



UNIVERSITAT
POLITÈCNICA
DE VALÈNCIA



UNIVERSITAT POLITÈCNICA DE VALÈNCIA

Escuela Técnica Superior de Ingeniería Informática

Estudio comparativo del uso de modelos de lenguaje en
sistemas multi-agente basados en SPADE

Trabajo Fin de Grado

Grado en Ingeniería Informática

AUTOR/A: Fernández Herreruela, Ismael

Tutor/a: Carrascosa Casamayor, Carlos

CURSO ACADÉMICO: 2024/2025

Resum

Aquest treball presenta el desenvolupament d'una eina interactiva i automatitzada per a l'avaluació de models de llenguatge natural. Mitjançant una arquitectura multi-agent basada en SPADE, s'integren models locals i per API per a realitzar proves comparatives utilitzant mètriques com BERTScore i avaluacions amb el model Gemini. El sistema permet analitzar de manera qualitativa i quantitativa el rendiment de models SLM i LLM, oferint resultats visuals i una interfície accessible per a l'usuari.

Paraules clau: Sistemes multi-agent, LLM, SLM, Mètriques

Resumen

Este trabajo presenta el desarrollo de una herramienta interactiva y automatizada para la evaluación de modelos de lenguaje natural. Mediante una arquitectura multiagente basada en SPADE, se integran modelos locales y por API para realizar pruebas comparativas utilizando métricas como BERTScore y evaluaciones con el modelo Gemini. El sistema permite analizar de forma cualitativa y cuantitativa el rendimiento de modelos SLM y LLM, ofreciendo resultados visuales y una interfaz accesible para el usuario.

Palabras clave: Sistemas multi-agente; LLM; SLM; Métricas

Abstract

This project presents the development of an interactive and automated tool for evaluating natural language models. Using a multi-agent architecture built with SPADE, both local and API-based models are integrated to perform comparative tests using metrics such as BERTScore and assessments with the Gemini model. The system enables both qualitative and quantitative analysis of SLM and LLM models, providing visual results and an accessible user interface

Key words: Multi-agent systems, LLM, SLM, Metrics

Índice general

Índice general	V
Índice de figuras	VII
Índice de tablas	VIII
1 Introducción	1
1.1 Motivación	1
1.2 Objetivos	1
1.3 Estructura de la memoria	2
2 Estado del arte	5
2.1 Introducción	5
2.2 Contexto tecnológico	5
2.3 Evolución de los Modelos de Lenguaje	5
2.4 Tendencias actuales	6
2.5 Impacto en la Industria y la Sociedad	6
2.6 Sistemas de Evaluación y Métricas	7
2.7 Crítica a las Soluciones Existentes	7
2.8 Desafíos y Oportunidades Futuras	7
2.9 Conclusiones	7
3 Presentación del problema	9
3.1 Introducción	9
3.2 Solución Propuesta	9
3.3 Plan de Trabajo	10
4 Tecnología utilizada	13
4.1 Introducción	13
4.2 SPADE (Smart Python Agent Development Environment)	13
4.2.1 Características principales	13
4.3 Python	14
4.4 Hugging face	14
4.4.1 Características Principales	14
4.5 GitHub	15
4.6 Gemini	15
4.7 BERTScore	15
4.8 SLM Y LLM	16
4.8.1 ¿Qué Són?	16
4.8.2 SLM	16
4.8.3 LLM	17
4.8.4 Integración en el proyecto	17
5 Diseño de la solución	19
5.1 Introducción	19
5.2 Diseño	19
5.3 Métricas	20
5.4 Soluciones	21

6	Desarrollo v1.0: diálogo modelo frente a humano en SPADE	23
6.1	Introducción	23
6.2	Arquitectura del sistema	23
6.2.1	Diseño base	23
6.2.2	Gestión de modelos	23
6.3	Proceso de integración	24
6.3.1	Inicialización y carga	24
6.3.2	Generación de respuestas de ChatBot	24
6.4	Sistema de evaluación	24
6.4.1	Métricas y análisis	24
6.5	Gestión de recursos	25
6.6	Interfaz de Usuario	25
6.6.1	Sistema de Interacción	25
6.6.2	Selección de modelo	25
6.6.3	Chat	26
6.6.4	Métricas Humanas y Scores	27
6.6.5	Métricas Generales	28
6.6.6	Pruebas	29
6.6.7	Pruebas con API	29
6.7	Pruebas	30
6.8	Puntos a destacar en la aproximación 1	33
7	Desarrollo v2.0: dialogo entre modelos en SPADE	35
7.1	Introducción	35
7.2	Funcionamiento	35
7.3	Estructura multiagente	35
7.4	Envío de mensajes y procesamiento de pruebas	35
7.5	Interfaz	36
7.5.1	Cambios en mostrar resultados	36
7.5.2	Mejoras en pruebas locales	36
7.5.3	Lanzar agentes y preguntas	37
7.5.4	Gráficos	40
7.6	Pruebas	41
7.6.1	Batería de pruebas predefinidas	41
7.6.2	Pruebas automatizadas en agentes hijo	43
7.7	Puntos a destacar en la aproximación 2	44
8	Conclusiones	47
	Bibliografía	49
<hr/>		
	Apéndices	
A	ODS	51
B	Manual de instalación	53
B.1	Descarga del trabajo	53
B.1.1	Acceso a github	53
B.1.2	Clonación del repositorio	54
B.2	Puesta en marcha	55

Índice de figuras

6.1	Muestra de ChatBot con botones para evaluación humana	24
6.2	Muestra de métricas con BERTscore	25
6.3	Muestra de interfaz web para seleccionar modelo	26
6.4	Muestra de interfaz web con desplegable para seleccionar modelo	26
6.5	Muestra de interfaz web con chat sin seleccionar modelo	27
6.6	Muestra de interfaz web con chat y modelo seleccionado	27
6.7	Muestra de interfaz web con métricas humanas y porcentajes de BERTscore	28
6.8	Muestra de interfaz web con métricas generales	28
6.9	Muestra de interfaz web con resultados de pruebas locales	29
6.10	Muestra de interfaz web con formulario para pruebas API	30
6.11	Muestra de interfaz web con resultados de pruebas API	30
7.1	Muestra de interfaz web con resultados de pruebas	36
7.2	Muestra de interfaz web con formulario para modificar casos de prueba	37
7.3	Muestra de interfaz web con formulario para lanzar agentes	38
7.4	Muestra de interfaz web con formulario para lanzar agentes con listado de modelos	38
7.5	Muestra de interfaz web con formulario para lanzar agentes con listado de modelos	38
7.6	Muestra de interfaz web con formulario para lanzar agentes con modelo API	39
7.7	Muestra de interfaz web con formulario para elegir número de preguntas	39
7.8	Muestra de terminal con informacion sobre el envio de preguntas	40
7.9	Muestra de terminal con información sobre la evaluación de la pregunta	40
7.10	Muestra respuesta de modelo en interfaz web	40
7.11	Muestra de la interfaz web con los gráficos	41
B.1	Repositorio de GitHub	53
B.2	Link para poder clonar repositorio	54
B.3	Comando para clonar el repositorio	54
B.4	Paso para abrir terminal en directorio	55
B.5	Paso para ejecutar un entorno virtual	56
B.6	Paso para entrar en el entorno virtual	56
B.7	Paso para descargar dependencias	57
B.8	Paso para lanzar SPADE	57
B.9	Paso para lanzar la herramienta	58

Índice de tablas

6.1	Preguntas de las pruebas de la primera aproximación	32
6.2	Resultados pruebas primera aproximación	33
6.3	Promedio de scores por modelo (Primera aproximación)	33
7.1	Listado de preguntas y respuestas esperadas utilizadas para la evaluación	42
7.2	Evaluación comparativa de modelos	42
7.3	Evaluación comparativa de modelos por tópicos	44

CAPÍTULO 1

Introducción

1.1 Motivación

En los últimos años, los modelos de lenguaje han experimentado una evolución muy rápida y avanzada, transformando el ámbito del Procesamiento del Lenguaje Natural (PLN) y la Inteligencia Artificial (IA). Gracias a los avances en plataformas extendidas globalmente como ChatGPT, Gemini o Copilot, ha aumentado considerablemente el uso de los modelos en la población, alimentando así al desarrollo de estas tecnologías.

Este TFG pretende adentrarse en este contexto de desarrollo de los modelos, con el objetivo principal de desarrollar una herramienta con las capacidades para evaluar y comparar diferentes modelos de lenguaje, tanto locales como modelos en API y de diferentes tamaños (SLM y LLM). Todo esto integrado en un entorno multiagente desarrollado con SPADE. La necesidad de sistemas de evaluación tanto cualitativa como cuantitativa sobre los modelos, ha sido la principal motivación para este proyecto. Todo esto se apoya en la experiencia del usuario mediante métricas subjetivas y una interfaz interactiva.

El trabajo se desarrolla en un enfoque iterativo, pasando por una primera versión centrada en un primer contacto entre el usuario y los modelos a través de un chat, y, continuando con una segunda versión con una arquitectura multiagente que permitirá realizar pruebas automatizadas concurrentemente sobre diferentes agentes. Para ello, se han integrado métricas avanzadas como BERTScore y se ha aprovechado el poder de modelos avanzados como Gemini de Google.

1.2 Objetivos

El principal objetivo de este trabajo es desarrollar una herramienta comparativa de modelos de lenguaje natural de forma sistemática y automatizada, facilitando así el análisis cualitativo y cuantitativo del rendimiento de modelos SLM (Small Language Models) Y LLM (Large Language Models).

Para alcanzar este objetivo general, se definen los siguientes objetivos:

1. Diseñar e implementar una arquitectura multiagente utilizando el framework SPADE que permita gestionar, lanzar y coordinar agentes con diferentes modelos de lenguaje integrados.
2. Integrar modelos de lenguaje locales y mediante API, permitiendo realizar pruebas con una amplia variedad de arquitecturas y tamaños de modelo, como distilGPT2, GPT-neo, DialoGPT o Gemini.

3. Establecer un sistema de evaluación robusto, que incluya tanto métricas automáticas (como BERTScore) como métricas manuales (evaluación subjetiva del usuario), para analizar aspectos como coherencia, relevancia y utilidad de las respuestas generadas.
4. Desarrollar una interfaz web interactiva que permita al usuario:
 - Seleccionar modelos
 - Interactuar con ellos mediante chat
 - Consultar resultados y estadísticas de evaluación de forma clara y visual.
5. Automatizar el proceso de pruebas mediante la ejecución de test desde agentes padres hacia agentes hijos, gestionando la generación de preguntas, la recolección de respuestas y su posterior análisis.
6. Comparar el rendimiento entre modelos locales y remotos, proporcionando información útil para la elección del modelo más adecuado según el contexto de uso, recursos disponibles y calidad esperada.
7. Documentar todo el desarrollo del sistema y su funcionamiento, asegurando su mantenibilidad, posibilidad de ampliación y reutilización futura por parte de otros desarrolladores o investigadores.

1.3 Estructura de la memoria

La memoria se organiza en ocho capítulos, cada uno enfocado en una parte esencial del desarrollo del proyecto:

- Capítulo 1 – Introducción: Presentación del contexto del proyecto, su motivación, los objetivos propuestos y la estructura general de la memoria.
- Capítulo 2 – Estado del arte: Análisis de los principales avances en modelos de lenguaje, su evolución reciente, las métricas de evaluación existentes, así como los desafíos actuales y oportunidades futuras en el ámbito del procesamiento del lenguaje natural.
- Capítulo 3 – Presentación del problema: Se exponen los retos específicos que aborda el proyecto, así como la solución propuesta y la planificación estructurada del trabajo desarrollado.
- Capítulo 4 – Tecnología utilizada: Se detallan las herramientas, lenguajes, frameworks y bibliotecas empleadas para el desarrollo del sistema, como SPADE, Python, Hugging Face, BERTScore, entre otras.
- Capítulo 5 – Diseño de la solución: Descripción de la arquitectura general del sistema y el enfoque iterativo adoptado, incluyendo el uso combinado de métricas automáticas y subjetivas para la evaluación de modelos.
- Capítulo 6 – Integración de modelos en SPADE v1.0: Primera versión funcional del sistema, enfocada en la interacción directa con modelos locales mediante interfaz web, junto con las herramientas de evaluación iniciales.
- Capítulo 7 – Integración de modelos en SPADE v2.0: Segunda versión del sistema, basada en una arquitectura multiagente distribuida, con capacidad para realizar pruebas concurrentes automatizadas y evaluaciones más complejas.

- Capítulo 8 – Conclusiones: Conclusiones finales del proyecto, las principales aportaciones realizadas, y se valoran los resultados obtenidos en relación con los objetivos iniciales.

CAPÍTULO 2

Estado del arte

2.1 Introducción

El capítulo aborda los avances más importantes en modelos de lenguaje durante los últimos años. Explica los conceptos de modelos de lenguaje pequeños (SLM) y grandes (LLM), revisa su evolución y analiza las principales métricas utilizadas para su evaluación. Además, presenta una crítica fundamentada sobre las limitaciones de los sistemas de evaluación actuales y reflexiona sobre las posibles direcciones futuras del campo, tanto a nivel tecnológico como social.

2.2 Contexto tecnológico

La inteligencia artificial ha experimentado una transformación significativa gracias a los avances en modelos de lenguaje. Tecnologías como BERT [3], GPT [2] y T5 [10] han redefinido la forma en que las máquinas procesan y generan lenguaje natural. Estas innovaciones han ampliado el alcance de tareas como la comprensión de texto, la traducción automática o la generación de respuestas, permitiendo la creación de nuevas aplicaciones y servicios inteligentes. La rapidez con la que esta tecnología ha evolucionado ha superado las expectativas iniciales, marcando un punto de inflexión en el desarrollo de soluciones basadas en IA.

2.3 Evolución de los Modelos de Lenguaje

El desarrollo de los modelos de lenguaje ha estado marcado tanto por avances en su arquitectura como por su creciente escalabilidad. Un ejemplo clave es BERT [12], que introdujo el aprendizaje contextualizado y permitió obtener representaciones lingüísticas más precisas y relevantes [4]. A partir de estos avances, ha sido posible distinguir dos grandes tipos de modelos: los Small Language Models (SLM) y los Large Language Models (LLM).

