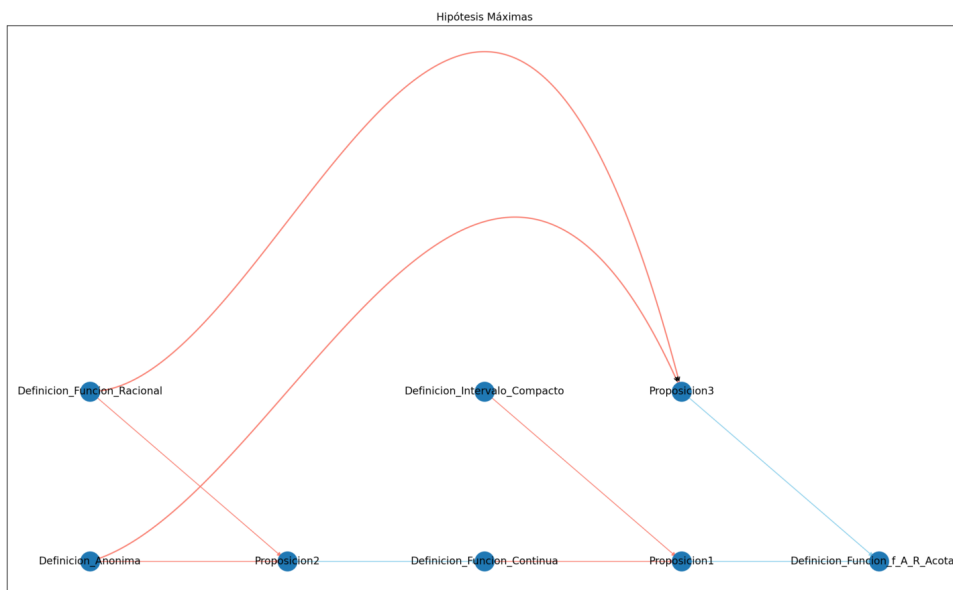


Figura 7.11: Ejemplo de grafo obtenido de la maximización de conclusiones.

prever errores potenciales en el razonamiento a largo plazo. En lugar de generar simplemente el siguiente token o acción más probable, el agente puede evaluar la validez de ciertos procesos de deducción lógica.

- **Procesos Algorítmicos:** La definición de herramientas como individuos de la clase *AlgorithmicProcess* dota al LLM de la capacidad de resolver problemas específicos mediante la implementación de algoritmos conocidos. Por ejemplo, el individuo *algoritmoEuclides*, que representa el algoritmo de Euclides para hallar el máximo común divisor, está asociado con la *Data Property hasCode* que contiene el código en Python para resolver este algoritmo.

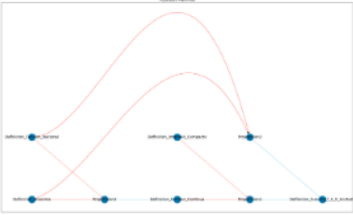
De esta forma, mediante una búsqueda por similitud en las descripciones de los algoritmos, el agente puede identificar y utilizar métodos algorítmicos precisos para resolver problemas complejos. Además, por la naturaleza de la ontología, sería muy sencillo enlazarlo con el resultado matemático que justifica su uso.

- **Mejor organización del conocimiento:** La ontología ayuda a estructurar y categorizar el conocimiento científico de manera que los agentes autónomos puedan acceder a la información de forma más organizada y eficiente.
- **Interoperabilidad entre diferentes dominios de conocimiento:** La ontología permite que los agentes autónomos integren y utilicen conoci-

Minimizar Hipótesis

Maximizar Conclusiones

Pseudodemostración



Asunciones

1. Definición Anónima
2. Definición Función Racional
3. Definición Intervalo Compacto

Figura 7.12: Ejemplo de maximización de conclusiones en la interfaz web.

mientos de diferentes dominios, mejorando su capacidad para abordar problemas multidisciplinares.

- **Facilidad en la actualización del conocimiento:** Con una ontología bien definida, actualizar y expandir el conocimiento que utiliza el agente autónomo es más sencillo y sistemático, lo cual es crucial para mantener la relevancia y precisión de la información.
- **Eficiencia en la detección de errores conceptuales:** La estructura ontológica facilita la identificación de errores conceptuales en el razonamiento del agente, mejorando la precisión y fiabilidad de sus respuestas.
- **Personalización del aprendizaje:** La ontología permite adaptar las respuestas y soluciones del agente a las necesidades específicas de los usuarios, ofreciendo una experiencia de aprendizaje más personalizada y efectiva.

Asimismo, se pueden restringir ciertas áreas de conocimiento para que el agente se ciña al conocimiento actual del usuario. En nuestro caso de uso, no usando enunciados de cursos superiores al del alumno que interactúa con el agente.

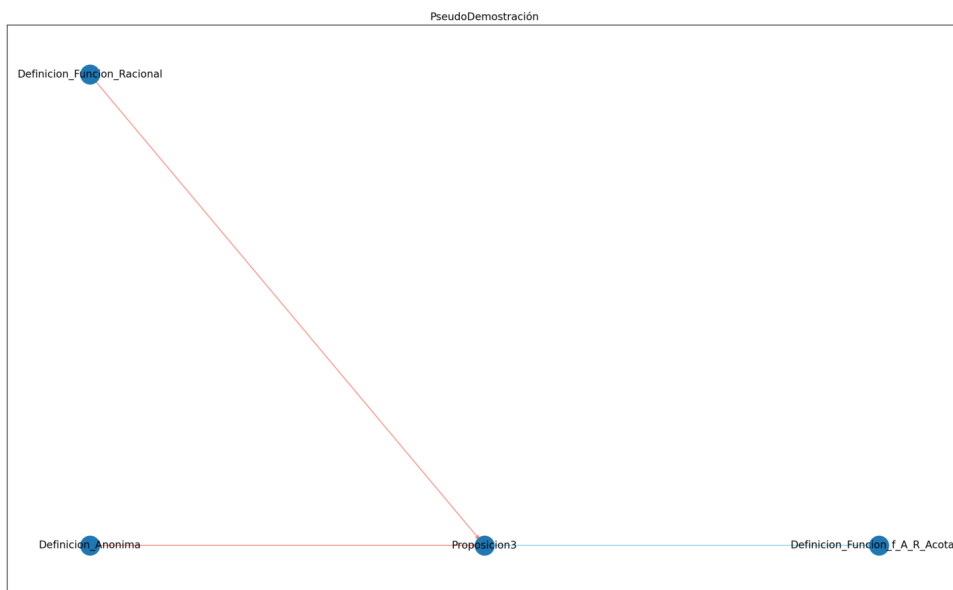


Figura 7.13: Ejemplo de grafo obtenido como pseudodemostración.

7.4.2. Aplicaciones Propuestas

A continuación, se detallarán dos de los agentes que los autores tenían en mente realizar pero que no han podido llevar a cabo por falta de tiempo. Estos agentes pretenden ilustrar el potencial de la ontología diseñada en aplicaciones prácticas.

Asistente de Profesorado

Un agente autónomo asistente de profesorado podría ser utilizado para organizar la estructura de distintas clases y explicar cadenas de razonamiento (pseudodemostraciones) dentro de un programa educativo. Por ejemplo, si las afirmaciones vistas en clase durante el curso se etiquetan adecuadamente, el asistente podría organizar los resultados que aparecerían en una pseudodemostración para un resultado concreto en un tiempo predeterminado. El asistente podría indicar cuáles resultados necesitan ser demostrados o simplemente nombrarlos, además de señalar desde qué archivo LaTeX se han añadido. Esto permitiría a los profesores planificar sus clases de manera más eficiente y asegurar que se cubren todos los conceptos necesarios en el orden lógico adecuado. Igualmente, los alumnos pueden identificar que parte de la bibliografía se emplea en cada parte de la clase de una sencilla.

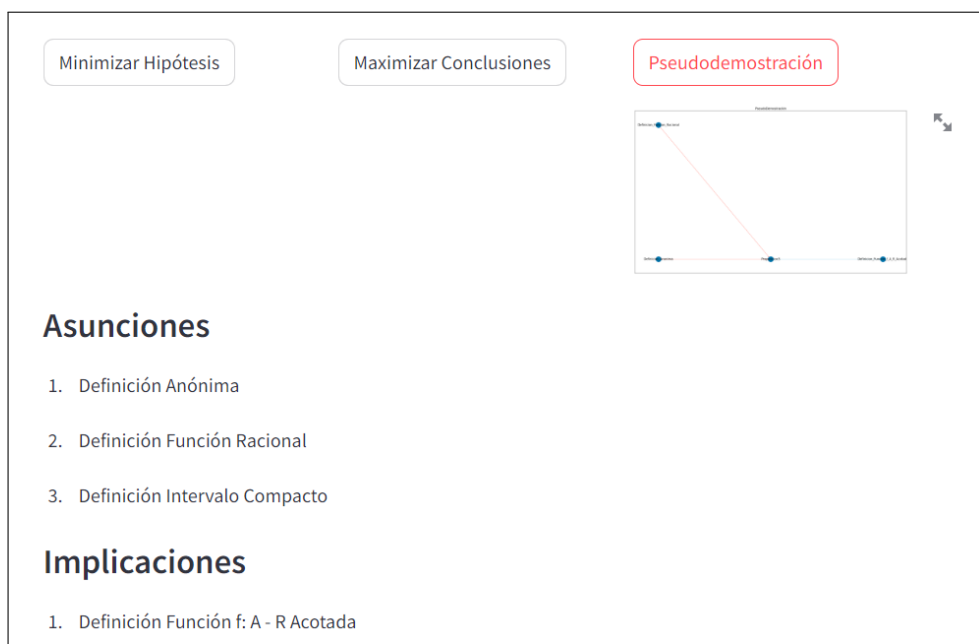


Figura 7.14: Ejemplo de pseudodemostración en la interfaz web.

Consultor de Afirmaciones Anónimas

Otro agente potencial es un consultor de afirmaciones anónimas, al cual se le podría preguntar en lenguaje natural si un conjunto de hipótesis implica un resultado específico. En primer lugar, el agente se encargaría de diferenciar las hipótesis y conclusiones de nuestra consulta y buscar los enunciados más parecidos en la base de datos vectorial. Este agente explicaría en lenguaje natural cada uno de los pasos que se dan en la pseudodemostración, calculada con la variante de Dijkstra, para llegar al resultado final. Este tipo de agente sería especialmente útil en entornos de investigación y educación, donde es crucial entender y validar cadenas de razonamiento complejas.

En esta tarea, nos apoyaríamos en la técnica de **meta prompting** [67], que consiste en usar un LLM con el prompt del usuario para elaborar un prompt más preciso y adecuado para el LLM y tarea en cuestión. Esta técnica optimiza la interacción entre el usuario y el modelo, mejorando la calidad y relevancia de las respuestas generadas por el agente autónomo.



Minimizar Hipótesis Maximizar Conclusiones Pseudodemostración

Camino no encontrado

Asunciones

1. Definición Anónima
2. Definición Intervalo Compacto

Implicaciones

1. Definición Función $f: A \rightarrow \mathbb{R}$ Acotada

Figura 7.15: Ejemplo de inexistencia del “camino” en la interfaz web.

Capítulo 8

Conclusiones y trabajo futuro

En este capítulo se resume y reflexiona sobre la investigación que hemos realizado sobre las ontologías para procesar conocimiento matemático.

8.1. Demostraciones en Vídeo

Para ilustrar las funcionalidades logradas con las herramientas estudiadas en el proyecto, se incluye un enlace a una carpeta de OneDrive donde el lector puede ver la interfaz web que hemos desarrollado en diferentes escenarios de uso: [enlace a la carpeta compartida](#).