Los SLM, como DistilGPT2 o DialoGPT-small [16], han sido optimizados para funcionar en entornos con recursos limitados o en aplicaciones que requieren respuestas rápidas, sin comprometer en exceso la calidad. Por otro lado, los LLM, como GPT-4 ¹ o Claude ², incorporan arquitecturas más complejas que, si bien exigen una mayor capacidad computacional, ofrecen resultados notablemente superiores en tareas lingüísticas

¹<https://openai.com/research/gpt-4>

²<https://www.anthropic.com/index/introducing-claude>

avanzadas. Esta diferenciación ha dado lugar a un nuevo enfoque en el desarrollo de aplicaciones basadas en inteligencia artificial, en el que la elección del modelo depende de un equilibrio entre precisión, velocidad de respuesta y eficiencia en el uso de recursos.

2.4 Tendencias actuales

En el panorama actual de la inteligencia artificial, una de las tendencias más destacadas es la consolidación de dos enfoques diferenciados en el desarrollo de modelos de lenguaje: los Small Language Models (SLM) y los Large Language Models (LLM). Esta división no solo responde a criterios técnicos, sino que refleja prioridades distintas dentro del ecosistema de investigación y desarrollo.

Por un lado, los SLM han ganado protagonismo por su capacidad de ofrecer buenos resultados con un coste computacional reducido. Su diseño los hace especialmente útiles en contextos donde se requiere rapidez, bajo consumo energético o integración en dispositivos con recursos limitados. Esta línea responde a una preocupación creciente por la sostenibilidad y la democratización del acceso a la IA.

En paralelo, los LLM continúan marcando hitos en tareas de lenguaje natural gracias a su escala y versatilidad. La mejora en la generación de texto, la comprensión semántica y la capacidad para manejar instrucciones complejas ha impulsado su adopción en entornos corporativos, servicios digitales y herramientas de productividad. Modelos como GPT-2 [9] y GPT-3 [1] abrieron el camino a esta tendencia, que hoy sigue avanzando con rapidez.

En conjunto, estas dos corrientes reflejan una tensión productiva entre eficiencia y potencia. Las decisiones de diseño y despliegue ya no dependen únicamente de qué modelo ofrece 'más', sino de cuál se adapta mejor a las necesidades específicas de cada aplicación, al tiempo que se consideran factores como la equidad, la privacidad y el impacto ambiental. Esta tendencia hacia una IA más estratégica, adaptable y consciente está redefiniendo el desarrollo de modelos de lenguaje en la actualidad.

2.5 Impacto en la Industria y la Sociedad

Desarrollar sistemas que permiten evaluar cómo funcionan los modelos de lenguaje tiene un impacto muy grande, tanto en la industria como en la sociedad. En el mundo empresarial, contar con herramientas de evaluación ayuda a las empresas a adoptar tecnologías de inteligencia artificial con más confianza, ya que obtienen métricas claras sobre el rendimiento de los modelos y su precisión. Gracias a eso, pueden tomar decisiones más meditadas sobre si integrar un modelo en su negocio, o qué modelo en concreto integrar.

Por otro lado, en el ámbito social, mejorar la calidad de los sistemas de inteligencia artificial está cambiando la manera en que nos relacionamos con la tecnología. Cada vez estas herramientas son más accesibles, de mejor calidad y fáciles de usar. Mientras que son realmente útiles para ciertas tareas o ámbitos, también hay que destacar que por culpa del uso en exceso de estas tecnologías hoy en día, esta empeorando las capacidades cognitivas de las personas [6].

2.6 Sistemas de Evaluación y Métricas

La forma en la que evaluamos a los modelos de lenguaje ha dado un gran salto en los últimos años. Ya no nos conformamos solo con que las respuestas "suenen bien"; ahora utilizamos métricas automáticas más avanzadas como BLEU [8] y BERTScore [15], además de otras como BLEURT [13] o COMET [11], que permiten una evaluación objetiva y semántica de la calidad del texto generado. Estas herramientas nos ayudan a determinar si una respuesta es coherente, relevante y de calidad, yendo más allá de lo superficial. Además, hoy en día contamos con interfaces de usuario mucho más amigables, que hacen que evaluar modelos sea casi tan fácil como chatear con ellos. Gracias a estas mejoras, la evaluación de modelos de lenguaje se ha vuelto más accesible y útil para investigadores, desarrolladores y cualquier persona interesada en el área.

2.7 Crítica a las Soluciones Existentes

Aunque las herramientas que usamos hoy para evaluar modelos de lenguaje han avanzado mucho, todavía tienen sus puntos débiles. Por ejemplo, algunas métricas como BLEU, que compara texto a nivel de palabras o frases, pueden pasar por alto respuestas que usan sinónimos o una estructura distinta, pero que comunican exactamente lo mismo. Es como penalizar a alguien por decir "estoy feliz" en lugar de "me siento alegre". Además, muchos sistemas de evaluación automatizada tienden a ser bastante rígidos. No siempre se adaptan bien a las necesidades de distintos contextos. Por ejemplo, lo que funciona para evaluar un chatbot médico no necesariamente sirve para calificar historias creativas o respuestas educativas. Aún nos falta dar con un enfoque más flexible y sensible al propósito.

2.8 Desafíos y Oportunidades Futuras

El futuro de los modelos de lenguaje es tan desafiante como una fuente de oportunidades. La necesidad de modelos más eficientes y accesibles ha llevado al desarrollo de nuevas arquitecturas y técnicas de optimización. La integración de diferentes tipos de modelos en sistemas representa una oportunidad para crear soluciones más versátiles y adaptables. Además, el desarrollo de nuevas métricas de evaluación y sistemas de prueba permitirá una mejor comprensión y mejora de los modelos.

2.9 Conclusiones

Hoy en día, el mundo de los modelos de lenguaje está en pleno auge: no paran de aparecer mejoras tanto en lo que pueden hacer estos modelos como en las formas de evaluarlos. La convivencia entre los modelos pequeños (SLM) y los grandes (LLM) ha generado un ecosistema súper variado, donde cada uno tiene sus beneficios y sus contrapartes. Todavía hay desafíos importantes por resolver, como lograr modelos más eficientes, accesibles para todos y que puedan evaluarse de forma justa y útil. Lo interesante es que todo indica que se vienen avances aún más importantes, con el potencial de transformar por completo la manera en que usamos y nos relacionamos con la tecnología cada día.

CAPÍTULO 3

Presentación del problema

3.1 Introducción

En el ámbito de la Inteligencia Artificial y el Procesamiento del Lenguaje Natural (PLN), existe una necesidad creciente de evaluar y comparar diferentes modelos de lenguaje, tanto locales como basados en API. Este proyecto aborda tres desafíos principales:

1. Evaluación de Calidad: ¿Cómo medir objetivamente la calidad de las respuestas de diferentes modelos de lenguaje?
2. Comparación de Arquitecturas: ¿Cómo se comparan los modelos locales con los servicios basados en API?
3. Automatización de Pruebas: ¿Cómo automatizar el proceso de evaluación para obtener resultados consistentes y reproducibles?

En este capítulo se describe con mayor detalle el problema que se ha planteado en el proyecto. Se expone la necesidad de contar con un sistema que permita comparar modelos de lenguaje de forma automatizada y rigurosa, y se propone una solución basada en el uso de agentes inteligentes que actúan de forma colaborativa. Además, se presenta el plan de trabajo que se ha seguido a lo largo del desarrollo, estructurado en distintas etapas que permiten entender cómo se ha organizado y ejecutado el proyecto de manera progresiva.

3.2 Solución Propuesta

El sistema está diseñado con una arquitectura que combina modelos de lenguaje locales, una API externa y una base de conocimiento. En el modelo local, se implementan modelos de lenguaje utilizando la biblioteca Transformers. Además, se integra con la API externa de Gemini de Google para realizar comparaciones entre respuestas generadas. Como referencia, el sistema también cuenta con una base de conocimiento, compuesta por un conjunto de preguntas y respuestas predefinidas.

Para evaluar el rendimiento del sistema, se emplean diversas métricas de evaluación. Entre ellas se encuentra BERTScore, que se utiliza para medir la similitud semántica entre las respuestas generadas y las respuestas esperadas. La precisión evalúa qué tan cercanas son las respuestas a las esperadas, mientras que la consistencia mide la estabilidad de las respuestas cuando se realizan múltiples ejecuciones del sistema.

En cuanto a sus características principales, el sistema ofrece una interfaz web que facilita la interacción del usuario. También incorpora un mecanismo de almacenamiento de resultados para su posterior análisis, y cuenta con herramientas de visualización de métricas, que permiten presentar los resultados de manera clara y comprensible.

3.3 Plan de Trabajo

Para abordar todo el proceso de creación y documentación de este trabajo, se ha segmentado en cinco etapas. Esta separación ha permitido llevar un orden y una mejor organización para poder realizar el trabajo de una manera eficiente y tener claro todo lo que se quería conseguir. Cada etapa está diseñada para poder realizar un desarrollo continuo y con un orden lógico.

La primera etapa está enfocada en la investigación y el análisis, estableciendo las bases teóricas y técnicas necesarias para el desarrollo. La segunda etapa trata sobre la creación inicial del sistema, implementando las funciones principales y la conexión con APIs externas. La tercera etapa se dedica al desarrollo de las funciones específicas del sistema de evaluación y la interfaz para el usuario. La cuarta etapa se centra en las pruebas y la mejora del sistema, garantizando su calidad y eficacia. Finalmente, la quinta etapa se encarga de la documentación y la finalización del proyecto, que incluye la redacción del informe de TFG.

Cada etapa ha sido programada con un tiempo estimado de un mes, lo que permite un desarrollo cuidadoso y sistemático. Las horas asignadas a cada actividad han sido calculadas según su complejidad y los estándares de calidad requeridos. Este enfoque ayuda a mantener un ritmo de desarrollo constante, a la vez que asegura la calidad del producto final.

Fase 1: Investigación y Análisis (Mes 1)

- Semana 1-2: Revisión Bibliográfica
 - Análisis de trabajos previos en la ETSINF (20 horas)
 - Estudio de frameworks y tecnologías existentes (15 horas)
 - Documentación de requerimientos (10 horas)
- Semana 3-4: Diseño Inicial
 - Definición de arquitectura del sistema (15 horas)
 - Diseño de la base de datos (10 horas)
 - Planificación de la interfaz de usuario (10 horas)

Fase 2: Desarrollo Base (Mes 2)

- Semana 5-6: Implementación Core
 - Desarrollo del sistema de agentes con SPADE (25 horas)
 - Implementación de la lógica de evaluación (20 horas)
 - Configuración del entorno de desarrollo (5 horas)
- Semana 7-8: Integración de APIs

- Integración con Hugging Face (15 horas)
- Implementación de la comunicación con APIs (20 horas)
- Desarrollo de sistema de caché (10 horas)

Fase 3: Desarrollo de Funcionalidades (Mes 3)

- Semana 9-10: Sistema de Evaluación
 - Implementación de métricas de similitud (20 horas)
 - Desarrollo del sistema de comparación (15 horas)
 - Integración con base de conocimiento (10 horas)
- Semana 11-12: Interfaz de Usuario
 - Desarrollo de templates HTML (15 horas)
 - Implementación de estilos CSS (10 horas)
 - Desarrollo de funcionalidades JavaScript (20 horas)

Fase 4: Testing y Optimización (Mes 4)

- Semana 13-14: Testing
 - Desarrollo de casos de prueba (15 horas)
 - Ejecución de pruebas unitarias (10 horas)
 - Pruebas de integración (15 horas)
- Semana 15-16: Optimización
 - Optimización de rendimiento (20 horas)
 - Mejora de la interfaz de usuario (10 horas)
 - Corrección de errores (15 horas)

Fase 5: Documentación y Finalización (Mes 5)

- Semana 17-18: Documentación Técnica
 - Documentación del código (15 horas)
 - Manual de usuario (10 horas)
 - Documentación de API (10 horas)
- Semana 19-20: Documentación del TFG
 - Redacción de memoria (30 horas)
 - Preparación de presentación (15 horas)
 - Revisión final (10 horas)

CAPÍTULO 4

Tecnología utilizada

4.1 Introducción

En este capítulo se explican las herramientas y tecnologías empleadas durante el proyecto. Se habla de SPADE como plataforma para construir sistemas multiagente, de Python y sus bibliotecas para la manipulación de modelos de lenguaje, de Hugging Face como repositorio de modelos preentrenados y del uso de métricas como BERTScore y Gemini para la evaluación. También se comenta cómo se ha integrado todo esto en una infraestructura coherente y funcional.

4.2 SPADE (Smart Python Agent Development Environment)

SPADE (Smart Python Agent Development Environment) es un framework de desarrollo de agentes inteligentes que implementa el protocolo XMPP¹ (Extensible Messaging and Presence Protocol)[7]. Fue diseñado para facilitar la creación de sistemas multiagente en Python, proporcionando una infraestructura robusta y flexible para la comunicación y coordinación entre agentes.

4.2.1. Características principales

La arquitectura del sistema está basada en un enfoque multiagente que posibilita la creación y gestión de múltiples agentes autónomos. Estos agentes operan de forma independiente y se comunican mediante un sistema distribuido que implementa comunicación asíncrona, permitiendo que los mensajes se envíen y reciban sin necesidad de sincronización inmediata. Además, el sistema incorpora un mecanismo de gestión de presencia que monitorea en todo momento el estado y la disponibilidad de cada agente. Para facilitar la interoperabilidad y estandarizar las interacciones, se utiliza el protocolo XMPP como base para la comunicación entre los agentes.

Entre los componentes básicos del sistema se encuentran los agentes, que son entidades autónomas con capacidad de tomar decisiones propias. Cada uno de estos agentes opera según ciertos comportamientos predefinidos, llamados 'behaviours', los cuales determinan su lógica de actuación. La interacción entre agentes se lleva a cabo a través de un sistema de mensajería que organiza y dirige los mensajes que se intercambian, mientras que la presencia mantiene actualizada la información sobre el estado operativo de cada agente.

¹<https://xmpp.org>

En cuanto a las funcionalidades clave, el sistema ofrece un robusto entorno de comunicación que incluye mensajería asíncrona y comunicación bidireccional, lo que permite una interacción fluida y continua entre los distintos agentes. La gestión de colas de mensajes asegura que los intercambios de información se procesen de manera ordenada, y los protocolos de comunicación estandarizados refuerzan la compatibilidad y escalabilidad del sistema. Por otro lado, la gestión de agentes incluye la capacidad de crear y eliminar agentes dinámicamente, monitorear su estado, coordinar recursos entre ellos y distribuir la carga de trabajo de manera equitativa para mantener el equilibrio del sistema y optimizar su rendimiento general

4.3 Python

Python² es un lenguaje muy conocido hoy en día, y a su vez de los más utilizados, no solo por su poca complejidad, sino por su libertad a la hora de utilizar librerías creadas por usuarios. Hay que remarcar que ofrece la posibilidad de ejecutar tareas de manera asíncrona y concurrente sin llegar a bloqueos y de una manera muy eficiente. Al tratarse de uno de los lenguajes más populares, es también uno de los más aprendidos en ámbitos académicos y personales, lo que beneficia a la hora de querer trabajar en equipo o permitir que más gente pueda analizar o investigar el código. En nuestro caso haremos especial énfasis en la biblioteca "Transformers", la cual nos permite la interacción con los modelos.