8.2. Resumen de los puntos principales

Las ontologías son estructuras formales que permiten la representación explícita del conocimiento en un dominio específico. En el ámbito del conocimiento matemático, las ontologías permiten organizar y estructurar conceptos, relaciones y propiedades matemáticas de manera coherente y accesible.

Gracias a esta estructura, podemos realizar consultas avanzadas que permiten representar relaciones entre diversos enunciados mediante cadenas de implicaciones lógicas.

Los modelos de lenguaje grandes (LLMs) se benefician enormemente de la integración de ontologías, ya que estas proporcionan un contexto rico y

preciso. Lo que impacta positivamente tanto en la comprensión como en la generación de texto en el ámbito matemático, esto permite superar las limitaciones del enfoque puramente textual habitual en los agentes de recuperación y generación de respuestas mediante generación aumentada (Retrieval Augmented Generation, RAG).

Además, la interfaz que hemos desarrollado, aunque no ha dado tiempo a implementar todas las funcionalidades deseadas, promete ser un avance significativo para la gestión de documentos matemáticos en comparación con el enfoque probabilístico actual que delega el procesamiento de la información a los LLMs.

Se han destacado varios beneficios clave de esta integración:

- **Mejora de la coherencia lógica:** Las ontologías ofrecen un marco estructurado que ayuda a los LLMs a interpretar y generar contenido matemático con mayor coherencia.
- **Posibilita un contexto enriquecido:** La estructura jerárquica y las relaciones definidas en las ontologías permiten a los LLMs comprender el contexto de manera más profunda y detallada.
- **Permite la gestión de datos complejos:** Las ontologías facilitan la organización y el manejo de grandes volúmenes de datos complejos, mejorando la capacidad de los LLMs para procesar y utilizar esta información de manera eficiente.
- **Suponen un avance en la transparencia de los modelos:** Aunque no siempre se comprende por qué un modelo redacta una respuesta de una forma concreta, las ontologías permiten visualizar la justificación del argumento mediante referencias claras.
- **Posibilita la elaboración de estrategias a medio plazo:** Los LLMs pueden seguir estrategias a medio plazo mediante cadenas de razonamiento extraídas de la ontología.

8.3. Trabajo Futuro

La investigación sobre el futuro de la integración de ontologías y LLMs podría proporcionar avances significativos para lograr una IA capaz de manejar conceptos científicos de manera rigurosa. Numerosos aspectos sobre

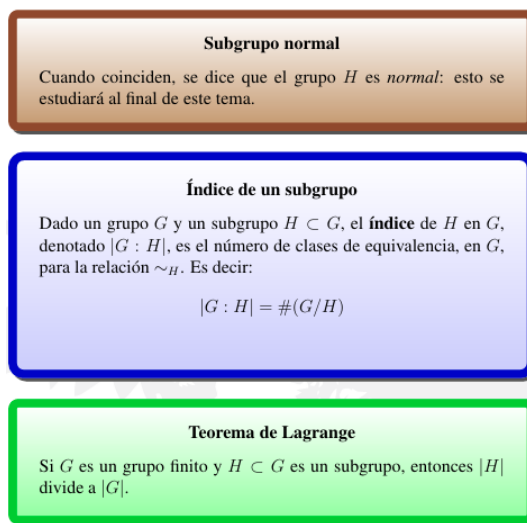


Figura 8.1: Imagen de entrada para GPT-4o

esta investigación considerados inicialmente fueron descartados por falta de tiempo. Sin embargo, este proyecto fue concebido de manera modular, donde las distintas partes se acoplan para lograr una misión global. De hecho, animamos a otros estudiantes e investigadores a implementar las piezas que no hemos podido desarrollar para lograr un conocimiento matemático más sólido y accesible, donde cualquiera pueda tener acceso a una tutoría de calidad.

Nos agrada la idea de entender la ciencia como un extenso puzzle, donde cada investigador y estudiante aporta su pieza única para construir una imagen completa y coherente del conocimiento. Esta metáfora resalta la importancia de la colaboración y la contribución individual en el avance científico. Cada pieza, por pequeña que sea, es fundamental para alcanzar una comprensión más profunda y holística. Esta idea se ilustra en la Figura 8.3, donde cada individuo trabaja en su parte del puzzle para eventualmente unir todas las piezas en un todo significativo. Nos gustaría destacar las siguientes cuestiones a considerar en futuras investigaciones:

- **Considerar el impacto de la evolución tecnológica:** Se anticipan mejoras continuas en las capacidades de los LLMs, lo que potenciará especialmente su capacidad de preprocesado de texto para poblar la ontología. Adaptar este sistema a la última tecnología es fundamental para lograr un rendimiento competente.
- **Investigaciones interdisciplinarias:** La colaboración entre disciplinas como la informática, la matemática y la lingüística será crucial para