Particularmente, utilizamos la biblioteca Transformers para trabajar con modelos de lenguaje, apoyándonos en PyTorch como framework principal de deep learning. Para la evaluación de la similitud entre respuestas, integramos BERTScore, mientras que para el desarrollo web asíncrono se empleó aiohttp³, lo que permitió gestionar múltiples peticiones simultáneamente de forma eficiente.

4.4 Hugging face

Hugging Face⁴ es una plataforma líder en el desarrollo y distribución de modelos de lenguaje basados en transformadores. Fundada en 2016, se ha convertido en un referente en el campo de la Inteligencia Artificial, especialmente en Procesamiento de Lenguaje Natural (NLP).

4.4.1. Características Principales

Biblioteca Transformers

La integración de modelos se ha realizado a través de la biblioteca Transformers de Hugging Face [14], facilitando el acceso a modelos preentrenados y su despliegue en producción.

Una de las principales ventajas de Transformers es la disponibilidad de una amplia colección de modelos preentrenados, que están listos para ser utilizados directamente en diversas tareas. Además, ofrece soporte para múltiples arquitecturas de modelos, entre las que se incluyen BERT, GPT y T5, lo que brinda flexibilidad para adaptarse a distintos enfoques y necesidades. La biblioteca también permite realizar ajustes finos o fine-tuning,

²<https://es.python.org/>

³<https://docs.aiohttp.org/en/stable/>

⁴<https://huggingface.co/>

lo que significa que es posible adaptar modelos generales a tareas específicas mediante un proceso de entrenamiento adicional con datos concretos. Otro aspecto destacable es su sistema de tokenización, que permite procesar texto de forma eficiente, dividiéndolo en unidades manejables por los modelos de lenguaje.

Complementando esta funcionalidad, Hugging Face proporciona un hub de modelos que actúa como un repositorio centralizado para el almacenamiento, publicación y distribución de modelos. Este espacio permite a los usuarios compartir sus modelos fácilmente con la comunidad, facilitando la colaboración y el acceso a soluciones desarrolladas por otros. Además, el hub incorpora un sistema de control de versiones, lo cual resulta fundamental para gestionar el ciclo de vida de los modelos, permitiendo seguir la evolución de cada versión con claridad. Todo esto se ve reforzado por una documentación completa, que incluye guías, ejemplos prácticos y explicaciones detalladas que ayudan tanto a principiantes como a desarrolladores avanzados a integrar y utilizar los modelos de manera efectiva

4.5 GitHub

GitHub⁵ es la herramienta de control de versiones más conocida en el mundo. Es también la más utilizada generalmente. Gracias a ella se puede realizar un seguimiento del desarrollo del código, y en casos necesarios volver a versiones anteriores o realizar bifurcaciones en la hora del desarrollo para poder realizarlo de manera concurrente.

Es fundamental a la hora de mantener el código, ya que permite fácil acceso, mantenimiento continuo y lo más importante, un sitio seguro donde estar almacenado.

4.6 Gemini

Gemini⁶ es un modelo de inteligencia artificial creado por Google DeepMind que entiende y genera texto de forma sorprendentemente natural[5]. Es como un “cerebro digital” capaz de leer, escribir y hasta analizar imágenes, lo que lo hace muy versátil para todo tipo de tareas relacionadas con el lenguaje. Gracias a su capacidad para razonar y adaptarse a diferentes temas, Gemini se ha convertido en una herramienta muy útil para quienes trabajan con procesamiento de lenguaje natural. En este proyecto, Gemini se ha usado como el motor que responde preguntas y luego compara sus respuestas con las correctas, evaluando qué tan bien lo hace con métricas como BertScore. Así, ha sido clave para medir de manera objetiva y automática el rendimiento de la IA en tareas de comprensión y generación de texto, facilitando el análisis y la mejora de los resultados de forma sencilla y eficiente.

4.7 BERTScore

BERTScore es una métrica de evaluación para los modelos, la cual se encarga de analizar el texto en busca de similitudes semánticas entre las palabras.

Gracias a este sistema, nos permite obtener una métrica más robusta y “humana”, ya que en comparación con otras métricas, estas lo que hacen son buscar coincidencias exactas de palabras o n-gramas, lo que las hacen más estrictas.

⁵<https://github.com/>

⁶<https://gemini.google.com/>

A continuación la explicación de cómo funciona exactamente BERTScore:

1. Cada palabra de los textos, tanto la hipótesis como la referencia, se convierten en vectores de representación (embedd) utilizando modelos de lenguaje preentrenados como por ejemplo BERT
2. Se calcula la similitud entre palabras. Por cada par de palabras entre la hipótesis y la referencia se calcula la similitud de coseno entre sus vectores con la siguiente fórmula donde x e y son los embeddings de las palabras:

$$\text{sim}(x, y) = \frac{x \cdot y}{\|x\| \cdot \|y\|}$$

3. Se emparejan las palabras mas similares de cada texto, pero hay que recalcar dos situaciones:
 - Por cada palabra de la hipótesis se busca su palabra mas similar en la referencia, y de esta manera se calcula la precisión
 - Por cada palabra de la referencia se busca su palabra mas similar en la hipótesis, y por esta parte se calcula el recall
4. Tras la obtención de la precisión y el recall, se calcula la métrica final, el F1-score, haciendo la média armónica de la precisión (P) y el recall (R):

$$F_1 = 2 \cdot \frac{P \cdot R}{P + R}$$

Tras todo este proceso, y con la obtención del F1-score, tendremos la capacidad de medir si dos textos son similares entre si o no.

4.8 SLM Y LLM

4.8.1. ¿Qué Són?

Como hemos mencionado previamente en la introducción del trabajo, los modelos de lenguaje (Language Models) representan una evolución fundamental en el campo de la inteligencia artificial, dividiéndose principalmente en dos categorías: los Small Language Models (SLM) y los Large Language Models (LLM). En el contexto de este proyecto, podemos observar una implementación práctica de ambos tipos de modelos y sus aplicaciones.

4.8.2. SLM

Los *Small Language Models* (SLM) son modelos de lenguaje más compactos y eficientes, diseñados para tareas específicas y con un menor consumo de recursos computacionales. En este proyecto, podemos ver la implementación de varios SLM como “distilgpt2” o “gpt-neo-125M”. Estos modelos son ideales para aplicaciones que requieren respuestas rápidas y eficientes, especialmente en entornos con recursos limitados. Los SLM se caracterizan por su capacidad de generar respuestas coherentes y contextualmente relevantes, aunque con un alcance más limitado en comparación con sus contrapartes más grandes.

4.8.3. LLM

Los *Large Language Models* (LLM), por otro lado, son modelos más complejos y potentes, capaces de manejar una amplia gama de tareas y generar respuestas más sofisticadas. En el proyecto, podemos observar la integración con modelos más grandes como "DialoGPT-medium". Estos modelos ofrecen una mayor capacidad de comprensión y generación de lenguaje, permitiendo respuestas más elaboradas y contextualmente ricas.

4.8.4. Integración en el proyecto

La implementación en este proyecto demuestra la importancia de la evaluación y comparación entre diferentes tipos de modelos. El sistema incluye una interfaz de pruebas que permite evaluar el rendimiento de los modelos locales y de modelos API, ambos pudiendo ser SLM o LLM. Esta comparación se realiza mediante métricas de evaluación como BERTScore, que mide la similitud semántica entre las respuestas generadas.

CAPÍTULO 5

Diseño de la solución

5.1 Introducción

El capítulo está centrado en cómo se ha diseñado la herramienta que se propone en el proyecto. Se explica el enfoque iterativo que ha seguido, empezando por una versión inicial centrada en la interacción con modelos locales y continuando con una versión más compleja con agentes que se comunican entre sí y permiten hacer pruebas automáticas. También se detalla cómo se combinan las métricas automáticas y las evaluaciones humanas para obtener una visión más completa del comportamiento de los modelos.

5.2 Diseño

Como solución, se va a realizar un diseño de una aplicación basado en la generación iterativa de prototipos, donde el objetivo principal es establecer un mecanismo que permita integrar modelos de lenguaje tanto grandes (LLM) como pequeños (SLM), abarcando tanto modelos locales como aquellos basados en servicios externos. Esta integración se realizará dentro del framework SPADE, permitiendo crear un ecosistema donde estos modelos puedan ser evaluados en diferentes contextos de interacción.

El propósito fundamental de este sistema es doble: por un lado, evaluar la interacción de estos modelos en diálogos directos con personas reales, y por otro, realizar evaluaciones automatizadas de diálogos entre diferentes modelos. Para lograr este objetivo, el desarrollo se abordará de manera incremental, comenzando con una primera aproximación que se centrará en la integración de modelos locales. En esta fase inicial, el sistema permitirá a los usuarios establecer diálogos directos con los modelos disponibles, facilitando una evaluación subjetiva de la satisfacción con las respuestas generadas. Esta primera iteración servirá como base para entender las necesidades reales de los usuarios y las limitaciones de los modelos en un contexto de interacción directa.

La siguiente iteración del prototipo expandirá significativamente las capacidades del sistema, introduciendo un mecanismo de pruebas automatizadas. En esta fase, los usuarios podrán configurar un número variable de agentes hijos según sus necesidades específicas. El sistema incorporará un agente padre que actuará como generador de preguntas, mientras que cada agente hijo podrá operar con un modelo de lenguaje diferente. Esta arquitectura flexible permitirá que tanto el generador de preguntas como los agentes hijos puedan utilizar modelos distintos, ya sean locales o basados en APIs externas.

Para evaluar el rendimiento de los modelos, el sistema implementará un conjunto de métricas tanto objetivas como subjetivas. En el caso de las métricas objetivas, se utilizarán dos enfoques complementarios: BERTScore, que proporcionará una evaluación

cuantitativa de la similitud semántica entre respuestas, y el modelo Gemini de Google, que ofrecerá una evaluación más sofisticada y contextual.

Esta combinación de métricas permitirá un análisis más completo y robusto, donde BERTScore aportará una medida objetiva de similitud, mientras que Gemini proporcionará una evaluación más profunda de aspectos como la coherencia, relevancia y calidad de las respuestas. Por otro lado, las métricas subjetivas se obtendrán a través de la interacción directa con usuarios, quienes podrán calificar aspectos como la satisfacción general, claridad y utilidad de las respuestas. El sistema almacenará y analizará estas métricas, generando visualizaciones que permitan comprender el rendimiento de los modelos en diferentes dimensiones.

Esta aproximación iterativa al desarrollo, combinada con un sistema de evaluación dual que integra tanto BERTScore como Gemini, no solo permite una implementación más controlada y evaluable, sino que también facilita la incorporación de mejoras basadas en datos concretos. El sistema resultante aspira a convertirse en una herramienta versátil para la evaluación y comparación de modelos de lenguaje, proporcionando insights valiosos tanto para investigadores como para desarrolladores que necesiten evaluar el rendimiento de diferentes modelos en contextos de diálogo.

5.3 Métricas

BERTScore es una métrica de evaluación de texto de última generación que aprovecha el poder de modelos de lenguaje preentrenados, como BERT, para medir cuán similares son, en términos de contexto, un texto generado y su correspondiente texto de referencia. A diferencia de otras métricas más tradicionales, BERTScore analiza las representaciones vectoriales (embeddings) de cada palabra en un espacio de alta dimensión, y a partir de ello calcula medidas como precisión, recall y F1. Gracias a este enfoque, es especialmente eficaz para captar el significado semántico y la coherencia del contenido, incluso cuando las palabras utilizadas no coinciden exactamente.

A diferencia de métricas tradicionales como BLEU o ROUGE, BERTScore captura mejor el significado semántico al utilizar representaciones contextuales de las palabras. El proceso consiste en generar embeddings para cada token en los textos de referencia y candidato, y luego calcular la similitud coseno entre estos embeddings. Esto permite una evaluación más precisa de la similitud semántica entre respuestas, especialmente útil para evaluar la calidad de las respuestas generadas por modelos de lenguaje.

Por otro lado, Gemini, desarrollado por Google, es un modelo de lenguaje avanzado que puede ser utilizado como evaluador de respuestas. A diferencia de BERTScore, que se centra en la similitud semántica, Gemini puede realizar una evaluación más holística y contextual. El modelo puede analizar aspectos como la coherencia, relevancia, claridad y calidad general de las respuestas, proporcionando una evaluación más completa y detallada. Gemini puede entender el contexto de la pregunta y evaluar si la respuesta es apropiada, precisa y útil, ofreciendo una perspectiva más humana en la evaluación de las respuestas generadas.

La combinación de ambas métricas proporciona una evaluación más robusta: mientras BERTScore ofrece una medida cuantitativa y objetiva de la similitud semántica, Gemini aporta una evaluación más cualitativa y contextual, similar a cómo un humano evaluaría las respuestas. Esta dualidad permite obtener una visión más completa del rendimiento de los modelos de lenguaje en diferentes aspectos de la generación de respuestas.

5.4 Soluciones

Como resultado del proceso de diseño y exploración de posibilidades técnicas, se han identificado y desarrollado dos soluciones que responden a diferentes niveles de complejidad y escenarios de uso. La primera de estas soluciones está enfocada en la interacción directa con modelos locales, facilitando la recolección de métricas subjetivas. La segunda, de carácter más avanzado, introduce una arquitectura distribuida de agentes para la evaluación automatizada de respuestas generadas por múltiples modelos, integrando tanto modelos locales como externos mediante APIs.

Ambas soluciones han sido concebidas dentro del mismo marco iterativo, y su desarrollo progresivo permite abordar distintos niveles de evaluación, desde lo más exploratorio y centrado en el usuario hasta lo más sistemático y escalable. Estas soluciones se describen con mayor detalle en los capítulos siguientes, donde se explican su implementación, arquitectura, funcionalidades clave y los resultados obtenidos durante su desarrollo y prueba.