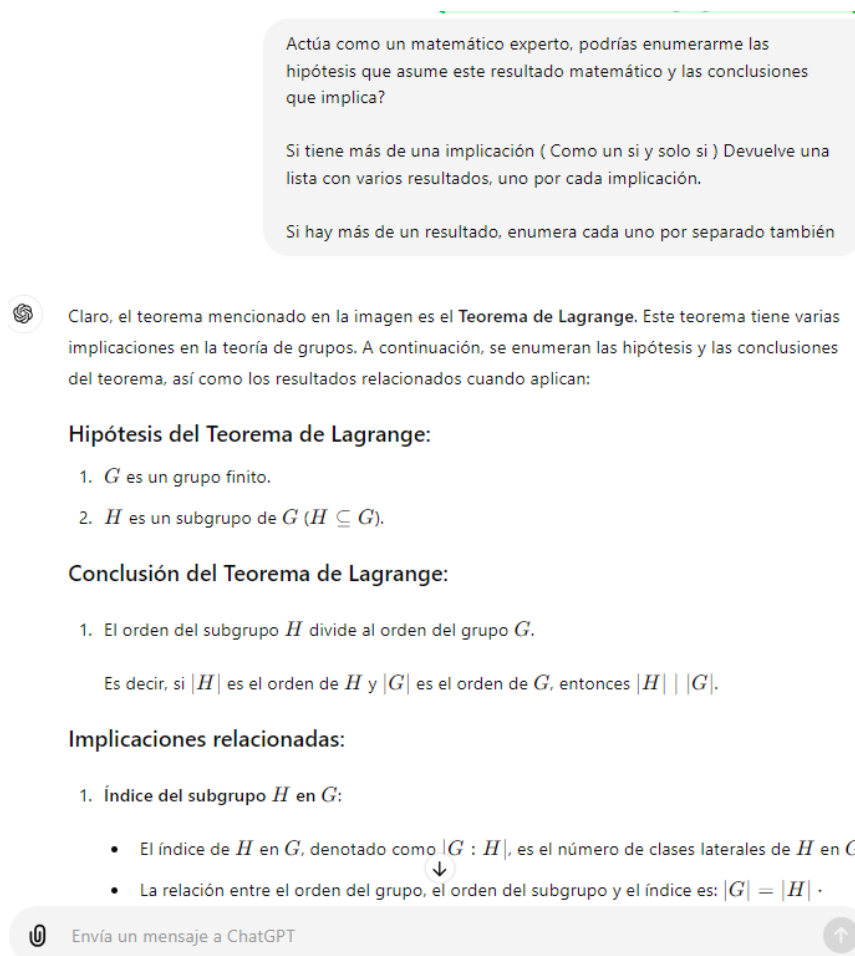


Figura 8.2: Respuesta de GPT-4o

avanzar en este campo, promoviendo una mayor comprensión y aprovechamiento del conocimiento científico. Una ontología que permita relacionar el conocimiento de distintas disciplinas sería muy útil para referenciar artículos y visualizar el conocimiento científico en general.

- **Implementar la extracción de pruebas:** Implementar la extracción de pruebas de los resultados utilizando las clases *Proof* y *MathematicalStep*, junto con las relaciones descritas en el Capítulo 3. Esto permitirá incluir una justificación clara para cada enunciado. Además, esta información ayudaría a implementar encontrar más aplicaciones a esta tecnología.
- **Mejorar la extracción de información:** Desarrollar técnicas más avanzadas para extraer automáticamente conceptos matemáticos de fuentes diversas, como libros de texto, artículos científicos y recursos en línea, para alimentar las ontologías de manera más eficiente y precisa.



Figura 8.3: Metáfora del puzzle (Generada por Dall-E 3)

- **Desarrollar ontologías especializadas:** Crear ontologías específicas para subcampos matemáticos particulares, como álgebra, geometría o análisis, para proporcionar un contexto más detallado y relevante a los LLMs en función del tipo de problema que estén abordando.
- **Optimizar de la escalabilidad:** Investigar métodos para mejorar la escalabilidad de las ontologías, incluyendo técnicas de indexación eficientes, almacenamiento de datos distribuido y procesamiento paralelo, para manejar conjuntos de datos cada vez más grandes y complejos.
- **Diseñar herramientas de evaluación automática:** Estas herramientas automatizadas permitirían evaluar la calidad y la consistencia de las ontologías y su compatibilidad con los LLMs, lo que podría incluir la detección de potenciales inconsistencias lógicas y la evaluación de la cobertura de conceptos.
- **Aplicar las ontologías en investigación sobre privacidad y ética:** Investigar cómo el uso de ontologías puede ayudar a abordar los im-

pactos éticos y sociales de la integración de LLMs, incluyendo aspectos como la privacidad de datos, el sesgo algorítmico y la equidad en el acceso al conocimiento matemático. Esto podría implicar el desarrollo de ontologías especializadas que capturen y gestionen de manera sistemática los aspectos éticos y sociales relevantes, facilitando así la implementación de políticas y prácticas que mitiguen estos desafíos.

- **Desarrollar aplicaciones prácticas para la industria:** Investigar cómo las ontologías y los LLMs pueden ser utilizados de manera efectiva en entornos empresariales, como en la automatización de tareas repetitivas y la asistencia en la toma de decisiones basada en datos.
- **Traducir enunciados a un lenguaje formal:** La traducción de los enunciados a un lenguaje formal, como la Lógica de Predicados, permite que el sistema realice deducciones y detecte contradicciones de manera precisa. Este proceso implica mapear conceptos y relaciones del lenguaje semiformal de los enunciados a una representación lógica estructurada, facilitando el análisis y la inferencia automatizada. Al utilizar un lenguaje formal, se incrementa la precisión en la interpretación y manipulación de los datos, habilitando capacidades avanzadas de razonamiento lógico dentro del sistema.

8.4. Conclusión final

La principal conclusión de nuestro trabajo es que las ontologías representan una herramienta poderosa para complementar las capacidades de los LLMs en el modelado y manejo del conocimiento matemático y científico en general.

La integración de estas tecnologías ofrece beneficios significativos, aunque también presenta desafíos que deben ser abordados mediante el desarrollo continuo de futuras investigaciones. Para ello, la colaboración interdisciplinar y el avance tecnológico serán claves para maximizar el potencial de estas herramientas y su impacto en diversas áreas del conocimiento científico y a su aplicación tecnológica.

Apéndice

Tipos de licencias de software gratuitas:

1. Licencia de Software Propietario:

Este tipo de licencia es común en software comercial y está diseñado para proteger los derechos de propiedad intelectual del desarrollador o empresa. Restringe el uso del software a una sola entidad, generalmente una empresa o un usuario individual, y prohíbe explícitamente la modificación, copia, o redistribución del software sin el permiso explícito del titular de los derechos. Es común en aplicaciones de software de escritorio y sistemas operativos.

2. Licencia de Código Abierto:

En contraste con el software propietario, el código fuente del software bajo esta licencia está disponible públicamente. Esto significa que los desarrolladores y usuarios pueden ver, modificar y distribuir el software, a menudo con la única condición de mantener el mismo nivel de apertura en las versiones modificadas. Ejemplos populares de licencias de código abierto incluyen la Licencia Pública General de GNU (GPL) [27] y la Licencia MIT [69]. Este tipo de licencia es fundamental en la comunidad de desarrollo de software, fomentando la colaboración y la innovación.