CAPÍTULO 6

Desarrollo v1.0: diálogo modelo frente a humano en SPADE

6.1 Introducción

En este capítulo se expone la implementación de la primera versión funcional del sistema. El enfoque se centra en ofrecer una vía para que el usuario pueda interactuar directamente con los modelos de lenguaje mediante una interfaz web sencilla. Se detalla cómo se lleva a cabo la gestión de los modelos, el proceso de generación del diálogo y la recopilación tanto de métricas automáticas como del feedback proporcionado por los usuarios. El capítulo finaliza con una batería inicial de pruebas y el análisis de los resultados obtenidos.

6.2 Arquitectura del sistema

6.2.1. Diseño base

El sistema consiste en una arquitectura que combina las capacidades multiagente de SPADE junto a los modelos preentrenados. Dentro de cada agente tenemos toda la configuración necesaria para cargar modelos localmente descargándolos de la web HuggingFace, lo cual permite una búsqueda sencilla para elegir los modelos deseados.

6.2.2. Gestión de modelos

La gestión se consigue gracias a un sistema dinámico que permite la carga y descarga de modelos. Gracias a esto podemos ahorrar espacio y recursos ya que pueden haber modelos que utilicen un gran número de recursos y poder liberarlos nos ayuda a un mejor funcionamiento de la herramienta. También se mantiene un registro de los modelos que descarguemos, para así poder obtener una lista y ejecutar el modelo que deseemos en cualquier momento. Se incluye la libertad de poder cargar cualquier modelo en la herramienta gracias a HuggingFace, permitiendo así la escalabilidad del sistema. La gestión de los recursos se logra mediante la implementación de diccionarios especializados para almacenar tanto las instancias de los modelos como sus tokenizadores.

6.3 Proceso de integración

6.3.1. Inicialización y carga

La integración comienza en el momento en el que se inicializa el agente, donde se establece toda la infraestructura necesaria para manejar y gestionar todos los modelos. Gracias al sistema implementado, los modelos se cargan solo cuando están disponibles y quieren ser utilizados. En el caso de querer utilizar un modelo que no esté disponible, se revisa si está descargado localmente y si no lo está, entonces lo descarga. La descarga solo se realiza cuando es necesaria, una vez descargado un modelo no se va a volver a descargar ya que se quedará guardado en registro. De esta manera optimizamos recursos y tiempos de respuesta.

6.3.2. Generación de respuestas de ChatBot

El núcleo de la integración reside en el sistema de ChatBot, el cual nos proporciona respuestas a las preguntas que le hagamos. Este sistema tiene una plantilla de medidas que guían al modelo en la generación de respuestas. Las medidas permiten modificar el output por ejemplo cambiando o estableciendo un máximo de palabras, párrafos, o incluso permitiendo libre creatividad para el output. Gracias a esto podemos hacer pruebas e ir modificando estas medidas según los resultados obtenidos.

6.4 Sistema de evaluación

6.4.1. Métricas y análisis

Para los modelos locales hay dos maneras de evaluar las respuestas:

1. En primer lugar tenemos el feedback humano. Para esto se ha diseñado un ChatBot con la opción de poder hacer click en una cara sonriente o en una triste dependiendo de si la respuesta obtenida es satisfactoria o no para el humano (véase 6.1).

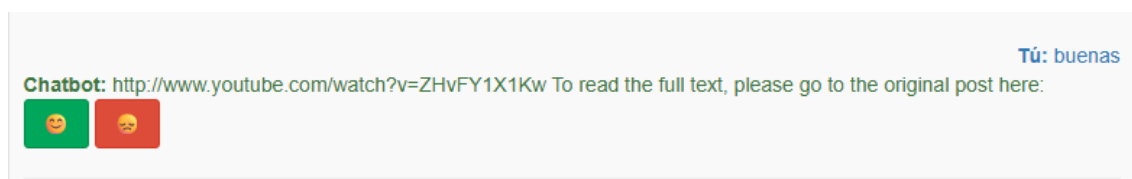


Figura 6.1: Muestra de ChatBot con botones para evaluación humana

2. En segundo lugar tenemos BERTscore, el cual consiste en una métrica de evaluación que funciona comparando la similitud semántica de dos textos, por lo que nos permite analizar el texto que enviamos al modelo y su respuesta, obteniendo un número entre 0 y 1. Cuanto más se acerque el número al 1, mejor y más relacionada

estará la respuesta con lo que hemos enviado. Hay que recalcar que nos devuelve tres métricas, la precisión, el recall, y el F1 que es la media de ambas. (véase 6.2).

Usuario: hola que tal

Chatbot: puera Reddit: @giant_u Twitter: <http://www.twitter.com/GiantFan> Facebook: <https://facebook>. Pocket: www.facebook

BERTScore: F1: 0.514, Precisión: 0.445, Recall: 0.608

Figura 6.2: Muestra de métricas con BERTScore

6.5 Gestión de recursos

Un aspecto crucial del proyecto es la gestión de recursos y la optimización del rendimiento. Los SLM, al ser más ligeros, permiten una implementación más eficiente en términos de memoria y procesamiento. El sistema implementa mecanismos de carga y descarga dinámica de modelos, permitiendo cambiar entre diferentes modelos según las necesidades específicas de cada tarea.

El sistema implementa estrategias de gestión de memoria que incluyen:

- Limpieza de caché de GPU
- Gestión eficiente de modelos en memoria
- Liberación proactiva de recursos

6.6 Interfaz de Usuario

6.6.1. Sistema de Interacción

Para toda la interacción con los agentes y los modelos, se dispone de una interfaz web que proporciona el propio sistema de SPADE, pero ampliada con todas las funciones para utilizar, evaluar y obtener todas las métricas de los modelos.

6.6.2. Selección de modelo

En primer lugar, necesitamos escoger un modelo a utilizar para poder utilizar todas las herramientas. Para esto se ofrece un desplegable con los modelos que el usuario ha seleccionado para utilizar (esto es completamente modificable y se pueden agregar/eliminar todos los modelos que sean necesarios) véase 6.3 y 6.4.

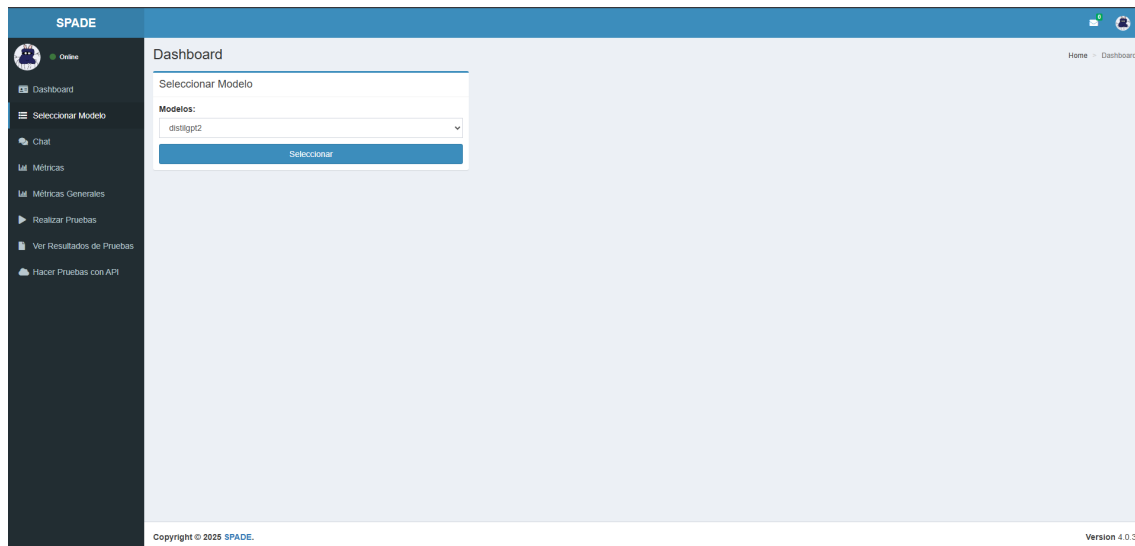


Figura 6.3: Muestra de interfaz web para seleccionar modelo

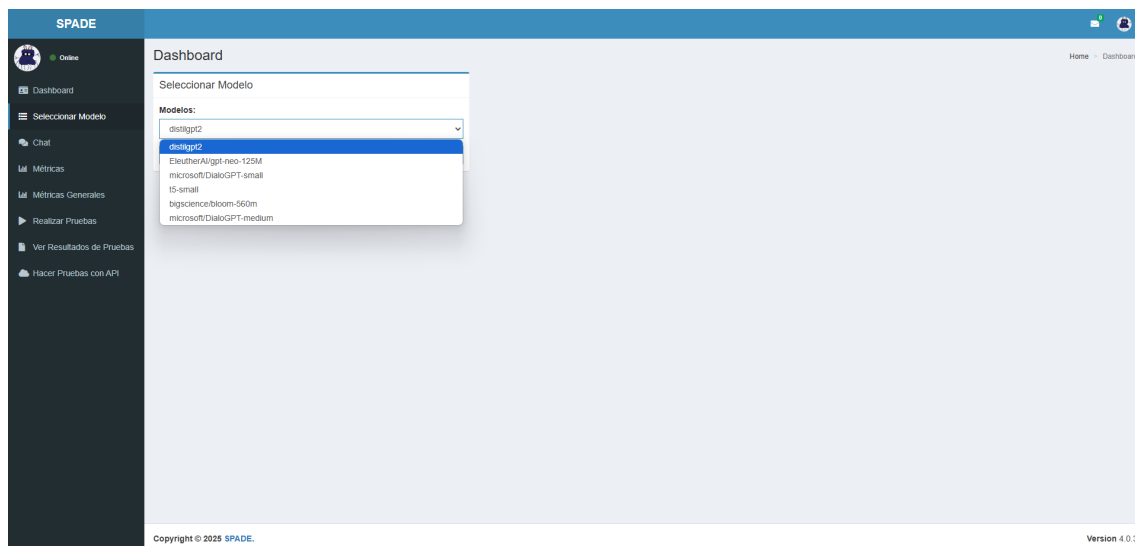


Figura 6.4: Muestra de interfaz web con desplegable para seleccionar modelo

6.6.3. Chat

En esta sección, podemos ver una interfaz muy clara y concisa, simplemente con un cuadro de texto grande el cual nos permitirá interactuar con el modelo que seleccionemos. En el caso de no haber seleccionado un modelo, nos avisará con un mensaje. En el caso de que estemos utilizando un modelo y queramos cambiar a otro, se podrá realizar desde el botón con el texto 'Cambiar modelo'.

Una vez comencemos la conversación con el modelo seleccionado, nos irá brindando las respuestas generadas por el mismo. Cada respuesta proporciona la posibilidad de dar feedback sobre ella, utilizando uno de los botones que aparecen justo debajo. En el caso de considerar que se trata de una respuesta coherente y satisfactoria podremos apretar en la cara feliz, en caso contrario la cara triste. Gracias a esto podemos obtener unas métricas humanas y conocer la satisfacción de la gente con cada modelo.

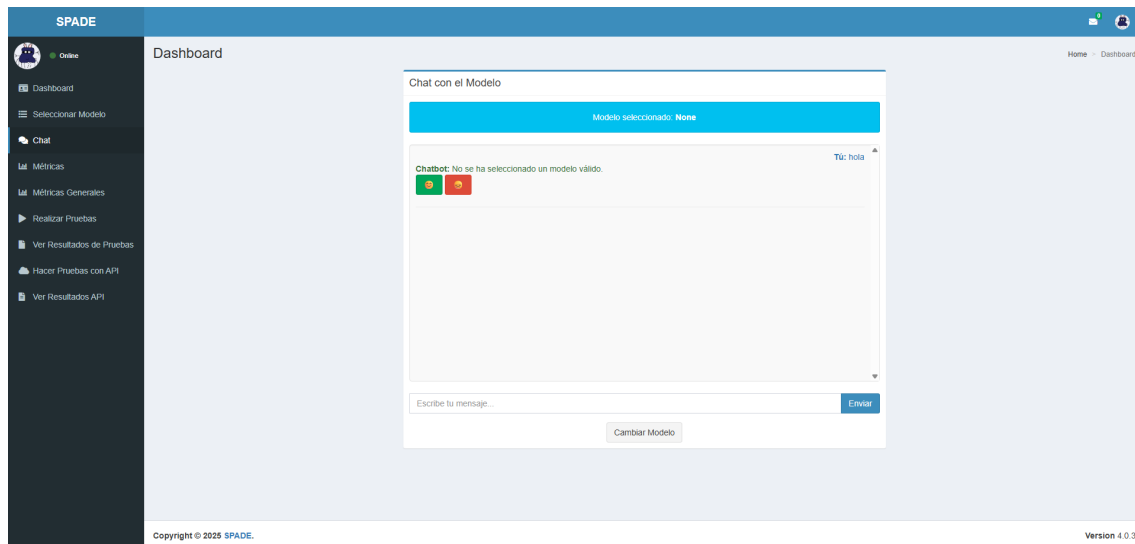


Figura 6.5: Muestra de interfaz web con chat sin seleccionar modelo

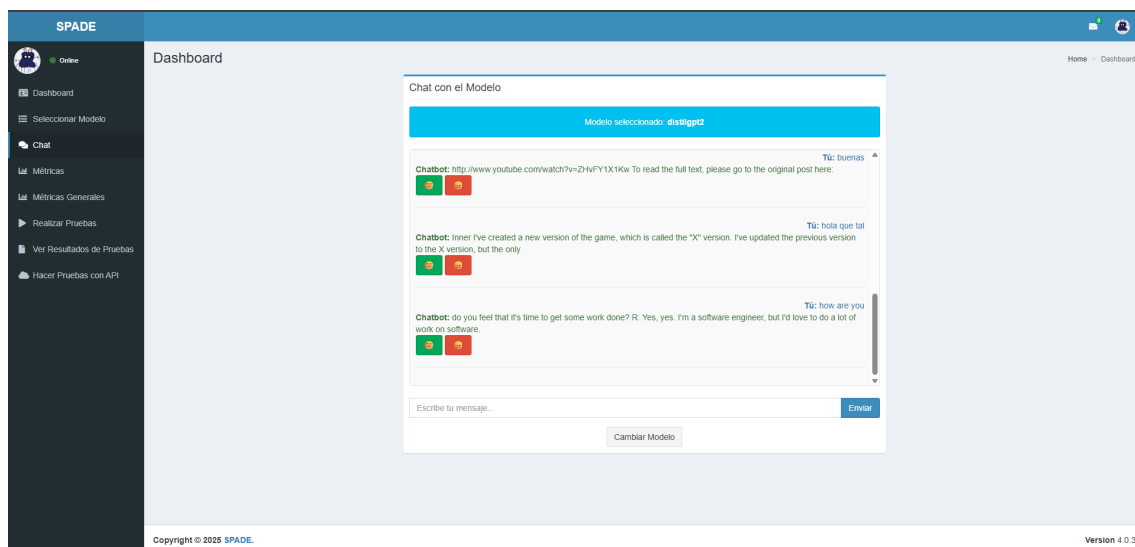


Figura 6.6: Muestra de interfaz web con chat y modelo seleccionado

6.6.4. Métricas Humanas y Scores

Como se ha mencionado previamente, en el Chat podemos dar feedback manual sobre las respuestas que nos da el modelo. En este apartado de la interfaz web podemos ver todas las estadísticas sobre un modelo concreto, tanto el feedback humano, como los scores generados por BERTscore y unas medias generales sobre la satisfacción e incluso alucinaciones.