3. Licencia de Dominio Público:

Esta licencia es la más abierta de todas, ya que el software bajo dominio público es liberado de cualquier derecho de autor. Esto significa que no hay restricciones en cuanto a cómo se puede usar, modificar o distribuir el software. Es una opción popular para los desarrolladores que desean contribuir de manera altruista a la comunidad de software.

4. Licencia de Software Gratuito:

Aunque este software no tiene costo, a menudo viene con restricciones adicionales, como la prohibición de su uso en entornos comerciales o

de su redistribución con fines de lucro. Estas licencias son comunes en herramientas y aplicaciones destinadas a usuarios individuales o pequeñas empresas. No implica en principio código abierto.

5. Licencia de Evaluación:

Diseñada para permitir a los usuarios probar el software antes de comprar, esta licencia ofrece acceso completo o limitado al software durante un período de tiempo determinado. Después de este período, el usuario debe comprar una licencia completa para continuar usando el software. Es una estrategia común en software empresarial y aplicaciones profesionales.

6. Licencia de Suscripción:

Esta modalidad de licencia se ha vuelto cada vez más popular, especialmente con el auge del software como servicio (SaaS). Los usuarios pagan una tarifa recurrente, generalmente mensual o anual, para acceder al software. Este modelo a menudo incluye actualizaciones regulares y soporte técnico. Si se suspende el pago, se pierde el acceso al software.

7. Licencia de Usuario Único:

Específica para un solo usuario, esta licencia prohíbe compartir el software con otros. Es típica en software especializado o de alto costo, asegurando que cada usuario adquiera su propia licencia.

8. Licencia de Empresa:

Orientada a organizaciones, esta licencia permite a múltiples usuarios dentro de la misma empresa acceder y utilizar el software. Estas licencias suelen ser más costosas pero ofrecen una solución integrada y a menudo personalizable para las necesidades de una empresa.

9. Licencia Creative Commons:

Aunque más conocida en el ámbito de los contenidos digitales como música, imágenes y textos, algunas aplicaciones de software también utilizan este tipo de licencia. Permite a los creadores especificar cómo otros pueden usar su trabajo, como permitir el uso con atribución, prohibir el uso comercial o permitir modificaciones.

10. Licencia Copyleft:

Esta licencia es una variante específica de las licencias de código abierto. Permite la redistribución y modificación del software, con la condición de que todas las versiones modificadas estén también disponibles bajo la misma licencia. Esto asegura que el software y sus derivados permanezcan libres y abiertos para la comunidad.

Documento Latex Ideal

A continuación se proporciona el código de el documento usado para los ejemplos de este trabajo.

```
1
2 \chapter*{Conceptos sobre Funciones}
3 \begin{definicion}
4 Intervalo compacto: Un intervalo  $[a,b]$  en  $\mathbb{R}$  es
5   compacto si es cerrado y acotado.
6 \end{definicion}
7
8 \begin{definicion}
9 Función continua: Una función  $f: A \rightarrow \mathbb{R}$  es
10   continua si para todo  $\epsilon > 0$ , existe  $\delta > 0$ 
11   tal que para todo  $x, y \in A$ , si  $|x - y| < \delta$ 
12   entonces  $|f(x) - f(y)| < \epsilon$ .
13 \end{definicion}
14
15 \begin{definicion}
16 Función  $f: A \rightarrow \mathbb{R}$  acotada: Una función  $f: A \rightarrow \mathbb{R}$ 
17   es acotada si existe un número real  $M$  tal
18   que  $|f(x)| \leq M$  para todo  $x \in A$ .
19 \end{definicion}
20
21 \begin{definicion}
22 Función real: Una función  $f: A \rightarrow \mathbb{R}$  es una
23   función que asigna a cada elemento de  $A$  un único número
24   real.
25 \end{definicion}
26
27 \begin{definicion}
28 Función racional: Una función  $f: A \rightarrow \mathbb{R}$  es
29   racional si se puede expresar como el cociente de dos
30   polinomios  $P(x)$  y  $Q(x)$ , es decir,  $f(x) =$ 
31    $\frac{P(x)}{Q(x)}$  donde  $Q(x) \neq 0$  para todo  $x \in$ 
32    $A$ .
33 \end{definicion}
34
35 \begin{definicion}
36 Raíces de un polinomio: Los valores de  $x$  que hacen que
37    $P(x) = 0$  se llaman raíces del polinomio  $P(x)$ .
38 \end{definicion}
```

```
26 \begin{teorema}
27 Sea  $f: A \rightarrow \mathbb{R}$  una función racional \\
28 $$
29 \begin{cases}
30 \text{El denominador no tiene raíces en } A \\
31 \end{cases} \backslash
32 \implies
33 \backslash
34 \begin{cases}
35 f \text{ está acotada} \\
36 \end{cases}
37 $$
38 \end{teorema}
39 \begin{teorema}
40 Sea  $f: A \rightarrow \mathbb{R}$  \\
41 $$
42 \begin{cases}
43 A \text{ compacto} \\
44 f \text{ continua} \\
45 \end{cases} \backslash
46 \implies
47 \backslash
48 \begin{cases}
49 f \text{ está acotada} \\
50 \end{cases}
51 $$
52 \end{teorema}
53 \begin{teorema}
54 Sea  $f: A \rightarrow \mathbb{R}$  una función racional \\
55 $$
56 \begin{cases}
57 \text{El denominador no tiene raíces en } A \\
58 \end{cases} \backslash
59 \implies
60 \backslash
61 \begin{cases}
62 f \text{ está acotada} \\
63 \end{cases}
64 $$
65 \end{teorema}
```

Bibliografía

- [1] ACCENTURE. A new era of generative ai for everyone. <https://www.accenture.com/content/dam/accenture/final/accenture-com/document/Accenture-A-New-Era-of-Generative-AI-for-Everyone.pdf>, 2023. Accessed: 2024-05-09.
- [2] ADAMATZKY, A. Towards fungal computer. *Royal Society Publishing* (2018).
- [3] AI, C. F. Huggingface command-r:35b. <https://huggingface.co/CohereForAI/c4ai-command-r-v01>.
- [4] AMANATULLA, M. Transformer architecture explained, 2023. Accessed: 2024-05-25.
- [5] AMATRIAIN, X. Prompt design and engineering: Introduction and advanced methods, 2024.
- [6] ANTHROPIC. The claude 3 model family: Opus, sonnet, haiku. <https://paperswithcode.com/paper/the-claude-3-model-family-opus-sonnet-haiku>, 2021. Accedido el 9 de mayo de 2024.
- [7] ASCII MATH. Asciihtml.js (ver 2.0): Translating ascii math notation to mathml and graphics. <http://www1.chapman.edu/~jipsen/mathml/asciihtml.html>.
- [8] AVILA, D. Desafíos Éticos en la ia: Sesgos de género y racial en modelos de lenguaje generativos. *LatinXinAI* (Dec 2023).
- [9] BAEK, J., TSAI, J., MARABLE, J. S. M., AND CUI, Y. Amazon recommender system. https://library.ucsd.edu/dc/object/bb8503744c/_2_1.pdf.

- [10] BIRYAL'TSEV, E. V., ELIZAROV, A. M., ZHIL'TSOV, N. G., LIPACHEV, E. K., NEVZOROVA, O. A., AND SOLOV'EV, V. D. Methods for analyzing semantic data of electronic collections in mathematics. *Automatic Documentation and Mathematical Linguistics* 48, 2 (2014), 81.
- [11] BRICEÑO, B., AND FERNÁNDEZ, E. ¿qué son los word embeddings y para qué sirven? <https://blogs.iadb.org/conocimiento-abierto/es/que-son-los-word-embeddings/>.
- [12] BUSSLER, C. Vector databases (are all the rage). <https://medium.com/google-cloud/vector-databases-are-all-the-rage-872c888fa348>.
- [13] CERVONE, D., AND SORGE, V. Mathjax: The present and the future. *Notices of the American Mathematical Society* (November 2020). DOI: <https://doi.org/10.1090/noti2163>.
- [14] CHIANG, W.-L., ZHENG, L., SHENG, Y., ANGELOPOULOS, A. N., LI, T., LI, D., ZHANG, H., ZHU, B., JORDAN, M., GONZALEZ, J. E., AND STOICA, I. Chatbot arena: An open platform for evaluating llms by human preference, 2024.
- [15] COHEREAI. Cohereai prompt engineering.
- [16] COMMUNITY, S. Streamlit documentation. <https://streamlit.io/>.
- [17] CORREA BUSQUETS, S., AND MACCARINI LLORENS, L. Autosupervisión de alucinaciones en grandes modelos del lenguaje: Llteam. *Journal of Computer-Assisted Linguistic Research* 7 (2023), 60–85.
- [18] CTAN. Comprehensive T_EX Archive Network. <http://www.ctan.org/>.
- [19] DATASCIENTEST. Word2vec: Nlp y word embedding, 2024. Accedido el 9 de mayo de 2024.
- [20] DE VRIES, A. The growing energy footprint of artificial intelligence. *Joule* 7 (10 2023).
- [21] DEBELLIS, M. *A Practical Guide to Building OWL Ontologies Using Protégé 5.5 and Plugins*, 3.0 ed. April 8 2021.
- [22] DOTHINKING. Pdf2docx. <https://pypi.org/project/pdf2docx/>.
- [23] ELIZAROV, A. M., KIRILLOVICH, A. V., LIPACHEV, E. K., NEVZOROVA, O. A., SOLOVYEV, V. D., AND ZHIL'TSOV, N. G. Mathematical knowledge representation: semantic models and formalisms. *Applied Informatics* 6 (2009). URL: <https://link.springer.com/article/10.1134/S1995080214040143>.

- [24] ELIZAROV, A. M., KIRILLOVICH, A. V., LIPACHEV, E. K., NEVZOROVA, O. A., SOLOVYEV, V. D., AND ZHILTSOV, N. G. Mathematical knowledge representation: Semantic models and formalisms. *Russian Journal of Mathematical Physics* 21, 4 (2014), 517–525.
- [25] FACE, H. Chatbot arena leaderboard, 2024. Accessed: 2024-05-09.
- [26] FALCONER, S. Ontograf: Exploring ontologies with a graphical interface. <https://protegewiki.stanford.edu/wiki/OntoGraf>.
- [27] FOUNDATION, F. S. Gnu general public license.
- [28] FOUNDATION, N. Mathjax. <http://www.mathjax.org/>.
- [29] GML4GTK. Pysugiyama repository. <https://github.com/gml4gtk/pysugiyama>.
- [30] GRACIO, S. L. Reinvent your recommender system using vector database and opinion mining. <https://medium.com/dailymotion/reinvent-your-recommender-system-using-vector-database-and-opinion-mining-a4f>
- [31] HAO, Y., SUN, Y., DONG, L., HAN, Z., GU, Y., AND WEI, F. Structured prompting: Scaling in-context learning to 1,000 examples, 2022.
- [32] HENDRYCKS, D., BURNS, C., BASART, S., CRITCH, A., LI, J., SONG, D., AND STEINHARDT, J. Aligning ai with shared human values. *Proceedings of the International Conference on Learning Representations (ICLR)* (2021).
- [33] HENDRYCKS, D., BURNS, C., BASART, S., ZOU, A., MAZEIKA, M., SONG, D., AND STEINHARDT, J. Measuring massive multitask language understanding. *Proceedings of the International Conference on Learning Representations (ICLR)* (2021).
- [34] HOLGADO, R. Chatgpt reveals personal data. <https://www.20minutos.es/tecnologia/ciberseguridad/chatgpt-filtraba-por-error-datos-personales-reales-con-solo-usar-este-pequeno>
- [35] HU, H. Mathematics notes in latex. <https://github.com/HechenHu/Mathematics-Notes>.
- [36] HUGGINGFACE. trocr. https://huggingface.co/docs/transformers/model_doc/trocr.
- [37] I2OCR. Pdf ocr i2ocr. <https://www.i2ocr.com/pdf-ocr-latex-equation>.