Figura 6.7: Muestra de interfaz web con métricas humanas y porcentajes de BERTscore

6.6.5. Métricas Generales

En este apartado se proporcionan unas métricas más generales, además del tamaño ocupado por cada modelo ya que es algo muy relevante a la hora de tener en cuenta qué modelo nos conviene más. En concreto las métricas generales más relevantes son:

- Respuestas positivas: las respuestas positivas que indica el usuario
- Respuestas negativas: las respuestas negativas que indica el usuario
- Tamaño del modelo: tamaño del modelo calculado en MB.
- Alucinaciones: respuestas que parecen correctas pero no lo son.
- Media de BERTscore: la media aritmética de todos los BERTscore del modelo.

Gracias a esta vista más generalizada sobre cada modelo, podemos guiarnos sobre cuál puede ser más conveniente.

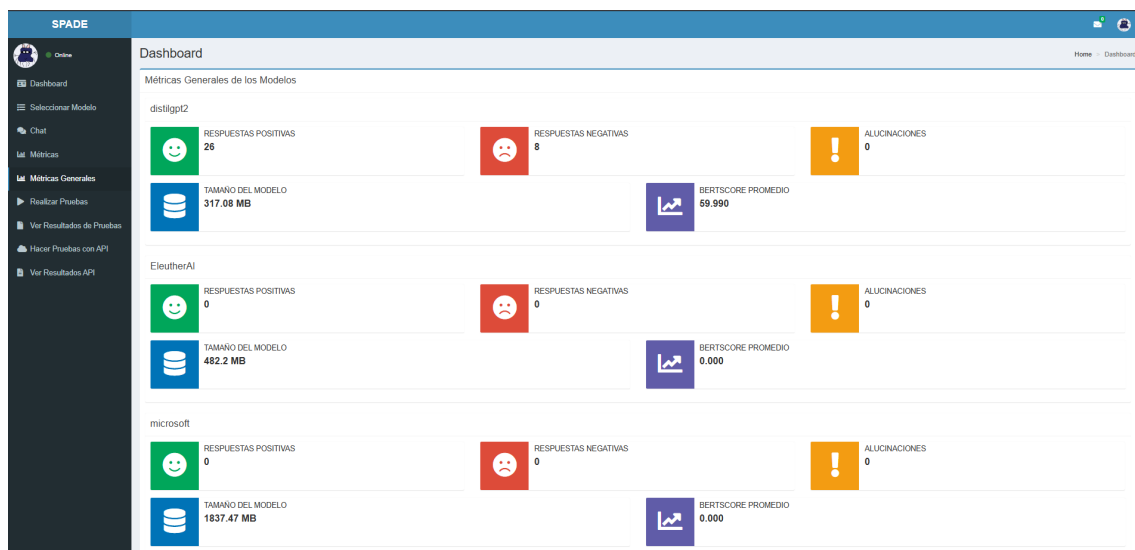


Figura 6.8: Muestra de interfaz web con métricas generales

6.6.6. Pruebas

Una de las partes más importantes a la hora de utilizar modelos SLM o LLM, es realizar pruebas sobre ellos. Gracias a las pruebas podremos ver cómo se desarrolla cada modelo en diferentes escenarios (responder preguntas, hacer operaciones matemáticas, resumir textos, etc) ya que cada modelo es mejor para ciertas situaciones. En este caso se ha desarrollado un entorno de pruebas automatizadas que se encarga de cargar un archivo con preguntas preseleccionadas, y la herramienta se encarga de enviárselas al modelo automáticamente. Al terminar se obtiene una pantalla como la que podemos observar en 6.9 donde ver los resultados de las pruebas, con la pregunta realizada, la respuesta esperada, y el score.

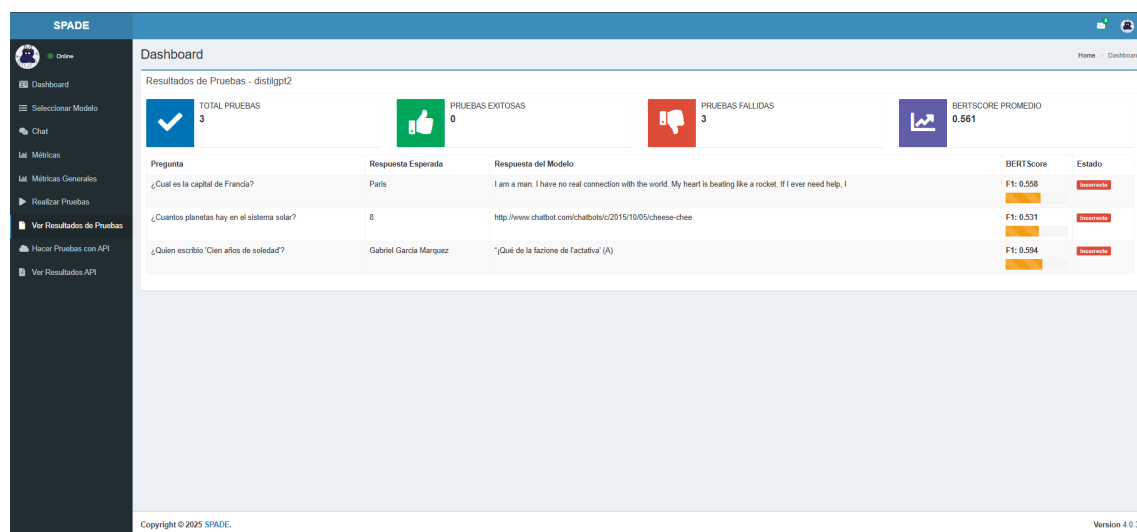


Figura 6.9: Muestra de interfaz web con resultados de pruebas locales

6.6.7. Pruebas con API

Además de las pruebas con ficheros locales, también se proporciona la opción de realizar las pruebas utilizando la API de un modelo preentrenado, para así evitar más cargas de modelos y poder obtener resultados provenientes de LLM's muy grandes como por ejemplo GEMINI. Gracias a esto podemos comparar resultados y sacar más conclusiones sobre cada modelo. Es importante destacar que las preguntas que se realizan son aleatorias.

En la figura 6.10 podemos observar el formulario a rellenar para hacer las pruebas. Es necesario proporcionar una clave API (en este caso de GEMINI) y el número de preguntas deseadas a realizar.

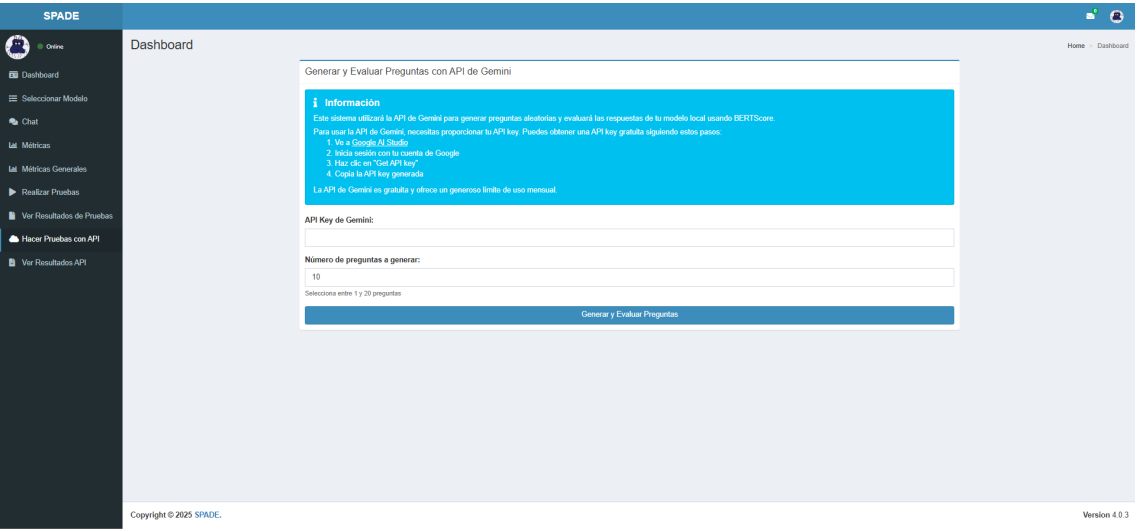


Figura 6.10: Muestra de interfaz web con formulario para pruebas API

Una vez rellenado y enviado el formulario, se procesarán todas las preguntas que podremos observar durante la ejecución en logs a tiempo real y al finalizar la ejecución.

En los resultados se muestra la pregunta que se envía al modelo local, la respuesta del modelo local y la respuesta del modelo que usamos con la API. De esta manera podemos comparar ambos resultados y ver como de coherentes son.

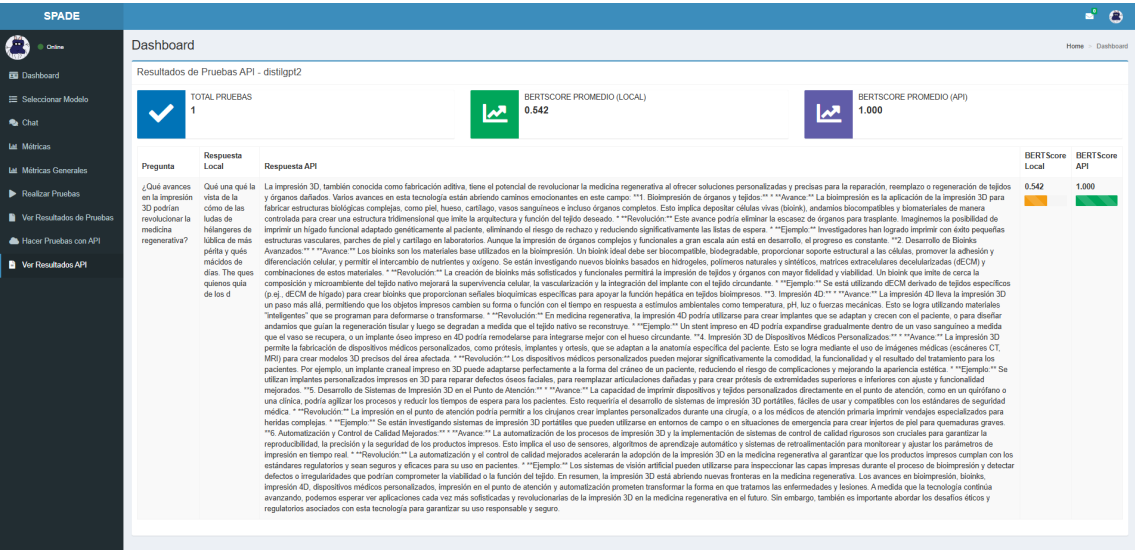


Figura 6.11: Muestra de interfaz web con resultados de pruebas API

6.7 Pruebas

Para las pruebas de esta aproximación se ha utilizado una simple batería de 50 preguntas con temas variados a 3 modelos distintos. Las pruebas en esta aproximación no son tan exhaustivas ya que esta primera simplemente está dedicada a poder comunicarse con un modelo y poder aportar el feedback subjetivo del usuario junto a poder hacer unas pruebas sencillas.

La batería de preguntas es la siguiente (tabla 6.1):

Nº	Pregunta	Respuesta esperada
1	¿Cuál es la capital de Francia?	París
2	¿Cuántos planetas hay en el sistema solar?	8
3	¿Quién escribió 'Cien años de soledad'?	Gabriel García Márquez
4	¿En qué año llegó el hombre a la Luna?	1969
5	¿Cuál es el océano más grande del mundo?	Océano Pacífico
6	¿Qué país tiene la mayor población del mundo?	China
7	¿Cuál es el elemento químico con el símbolo O?	Oxígeno
8	¿Quién pintó la Mona Lisa?	Leonardo da Vinci
9	¿Cuál es el río más largo del mundo?	Amazonas
10	¿Cuál es la moneda de Japón?	Yen
11	¿Cuántos continentes hay?	7
12	¿Cuál es el país más grande del mundo?	Rusia
13	¿Qué instrumento mide la temperatura?	Termómetro
14	¿En qué continente está Egipto?	África
15	¿Quién descubrió América?	Cristóbal Colón
16	¿Cuál es la capital de Argentina?	Buenos Aires
17	¿Qué gas necesitan las plantas para la fotosíntesis?	Dióxido de carbono
18	¿Cuál es el país con más hispanohablantes?	México
19	¿Qué planeta es conocido como el planeta rojo?	Marte
20	¿Cuántos huesos tiene el cuerpo humano adulto?	206
21	¿Quién fue Albert Einstein?	Un físico
22	¿Cuál es el idioma más hablado en el mundo?	Chino mandarín
23	¿En qué país se encuentra la Torre Eiffel?	Francia
24	¿Qué órgano del cuerpo humano bombea la sangre?	Corazón
25	¿Quién escribió 'Don Quijote de la Mancha'?	Miguel de Cervantes
26	¿Cuál es la fórmula química del agua?	H ₂ O
27	¿Cuál es el animal terrestre más rápido?	Guepardo
28	¿En qué país está el desierto del Sahara?	Varios países del norte de África
29	¿Qué número romano representa el 10?	X
30	¿Cuál es la capital de España?	Madrid
31	¿Qué año comenzó la Segunda Guerra Mundial?	1939
32	¿Qué instrumento tiene teclas, cuerdas y martillos?	Piano
33	¿Quién escribió 'La Odisea'?	Homero
34	¿Qué metal es líquido a temperatura ambiente?	Mercurio
35	¿Cuál es la capital de México?	Ciudad de México
36	¿Cuál es el animal más grande del mundo?	Ballena azul
37	¿Qué planeta está más cerca del sol?	Mercurio
38	¿Qué país ganó la Copa Mundial de la FIFA 2018?	Francia
39	¿Cuántos colores tiene el arcoíris?	7
40	¿Cuál es el símbolo químico del oro?	Au
41	¿Qué lengua hablaban los romanos?	Latín
42	¿Quién pintó 'La última cena'?	Leonardo da Vinci
43	¿Qué órgano produce la insulina?	Páncreas
44	¿Cuál es la capital de Colombia?	Bogotá