- [38] JIN, W., CHENG, Y., SHEN, Y., CHEN, W., AND REN, X. A good prompt is worth millions of parameters: Low-resource prompt-based learning for vision-language models, 2022.
- [39] KHAWAJA, R. Demystifying embeddings 101 – the foundation of large language models. <https://datasciencedojo.com/blog/embeddings-and-llm/>, 2023.
- [40] KIEFFER, S., AVIGAD, J., AND FRIEDMAN, H. A language for mathematical knowledge management, 2011.
- [41] KINDERMANN, P. Graph visualization course. <https://www.youtube.com/@PhilippKindermann>.
- [42] KINDERMANN, P. Visualization of graphs. lecture 3: Force-directed drawing algorithms. <https://seafilerlp.net/f/1f06c2943e114d429b48/>.
- [43] KINDERMANN, P. Visualization of graphs. lecture 4: Straight-line drawings of planar graphs. <https://seafilerlp.net/f/51c1f201d4424113be3c/>.
- [44] KINDERMANN, P. Visualization of graphs. lecture 8: Hierarchical drawing. <https://seafilerlp.net/f/51c1f201d4424113be3c/>.
- [45] KNUTH, D. E. *The T_EX Book*. Addison-Wesley Publishing Company, 1986.
- [46] KOBOUROV, S. G. Spring embedders and force directed graph drawing algorithms. <https://arxiv.org/abs/1201.3011>.
- [47] KOHLHASE, M. *OMDoc – An Open Markup Format for Mathematical Documents (Version 1.2)*, vol. 4180 of *Lecture Notes in Artificial Intelligence (LNAI)*. Springer, 2006.
- [48] KUHN, S., HELMUS, T., LANCASHIRE, R. J., MURRAY-RUST, P., RZEPA, H. S., STEINBECK, C., AND WILLIGHAGEN, E. L. Chemical markup, xml, and the world wide web. 7. cmlspect, an xml vocabulary for spectral data. *J. Chem. Inf. Mod.* 47, 6 (2007), 2015.
- [49] LAB, U. K. P. Sentence transformer on pypi. <https://pypi.org/project/sentence-transformers/>.
- [50] LAMY, J.-B. owlready2 documentation. <https://owlready2.readthedocs.io/en/v0.44/>.

- [51] LANGCHAIN. Langchain custom chat. https://python.langchain.com/docs/modules/model_io/chat/custom_chat_model/.
- [52] LANGCHAIN. Langchain customllm. https://python.langchain.com/docs/modules/model_io/llms/custom_llm/.
- [53] LANGCHAIN. Langchain documentation. https://python.langchain.com/docs/get_started/quickstart.
- [54] LANGE, C. Enabling collaboration on semiformal mathematical knowledge by semantic web integration.
- [55] LEE, Y., KIM, S., YU, T., ROSSI, R. A., AND CHEN, X. Learning to reduce: Optimal representations of structured data in prompting large language models, 2024.
- [56] LI, Z., SHI, Y., LIU, Z., YANG, F., LIU, N., AND DU, M. Quantifying multilingual performance of large language models across languages. *arXiv preprint arXiv:2404.11553* (2024).
- [57] LIU, X., ZHANG, J., ZHANG, H., XUE, F., AND YOU, Y. Hierarchical dialogue understanding with special tokens and turn-level attention, 2023.
- [58] LOBO, A. Llamus wrapper. <https://gitlab.com/universidad4774909/llamuswrapper>.
- [59] LOBO, A. Sugiyama repository python3. <https://gitlab.com/universidad4774909/tfg/sugiyama-framework-python3>.
- [60] LOBO, A. Chatex web ui. <https://gitlab.com/universidad4774909/tfg/chatex-webui>, 2024.
- [61] LUO, H., AND SPECIA, L. From understanding to utilization: A survey on explainability for large language models, 2024.
- [62] MA, S., WANG, H., MA, L., WANG, L., WANG, W., HUANG, S., DONG, L., WANG, R., XUE, J., AND WEI, F. The era of 1-bit llms: All large language models are in 1.58 bits, 2024.
- [63] MATHPIX. Math pix pdf to latex. <https://mathpix.com/pdf-to-latex>.
- [64] META. Introducing meta llama 3: The most capable openly available llm to date. <https://ai.meta.com/blog/meta-llama-3/>.

- [65] MIKOLOV, T., CHEN, K., CORRADO, G., AND DEAN, J. Efficient estimation of word representations in vector space. In *Proceedings of the International Conference on Learning Representations (ICLR)* (2013).
- [66] MIKOLOV, T., SUTSKEVER, I., CHEN, K., CORRADO, G. S., AND DEAN, J. Distributed representations of words and phrases and their compositionality. *Advances in neural information processing systems* 26 (2013).
- [67] MIRAC SUZGUN, A. T. K. Meta-prompting: Enhancing language models with task-agnostic scaffolding. <https://arxiv.org/abs/2401.12954>.
- [68] MISHRA, M., STALLONE, M., ZHANG, G., SHEN, Y., PRASAD, A., AND OTROS. Granite code models: A family of open foundation models for code intelligence. *arXiv preprint arXiv:2405.04324* (2024).
- [69] MIT. Mit license. <https://opensource.org/license/MIT>.
- [70] MITTAL, A. Lora, qlora y qa-lora: Adaptabilidad eficiente en modelos de lenguaje grandes mediante factorización matricial de bajo rango. *Unite.AI* (Octubre 2023).
- [71] MUKULPATNAIK. Researchgpt. <https://github.com/mukulpatnaik/researchgpt>, 2023.
- [72] MURRAY-RUST, P. Semantic science and its communication - a personal view. *Journal of Cheminformatics* 3 (2011), 48.
- [73] NAQVI, F. A. Command-r: Un nuevo modelo de ia escalable de 35 mil millones de cohereforai para uso empresarial. <https://digialps.com/command-r-a-new-35b-scalable-ai-model-by-cohereforai-for-enterprise-usage/#h-6-licensing-and-acceptable-use>.
- [74] NEVZOROVA, O., ZHILTSOV, N., KIRILLOVICH, A., AND LIPACHEV, E. Ontomathpro ontology: A linked data hub for mathematics. In *5th International Conference, Proceedings* (2014), vol. 468, Springer International Publishing, pp. 105–119. Source: [ArXiv].
- [75] NIKOLOV, N. *Sugiyama Algorithm*. 01 2016, pp. 2162–2166.
- [76] NUCLIA. NucliaDB. <https://github.com/nuclia/nucliadb>, 2022.
- [77] NUCLIADB. Nucliadb docker image. <https://hub.docker.com/r/nuclia/nucliadb>. Accessed on April 21, 2024.
- [78] OLLAMA. Ollama. <https://github.com/ollama/ollama/tree/main>.