Nº	Pregunta	Respuesta esperada
45	¿Qué científico propuso la teoría de la evolución?	Charles Darwin
46	¿En qué país está la Gran Muralla?	China
47	¿Cuál es el idioma oficial de Brasil?	Portugués
48	¿Qué planeta tiene un sistema de anillos?	Saturno
49	¿Qué continente no tiene desiertos?	Europa
50	¿Quién escribió 'Romeo y Julieta'?	William Shakespeare

Tabla 6.1: Preguntas de las pruebas de la primera aproximación

De estas preguntas hemos obtenido los scores por cada pregunta y el promedio. En la siguiente tabla podemos observar el score por pregunta:

Nº	Distilgpt2	Gemini	DialoGPT-small
1	56.3 %	61.2 %	70.9 %
2	55.5 %	58.9 %	68.2 %
3	57.9 %	57.6 %	60.9 %
4	54.9 %	51.9 %	63.8 %
5	59.5 %	53.5 %	63.8 %
6	59.0 %	61.3 %	76.1 %
7	58.3 %	58.5 %	71.0 %
8	57.3 %	54.7 %	69.9 %
9	58.2 %	58.2 %	69.1 %
10	50.8 %	54.2 %	55.7 %
11	54.4 %	59.2 %	63.5 %
12	54.8 %	56.3 %	61.6 %
13	56.2 %	60.5 %	74.0 %
14	59.2 %	60.2 %	61.6 %
15	47.3 %	55.8 %	52.8 %
16	58.0 %	59.5 %	0.0 %
17	50.0 %	59.2 %	51.7 %
18	53.6 %	56.3 %	65.3 %
19	54.4 %	59.9 %	66.4 %
20	56.9 %	58.1 %	56.4 %
21	61.2 %	61.1 %	65.4 %
22	55.7 %	61.6 %	70.8 %
23	55.1 %	53.2 %	53.0 %
24	60.9 %	59.0 %	66.3 %
25	59.2 %	59.9 %	63.9 %
26	53.7 %	59.3 %	64.3 %
27	60.9 %	57.1 %	58.3 %
28	61.4 %	60.6 %	68.9 %
29	55.0 %	55.5 %	58.7 %
30	52.7 %	58.8 %	69.2 %
31	59.4 %	61.4 %	65.5 %
32	58.2 %	56.8 %	55.7 %
33	65.2 %	65.4 %	69.7 %
34	59.2 %	64.2 %	68.2 %

Nº	Distilgpt2	Gemini	DialoGPT-small
35	53.5 %	57.7 %	63.7 %
36	64.2 %	64.7 %	66.7 %
37	64.0 %	64.3 %	64.8 %
38	54.2 %	57.2 %	52.0 %
39	56.1 %	57.5 %	66.4 %
40	55.6 %	57.9 %	59.8 %
41	52.1 %	50.7 %	56.0 %
42	59.5 %	61.1 %	68.9 %
43	60.0 %	54.9 %	65.0 %
44	58.5 %	56.6 %	68.3 %
45	58.5 %	57.5 %	66.4 %
46	65.3 %	58.1 %	65.1 %
47	61.0 %	62.0 %	73.5 %
48	56.1 %	59.8 %	62.6 %
49	61.9 %	58.2 %	75.6 %
50	61.7 %	53.6 %	60.4 %

Tabla 6.2: Resultados pruebas primera aproximación

A continuación los resultados promedios por modelo (tabla 6.3):

Tabla 6.3: Promedio de scores por modelo (Primera aproximación)

Modelo	Promedio (%)
Distilgpt2	57.16
Gemini	58.41
DialoGPT-small	63.64

Cada modelo respondió a las mismas preguntas, y se compararon las respuestas utilizando métricas automáticas, especialmente BERTScore. Se presentan los resultados individuales por pregunta, así como los valores promedio de precisión, lo que permite observar diferencias significativas de rendimiento entre los modelos. Aunque esta fase no tiene un enfoque exhaustivo, sirve como punto de partida para futuras evaluaciones más completas.

6.8 Puntos a destacar en la aproximación 1

Como conclusión de esta aproximación, se puede afirmar que el sistema desarrollado cumple con el objetivo principal de proporcionar una herramienta versátil e interactiva para la evaluación de modelos de lenguaje. Esta herramienta permite al usuario establecer un canal de comunicación directo con un modelo, el cual puede ser uno predefinido o un modelo cargado manualmente por el propio usuario. Esta funcionalidad amplía significativamente las posibilidades de personalización y adaptabilidad del sistema, permitiendo evaluar no solo modelos generales, sino también aquellos diseñados o afinados para tareas específicas.

A partir de esta interacción, el sistema permite recoger información en forma de feedback subjetivo por parte del usuario. Esta retroalimentación es clave para valorar aspectos como la coherencia, naturalidad o adecuación de las respuestas generadas por el modelo, que a menudo son difíciles de capturar únicamente con métricas automáticas. De esta forma, se integra un componente humano en el proceso de evaluación, lo que proporciona una perspectiva más completa y personal.

Junto a este componente subjetivo, el sistema incorpora una capa de análisis cuantitativo mediante el cálculo de métricas automáticas. Por cada mensaje intercambiado, se generan una serie de scores utilizando herramientas como BERTscore o, alternativamente, el modelo Gemini, en función de la preferencia del usuario. Estas métricas permiten obtener valores objetivos como precisión, recall y relevancia, que ayudan a medir la calidad de las respuestas del modelo en comparación con una referencia o con los criterios establecidos. El hecho de poder elegir entre BERTscore o Gemini aporta flexibilidad y adaptabilidad a distintos enfoques de evaluación.

Además, el sistema ofrece dos modalidades complementarias para la realización de pruebas automatizadas. La primera consiste en el uso de un archivo predefinido que contiene una batería de preguntas. Estas preguntas se envían automáticamente al modelo seleccionado, lo que permite evaluar su comportamiento de forma sistemática y sin intervención manual, facilitando la comparación entre modelos o versiones.

La segunda modalidad hace uso de APIs externas, como la API de Gemini, que permite generar dinámicamente un conjunto de preguntas a partir de un tópico introducido por el usuario. Esta opción añade una capa de automatización inteligente al proceso, ya que las preguntas se adaptan al tema deseado y permiten evaluar la capacidad del modelo para responder en contextos específicos. Las preguntas generadas se envían directamente al modelo para su procesamiento, y posteriormente se evalúan sus respuestas utilizando las métricas mencionadas anteriormente.

En conjunto, el sistema no solo permite interactuar con modelos de lenguaje de forma flexible y personalizada, sino que también proporciona un entorno completo para su evaluación, combinando tanto métricas objetivas como valoración humana. Esto lo convierte en una herramienta útil tanto para la investigación como para el desarrollo y mejora de modelos de lenguaje.

CAPÍTULO 7

Desarrollo v2.0: dialogo entre modelos en SPADE

7.1 Introducción

A raíz de la primera versión de la herramienta, se quería conseguir llegar a realizar todas las pruebas de manera distribuida, pudiendo así probar de manera más extensa los modelos y poder realizarlo concurrentemente en varios agentes. Es por eso que se ha desarrollado una segunda versión con estas capacidades, teniendo así una versión para cada contexto.

7.2 Funcionamiento

El funcionamiento de esta versión es prácticamente igual a la anterior, es decir, todas las funcionalidades existentes en la primera versión también funcionan en esta segunda. La diferencia se encuentra en la función extra multiagentes. Esta nueva función dota a la herramienta de la posibilidad de lanzar agentes extra, asignarles el modelo deseado y lanzarles las pruebas.

7.3 Estructura multiagente

La estructura multiagente es posible gracias a todo el sistema que lleva incorporado SPADE. Tenemos la posibilidad de crear tantos agentes como queramos, y, entre ellos existe la posibilidad de comunicarse mediante mensajes. El sistema de mensajería es gracias a un servidor XMPP al cual se conectan todos los agentes a la hora de lanzarlos. También hay que recalcar que para poder entablar comunicación entre ellos deben 'conocerse', es decir, tienen que compartirse sus ID para poder ser 'contactos'.

En este caso, al lanzar los agentes desde un agente 'padre', el 'padre' ya tiene los ID de los 'hijos' (en este caso son los agentes lanzados por el padre), por lo que tiene la capacidad de enviarles mensajes al conocer sus ID ya que los ha creado él.

7.4 Envío de mensajes y procesamiento de pruebas

El envío de mensajes y el procesamiento de las respuestas en la plataforma sigue un flujo sencillo e intuitivo. El usuario a través de la interfaz web selecciona el modelo

o agente al que desea dirigir la consulta. Una vez enviado el mensaje, este es recibido por el sistema, que lo reenvía al modelo de lenguaje seleccionado para que genere una respuesta. El modelo procesa la información y devuelve una respuesta, que es presentada al usuario en la misma interfaz de manera clara y estructurada. Además, tras recibir la respuesta, el sistema puede aplicar automáticamente métodos de evaluación para valorar la calidad o relevancia de la respuesta generada, mostrando estos resultados junto al mensaje. De este modo, el usuario no solo obtiene una contestación a su consulta, sino que también puede visualizar métricas que ayudan a analizar el rendimiento del modelo o agente utilizado, todo ello de forma transparente y accesible.

7.5 Interfaz

En esta versión solo se mostraran las interfaces agregadas para esta, ya que todo lo demas es exactamente lo mismo que en la versión uno.

7.5.1. Cambios en mostrar resultados

Se ha cambiado la interfaz de visualizar los resultados, aportando una apariencia mas simple y dando la posibilidad de establecer unos marcos en porcentajes. sobre los cuales podemos establecer los porcentajes que creemos que son una respuesta válida, una respuesta incorrecta o una respuesta buena. Podemos observar el resultado de esta interfaz en la figura 7.1.

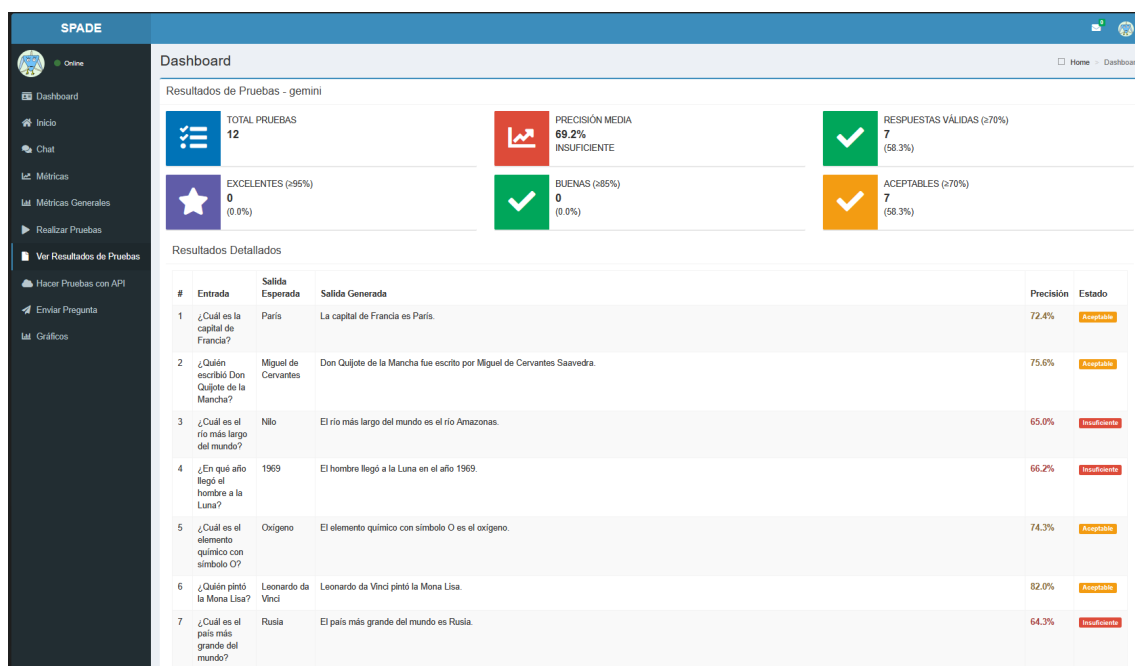


Figura 7.1: Muestra de interfaz web con resultados de pruebas

7.5.2. Mejoras en pruebas locales

Ahora el sistema de pruebas locales ofrece una experiencia mucho más flexible y dinámica, ya que permite modificar el fichero de preguntas predefinidas justo antes de lanzar las pruebas. Aunque este fichero ya formaba parte de la herramienta, la nueva interfaz facilita que cualquier usuario pueda añadir o eliminar preguntas de manera sencilla y rápida, adaptando así el conjunto de cuestiones a los objetivos concretos de cada

evaluación. Esta mejora no solo hace el proceso más cómodo, sino que también permite personalizar las pruebas de forma intuitiva y eficiente. Podemos observar este nuevo sistema en la figura 7.2

The screenshot displays the SPADE web interface. On the left is a dark sidebar with navigation links: Dashboard, Inicio, Chat, Métricas, Métricas Generales, Realizar Pruebas, Ver Resultados de Pruebas, Hacer Pruebas con API, Enviar Pregunta, and Gráficos. The main content area is titled 'Dashboard' and 'Editar Casos de Prueba'. It features a section 'Casos de Prueba para gemini' with a dropdown for 'Método de Evaluación' set to 'BERTScore - Evaluación por similitud semántica'. Below this, there are two test cases. 'Caso de Prueba #1' has a question '¿Cuál es la capital de Francia?' and an expected answer 'Paris'. 'Caso de Prueba #2' has a question '¿Cuántos planetas hay en el sistema solar?' and an expected answer '8'. At the bottom of the form are buttons for 'Guardar Cambios' and 'Ejecutar Pruebas'. The footer includes 'Copyright © 2025 SPADE.' and 'Version 4.0.3'.

Figura 7.2: Muestra de interfaz web con formulario para modificar casos de prueba

7.5.3. Lanzar agentes y preguntas

Además de la nueva implementación en esta versión, hablaremos sobre la interfaz interactiva para la configuración del modelo. Esta interfaz consiste en varios formularios, los cuales se deben rellenar para poder realizar el "lanzamiento" de los agentes hijos, la selección del modelo e incluso el envío de las preguntas a realizar. Entonces, el formulario nos permite añadir o eliminar los agentes hijos que queremos lanzar y seleccionar el modelo para cada hijo.

Podemos observar en la figura 7.3 los primeros formularios a rellenar.

A parte de este formulario tenemos un apartado con información en la parte inferior de la página.

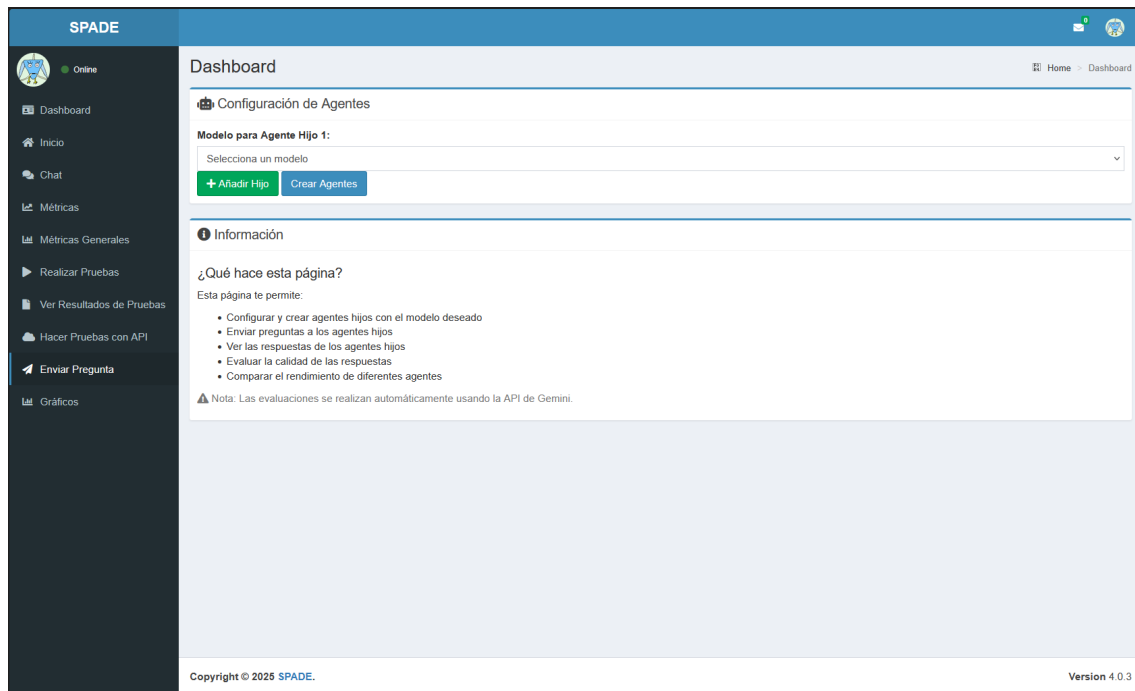


Figura 7.3: Muestra de interfaz web con formulario para lanzar agentes

Hay que destacar que los modelos a escoger serán los mismos que se muestran a la hora de querer hacer las pruebas locales, es decir, los que se hayan agregado al listado manualmente. Esto se puede observar en la figura 7.4

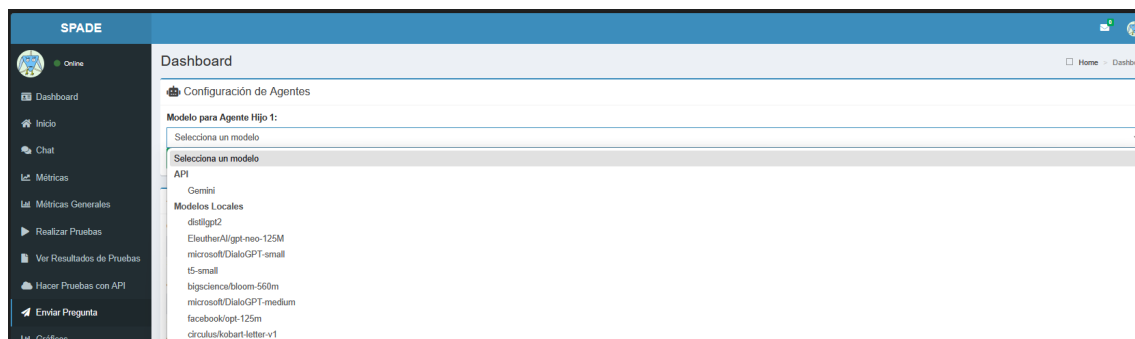


Figura 7.4: Muestra de interfaz web con formulario para lanzar agentes con listado de modelos

Trás rellenar ese formulario y hacer click en el botón de 'Iniciar agentes', podremos ver en la terminal a tiempo real como se va generando el nuevo agente (figura 7.5).

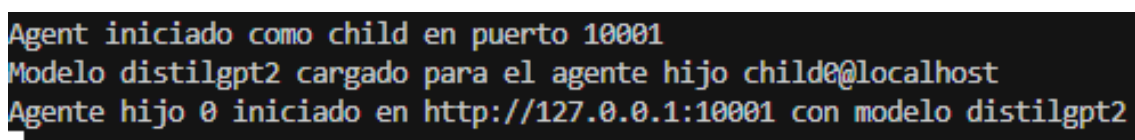


Figura 7.5: Muestra de interfaz web con formulario para lanzar agentes con listado de modelos

En el caso de seleccionar un modelo API, será necesario introducir la API key correspondiente (7.6).

Figura 7.6: Muestra de interfaz web con formulario para lanzar agentes con modelo API

Una vez finalizado este proceso, se nos actualizará la página con el siguiente formulario a rellenar, donde deberemos indicar los siguientes parámetros: (figura 7.7).

- Número de preguntas a enviar a cada agente hijo
- Modelo encargado de generar las preguntas
- Tópico de las preguntas
- Método de evaluación de las preguntas (BERTscore o Gemini)

Figura 7.7: Muestra de interfaz web con formulario para elegir número de preguntas

Igual que anteriormente, cuando lancemos las preguntas, tendremos feedback a tiempo real por la terminal, pudiendo observar como se va realizando el envio de mensajes y el procesamiento de las respuestas (figura 7.8). Hay que recalcar que a la hora de escoger el modelo para generar las preguntas, tendremos 2 opciones:

- Modelo local previamente seleccionado
- Google Gemini

El modelo local que se utilizará será el que habremos seleccionado en la pantalla de selección de agente, y por lo tanto si se desease utilizar otro modelo, deberíamos volver a esa pantalla y seleccionar otro modelo.

```
Mensaje enviado a childe@localhost: ¿Cómo afectaría el descubrimiento de vida extraterrestre a las religiones del mundo?
Pregunta enviada a childe@localhost: ¿Cómo afectaría el descubrimiento de vida extraterrestre a las religiones del mundo?
Mensaje enviado a parent@localhost: Cáomo Bautista se contrarrio una siciónero oporta su trasidad comodo hacerte algunos con los cuales para que teneran empresado e p
Respuesta enviada al padre: Cáomo Bautista se contrarrio una siciónero oporta su trasidad comodo hacerte algunos con los cuales para que teneran empresado e p
```

Figura 7.8: Muestra de terminal con informacion sobre el envio de preguntas

Finalmente, tras realizarse todos los envíos, lo último que quedaría sería la evaluación de las respuestas y mostrarlas en la página. Esto se puede observar como se realiza en la terminal (figura 7.9) y finalmente el resultado en la página (figura 7.10).

```
**En resumen, el teorema de Noether es un principio fundamental que vincula las simetrías continuas de un sistema físico con las leyes de conservación, proporcionando una herramienta poderosa para comprender y construir teorías en física teórica.** Permite deducir las cantidades físicas que se conservan en un sistema a partir de sus simetrías, lo que simplifica el análisis y la predicción del comportamiento del sistema.
Respuesta evaluada de childe@localhost usando método gemini: {'relevance': 1.0, 'accuracy': 1.0, 'completeness': 0.9, 'clarity': 0.95, 'overall_score': 0.96, 'method': 'gemini'}
```

Figura 7.9: Muestra de terminal con información sobre la evaluación de la pregunta

?

Pregunta

¿Cómo influye el contexto sociocultural en la evolución del lenguaje?

↩

Respuesta

El contexto sociocultural ejerce una influencia profunda y multifacética en la evolución del lenguaje. A continuación, se describen algunas de las principales formas en que esto ocurre: **1. Necesidades Comunicativas:** Las nuevas tecnologías y descubrimientos científicos requieren la creación de nuevos términos y expresiones para describir estos conceptos y facilitar su comunicación. Por ejemplo, palabras como "emoji", "streaming", "influencer" o "blockchain" han surgido recientemente para adaptarse a estas necesidades. **2. Cambios sociales:** La evolución de las normas sociales, las estructuras familiares, las relaciones laborales, etc., también impulsa la evolución del lenguaje. Por ejemplo, el desarrollo del lenguaje inclusivo es una respuesta a las demandas de igualdad de género y la necesidad de visibilizar a grupos minoritarios. **3. Contextos específicos:** Cada grupo social o profesional desarrolla un léxico y jerga particular adaptada a sus actividades y preocupaciones. Por ejemplo, el lenguaje utilizado en medicina, informática, o en el mundo del deporte tiene características y términos propios. **4. Identidad y Pertenencia:** El lenguaje como marcador de identidad. El lenguaje que utilizamos nos ayuda a definirnos como parte de un grupo social, regional o generacional. A través del acento, el dialecto, la jerga y las expresiones idiomáticas, mostramos nuestra pertenencia a una comunidad. **5. Mantenimiento y revitalización de lenguas minoritarias:** En contextos donde una lengua minoritaria está amenazada, el esfuerzo consciente por mantenerla y revitalizarla es una forma de afirmar la identidad cultural de un grupo. **6. Cambios en las actitudes lingüísticas:** La valoración social de determinadas formas de hablar influye en su adopción o rechazo. Por ejemplo, en algunos contextos, hablar una lengua extranjera con fluidez puede ser un signo de estatus, mientras que en otros, el uso de un dialecto local puede ser un símbolo de orgullo. **7. Contacto Lingüístico:** El contacto entre diferentes lenguas conlleva la adopción de palabras y expresiones de una lengua a otra (préstamos). Esto es especialmente común en áreas donde hay un contacto cultural intenso o donde una lengua ejerce una influencia dominante. Por ejemplo, el inglés ha influido en muchas lenguas, y a su vez, ha adoptado palabras de otros idiomas. **8. Calcos semánticos:** En lugar de adoptar una palabra directamente, se puede traducir su significado y crear una nueva palabra en la lengua receptora. **9. Pidgin y creole:** En situaciones de contacto lingüístico extremo, como en contextos coloniales o comerciales, pueden surgir pidgins (lenguas simplificadas de comunicación) y creoles (lenguas que se desarrollan a partir de un pidgin y se convierten en la lengua materna de una comunidad). **10. Ideología y Poder:** El lenguaje como herramienta de control. El lenguaje puede ser utilizado para legitimar el poder, promover ideologías y manipular la opinión pública. El uso de eufemismos, propaganda y discursos de odio son ejemplos de cómo el lenguaje puede ser utilizado para fines políticos o sociales. **11. Influencia de los medios de comunicación:** Los medios de comunicación, como la televisión, la radio, Internet y las redes sociales, tienen un gran impacto en la difusión de nuevas palabras, expresiones y formas de hablar. También pueden influir en la percepción social de diferentes variedades lingüísticas. **12. Estándares lingüísticos:** La imposición de una variedad lingüística estándar como la "correcta" puede marginar a aquellos que hablan otras variedades, lo que tiene implicaciones sociales y económicas. **13. Transmisión Cultural:** El lenguaje es el vehículo principal para transmitir historias, mitos, leyendas y conocimientos culturales de generación en generación. **14. Educación:** La educación formal juega un papel fundamental en la transmisión de la lengua estándar y en la formación de la conciencia lingüística. **15. Creencias y valores:** El lenguaje refleja y refuerza las creencias, los valores y las normas sociales de una cultura. Por ejemplo, las metáforas que utilizamos para describir el tiempo, el espacio o las emociones reflejan nuestras concepciones culturales del mundo. En resumen, el contexto sociocultural es un motor fundamental de la evolución del lenguaje. Las necesidades comunicativas, la identidad, el contacto lingüístico, la ideología y la transmisión cultural interactúan de forma compleja para dar forma a la forma en que hablamos y cómo cambian las lenguas a lo largo del tiempo. Comprender esta relación es crucial para entender la diversidad lingüística y cultural de la humanidad.

★ Evaluación

Relevancia: 1.00

Precisión: 1.00

Compleitud: 0.90

Claridad: 1.00

Puntuación General: 0.97

Método: GEMINI

👤 Agente: childe@localhost

Figura 7.10: Muestra respuesta de modelo en interfaz web

7.5.4. Gráficos

La interfaz web de los gráficos (figura 7.11) en la aplicación permite visualizar de manera clara los resultados de las evaluaciones realizadas por los distintos modelos de lenguaje. A través de representaciones visuales como gráficos de barras o líneas, el usuario puede comparar fácilmente métricas como precisión, relevancia o claridad entre diferentes respuestas o modelos, facilitando así la interpretación de los datos y la toma de decisiones. Además, la interactividad de estos gráficos permite explorar los resultados en detalle, haciendo que el análisis del rendimiento de los modelos sea mucho más intuitivo

y accesible que si solo se mostraran los valores numéricos. En conjunto, la funcionalidad gráfica transforma la experiencia de evaluación en algo visual, comprensible y dinámico.