- [79] OPENAI. GPT-4 Technical Report. Tech. rep., OpenAI, 2024. arXiv:2303.08774 [cs.CL].
- [80] ORACLE. Mysql, 2024.
- [81] PFLAUM, M. J., AND TULEY, J. Liber mathematicae: A web-based documentation and collaboration project for mathematics, 2011.
- [82] PFLAUM, M. J., AND TULEY, J. Liber mathematicae: A web-based documentation and collaboration project for mathematics, 2011. arXiv:1102.5720.
- [83] RICHARDS, T. B. autogpt. <https://github.com/Significant-Gravitas/AutoGPT>.
- [84] R/LOCALLLAMA. Today's open source models beat closed source models from 1.5 years ago. https://www.reddit.com/r/LocalLLaMA/comments/1c33agw/todays_open_source_models_beat_closed_source/.
- [85] RONG, X. word2vec parameter learning explained. *arXiv preprint arXiv:1411.2738* (2014).
- [86] RUBIO, F. The 5 most powerful pathfinding algorithms. <https://www.graphable.ai/blog/pathfinding-algorithms/#:~:text=Dijkstra's%20algorithm%20is%20used%20to,each%20node%20in%20the%20graph>.
- [87] SAMSI, S., ZHAO, D., MCDONALD, J., LI, B., MICHALEAS, A., JONES, M., BERGERON, W., KEPNER, J., TIWARI, D., AND GADEPALLY, V. From words to watts: Benchmarking the energy costs of large language model inference. *arXiv preprint arXiv:2310.03003* (2023).
- [88] SANDERSON, G. 3blue1brown. <https://www.youtube.com/c/3blue1brown>, 2024. Canal de YouTube: 3blue1Brown.
- [89] SANDERSON, G. Pero, ¿qué es un gpt? introducción visual a los transformadores | aprendizaje profundo, capítulo 5. <https://www.youtube.com/watch?v=wjZofJX0v4M&t=431s>, 2024. Canal de YouTube: 3blue1Brown.
- [90] SANDERSON, G. Visualizar la atención, el corazón de un transformador | capítulo 6, aprendizaje profundo. <https://www.youtube.com/watch?v=eM1x5fFNoYc>, 2024. Canal de YouTube: 3blue1Brown.

- [91] SATPUTE, A., GIESSING, N., GREINER-PETTER, A., SCHUBOTZ, M., TESCHKE, O., AIZAWA, A., AND GIPP, B. Can llms master math? investigating large language models on math stack exchange, 2024.
- [92] STARDOG UNION. Pellet reasoner.
- [93] TAL, B. How to use llms for content creation: A practical guide. <https://www.linkedin.com/pulse/how-use-llms-content-creation-practical-guide-boaz-tal-q9use/>, 2023.
- [94] TANG, Y. Comparing different vector embeddings. <https://thenewstack.io/comparing-different-vector-embeddings/>.
- [95] TENSORFLOW. word2vec | text | tensorflow. <https://www.tensorflow.org/text/tutorials/word2vec>. Accedido: 2024-05-09.
- [96] TIKHONOV, A., AND YAMSHCHIKOV, I. P. Mapping the landscape of llm evaluation. *Inworld.AI* (2023).
- [97] VASWANI, A., SHAZEER, N., PARMAR, N., USZKOREIT, J., JONES, L., GOMEZ, A. N., KAISER, L., AND POLOSUKHIN, I. Attention is all you need. *Advances in neural information processing systems* 30 (2017).
- [98] VYAS, K. 8 big advantages of using mysql. *Datamation* (2023).
- [99] WEI, D. Creating rag agent. <https://medium.com/@weidagang/chat-with-your-own-data-through-llm-rag-a47196a5c440>.
- [100] WILLIAMS, E. El rol de las bases de datos vectoriales en las aplicaciones contemporáneas de inteligencia artificial generativa. <https://vivevirtual.es/noticias-ia/el-rol-de-las-bases-de-datos-vectoriales-en-las-aplicaciones-contemporaneas-de->
- [101] WOLFRAM. Computable document format (cdf) for interactive content. URL: <http://wolfram.com/cdf>.
- [102] WOLFRAM, S. Wolfram mathematica. <http://www.wolfram.com/mathematica/>.
- [103] WOLFRAM, S. Wolframalpha computational knowledge engine. <http://www.wolframalpha.com/>.
- [104] WOLFRAM, S. *A New Kind of Science*. Wolfram Media, Inc., 2002.

-
- [105] XIAN, Y., SCHIELE, B., AND AKATA, Z. Zero-shot learning - the good, the bad and the ugly. *arXiv: Computer Vision and Pattern Recognition arXiv:1703.04394v2* (2020).
- [106] YE, Q., AXMED, M., PRYZANT, R., AND KHANI, F. Prompt engineering a prompt engineer, 2024.
- [107] YOUNES, A. Chat with your data. <https://github.com/AlmutazYounes/chat-with-your-data>.