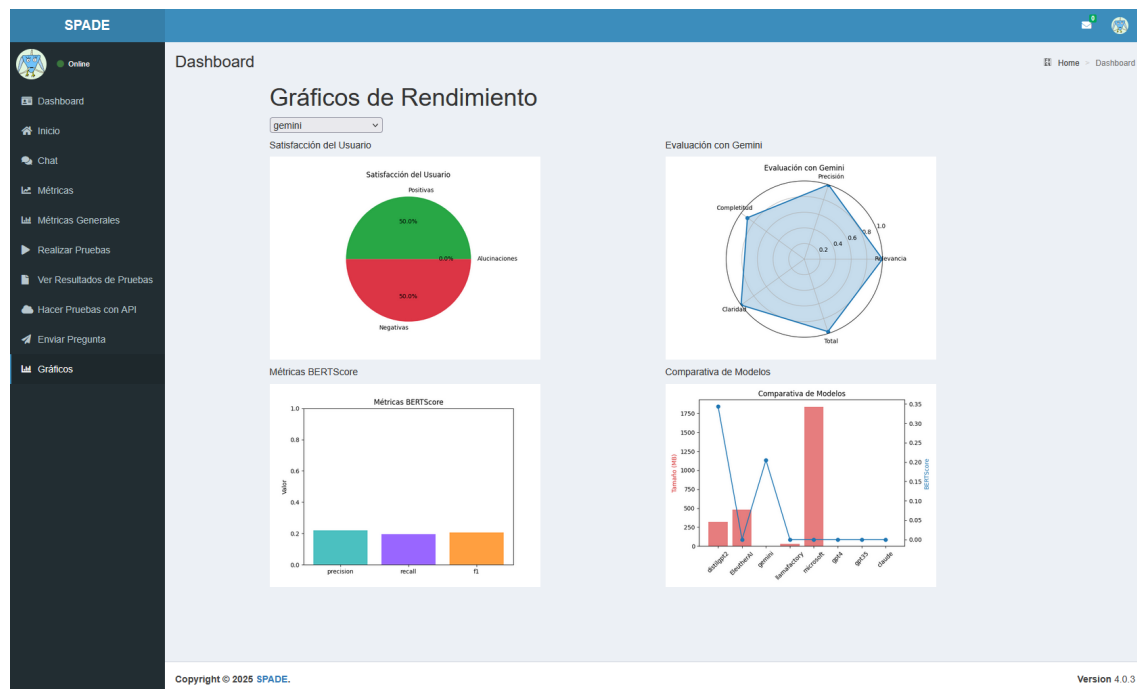


Figura 7.11: Muestra de la interfaz web con los gráficos

7.6 Pruebas

Se han realizado una serie de pruebas para demostrar el funcionamiento de la herramienta. Estas pruebas consisten en lo siguiente:

- Batería de pruebas predefinidas aplicadas a cada modelo
- Generación aleatoria de preguntas, generadas por diferentes modelos y diferentes tópicos.

En este caso solo se han utilizado 3 modelos, distilgpt2, gemini y gpt-neo-125m. Se han seleccionado estos 3 modelos para poder tener 1 ejemplo de API como es gemini, y 2 ejemplos locales.

Las pruebas están limitadas a 15 preguntas por ejecución, esto es debido a que el plan gratuito de la API de gemini tiene un límite de peticiones por lo que es un condicionante. Evidentemente con otro plan de gemini de pago se pueden realizar muchas más peticiones.

7.6.1. Batería de pruebas predefinidas

Gracias a la opción de tener un archivo con pruebas predefinidas modificable, se ha establecido una serie de 15 preguntas predefinidas sobre diferentes ámbitos para lanzar a cada modelo. Las preguntas en cuestión se pueden observar en la Tabla 7.1:

Tabla 7.1: Listado de preguntas y respuestas esperadas utilizadas para la evaluación

Nº	Pregunta	Respuesta esperada
1	¿Cuál es la capital de Francia?	París
2	¿Quién escribió Don Quijote de la Mancha?	Miguel de Cervantes
3	¿Cuál es el río más largo del mundo?	Nilo
4	¿En qué año llegó el hombre a la Luna?	1969
5	¿Cuál es el elemento químico con símbolo O?	Oxígeno
6	¿Quién pintó la Mona Lisa?	Leonardo da Vinci
7	¿Cuál es el país más grande del mundo?	Rusia
8	¿Cuántos planetas hay en el sistema solar?	8
9	¿Cuál es el océano más grande del mundo?	Océano Pacífico
10	¿Quién fue el primer presidente de Estados Unidos?	George Washington
11	¿Cuál es el inventor del teléfono?	Alexander Graham Bell
12	¿Cuál es el país más antiguo del mundo?	San Marino
13	¿Qué idioma se habla en Brasil?	Portugués
14	¿Qué órgano del cuerpo humano bombea la sangre?	Corazón
15	¿Cuál es la moneda oficial de Japón?	Yen

Los resultados los podemos observar en la Tabla 7.2.

Tabla 7.2: Evaluación comparativa de modelos

Pregunta	distilgpt2		Gemini		gpt-neo-125m	
	BERTScore	Gemini	BERTScore	Gemini	BERTScore	Gemini
Pregunta 1	58.5 %	0.0 %	72.4 %	100.0 %	59.9 %	0.0 %
Pregunta 2	64.4 %	1.2 %	75.6 %	100.0 %	56.3 %	4.3 %
Pregunta 3	55.9 %	0.0 %	65.0 %	100.0 %	61.1 %	0.5 %
Pregunta 4	51.1 %	3.0 %	66.2 %	100.0 %	54.4 %	0.0 %
Pregunta 5	58.0 %	6.0 %	74.3 %	100.0 %	59.6 %	1.3 %
Pregunta 6	55.3 %	0.0 %	82.0 %	100.0 %	60.8 %	7.9 %
Pregunta 7	50.0 %	3.1 %	64.3 %	100.0 %	54.5 %	2.0 %
Pregunta 8	57.7 %	8.5 %	63.8 %	100.0 %	58.4 %	0.0 %
Pregunta 9	59.2 %	0.5 %	76.7 %	92.0 %	58.1 %	0.0 %
Pregunta 10	59.9 %	2.3 %	71.8 %	100.0 %	57.6 %	6.5 %
Pregunta 11	53.9 %	0.0 %	72.6 %	92.5 %	51.4 %	1.0 %
Pregunta 12	57.3 %	0.0 %	51.0 %	95.0 %	57.4 %	2.0 %
Pregunta 13	56.3 %	3.4 %	70.6 %	100.0 %	60.2 %	0.0 %
Pregunta 14	56.0 %	7.0 %	60.5 %	100.0 %	58.8 %	4.3 %
Pregunta 15	58.8 %	0.0 %	57.2 %	100.0 %	57.4 %	5.0 %

Observando estos resultados, podemos ver que el modelo que mejor se comporta es Gemini, ya que obtiene unos resultados muy buenos y consistentes en ambos scores. Por otro lado, los otros modelos obtienen un alto BERTScore pero un muy bajo score de Gemini, esto puede ser debido a que BERTScore analiza las palabras pero no es capaz de analizar el contexto ni la intención, simplemente captura similitudes contextuales incluso si las palabras no tienen nada que ver. En cambio, gemini al ser un modelo de lenguaje LLM previamente entrenado, es capaz de entender todo el contexto. Esto lo que permite es que, aún dando como respuesta palabras que contextualmente pueden estar relaciona-

das, si esas palabras no forman una frase con sentido o tienen relación con la pregunta, proporcionará un score muy bajo.

7.6.2. Pruebas automatizadas en agentes hijo

Para las pruebas en los hijos haremos una aproximación diferente. En primer lugar se han seleccionado 5 tópicos distintos para realizar pruebas. Estos tópicos son los siguientes:

1. Deportes
2. Matemáticas
3. Cine
4. Historia
5. Física

Estos tópicos han sido seleccionados de manera aleatoria, para poder generar muchas respuestas por parte de los modelos, y así poder también sacar una conclusión acerca de cuáles son los posibles tópicos en los que cada modelo se desarrolla mejor. Esto también aporta a poder realizar más pruebas por cada modelo, pudiendo así sacar también un score total sobre cada modelo.

Es importante destacar el flujo de las pruebas. Cada modelo ejecutará 5 baterías de 10 pruebas. Cada batería será sobre uno de los tópicos de la lista. Las preguntas serán generadas por un modelo diferente en cada ejecución, es decir, al tener 3 modelos, en primer lugar en el modelo padre elegiremos uno de esos modelos, y desde ese padre, lanzaremos 3 hijos, cada hijo con uno de los modelos. Como resultado tendremos un sistema con cuatro agentes: un padre y tres hijos. El modelo del padre será el que se repita, de esta manera podemos probarlo como generador de preguntas y como generador de respuestas. En total, se ejecutarán unas 100 preguntas por modelo (50 con BERTScore y 50 con Gemini), haciendo un total de 300 entre los 3 modelos.

Las baterías de preguntas son totalmente aleatorias, generadas por el modelo seleccionado en cada caso.

Se han seleccionado una batería generada por Gemini como ejemplo para mostrar como son las preguntas que se generan automáticamente. En este caso el tópico era Cine.

1. ¿Cómo ha evolucionado la representación de la mujer en el cine desde sus inicios?
2. ¿Qué papel juega la música en la construcción de la narrativa cinematográfica?
3. ¿De qué manera el contexto social influye en la producción y recepción de una película?
4. ¿Cuáles son las diferencias clave entre el cine de autor y el cine comercial?
5. ¿Cómo se define y analiza un plano cinematográfico en términos de composición y significado?
6. ¿Qué impacto ha tenido la tecnología digital en la creación y distribución de películas?

7. ¿Qué elementos considera relevantes para evaluar la calidad de una actuación cinematográfica?
8. ¿Cómo se manifiesta la ideología en las películas y qué herramientas se utilizan para analizarla?
9. ¿Qué estrategias narrativas son comunes en el género del cine noir?
10. ¿Qué ejemplos concretos demuestran el poder del cine como herramienta de propaganda?

Acto seguido se mostraran en la siguiente tabla los resultados de todas las pruebas realizadas con los tópicos mencionados previamente.

Tabla 7.3: Evaluación comparativa de modelos por tópicos

Tópico	distilgpt2		Gemini		gpt-neo-125m	
	BERTScore	Gemini	BERTScore	Gemini	BERTScore	Gemini
Deporte	57.0 %	0.0 %	72.1 %	83.1 %	55.2 %	1.0 %
Matemáticas	60.1 %	0.0 %	69.4 %	98.5 %	59.5 %	0.6 %
Cine	58.8 %	0.0 %	66.3 %	97.0 %	66.1 %	0.1 %
Historia	57.7 %	0.0 %	60.0 %	98.0 %	63.2 %	3.0 %
Física	59.5 %	0.0 %	63.1 %	99.2 %	67.3 %	2.2 %

Al comparar el rendimiento de los modelos de lenguaje distilgpt2, Gemini y gpt-neo-125m, se observa una diferencia muy clara entre ellos. El modelo Gemini destaca por encima del resto en todas las áreas evaluadas, obteniendo puntuaciones casi perfectas en su propio sistema de evaluación (entre el 83,1 % y el 99,2 %) y también alcanzando los valores más altos en la métrica BERTScore (entre el 60,0 % y el 72,1 %). Por el contrario, distilgpt2 y gpt-neo-125m presentan resultados mucho más bajos. Aunque en BERTScore logran cifras aceptables (entre el 55,2 % y el 67,3 %), en la evaluación de Gemini apenas alcanzan puntuaciones, muchas veces quedándose en un 0,0 %. Esto refleja que sus respuestas, en general, no son adecuadas ni coherentes.

Una de las razones principales de esta diferencia está en el tamaño y la complejidad de los modelos. Gemini es un modelo de última generación, mucho más avanzado que distilgpt2 y gpt-neo-125m, que son más antiguos y pequeños. Esto significa que no tienen la misma capacidad para entender preguntas complejas ni para generar respuestas que sean claras, precisas y bien estructuradas.

Además, la diferencia también se explica por cómo se mide el rendimiento. BERTScore se basa en la similitud entre palabras, por lo que si un modelo genera palabras parecidas a las de la respuesta correcta, puede obtener una buena puntuación, incluso si el texto no tiene mucho sentido. En cambio, la evaluación de Gemini es más exigente: tiene en cuenta si la respuesta es correcta, coherente y lógica en su conjunto. Por eso, los modelos más simples no logran buenos resultados en esta métrica, lo que deja en evidencia sus limitaciones cuando se enfrentan a tareas más complejas.

7.7 Puntos a destacar en la aproximación 2

La segunda aproximación representa una evolución significativa respecto a la primera versión, surgiendo como una aproximación multiagente para el análisis de modelos

de lenguaje. Mientras que la versión 1.0 se centraba en una implementación más básica y directa, la versión 2.0 introduce un enfoque completamente distinto, orientado a una arquitectura más modular y escalable.

La principal diferencia conceptual radica en que la versión 2.0 incorpora un sistema multiagente que permite una evaluación más profunda y detallada de los modelos de lenguaje. Esta nueva arquitectura permite la interacción entre diferentes agentes especializados, cada uno con roles específicos en el proceso de evaluación y análisis.

En términos de funcionalidad, la versión 2.0 introduce mejoras significativas en la gestión de modelos, implementando un sistema más robusto para el almacenamiento y carga de modelos, así como una gestión más eficiente de la configuración.

Una de las innovaciones más destacadas es la implementación de un sistema de protocolos de comunicación entre agentes, que permite una interacción más fluida y coordinada. Esto se traduce en una capacidad mejorada para generar texto proveniente de los modelos, evaluar respuestas y analizar el rendimiento de los modelos de manera más exhaustiva.

La versión 2.0 también introduce mejoras significativas en la interfaz de usuario, ofreciendo visualizaciones más detalladas y herramientas de análisis más completas. Esto incluye gráficos comparativos, métricas de satisfacción más detalladas y un sistema de evaluación más robusto que permite un análisis más profundo del rendimiento de los modelos.

La gestión de pruebas y casos de test también ha sido mejorada significativamente, permitiendo una configuración más flexible y detallada de los escenarios de prueba. Esto incluye la capacidad de ejecutar pruebas con diferentes modelos y configuraciones, así como un sistema más robusto para el almacenamiento y análisis de resultados.

La parte mas importante de esta segunda aproximación, reside en poder lanzar agentes con el modelo que el usuario desee, puede ser tanto API como un modelo local descargado del protal HuggingFace.

En resumen, la versión 2.0 representa una evolución significativa en términos de arquitectura, funcionalidad y capacidades de análisis, surgiendo como una herramienta más completa y sofisticada para la evaluación de modelos de lenguaje. Esta nueva versión no solo mejora las capacidades existentes, sino que introduce conceptos y funcionalidades completamente nuevos, reflejando un enfoque más maduro y orientado a la investigación en el campo del análisis de modelos de lenguaje

CAPÍTULO 8

Conclusiones

La realización de este Trabajo de Fin de Grado ha permitido no solo adquirir un conocimiento profundo sobre los modelos de lenguaje actuales, sino también comprobar de forma práctica las diferencias entre modelos de distintas escalas y arquitecturas. El desarrollo de una herramienta interactiva y automatizada para la evaluación comparativa ha demostrado ser una solución eficaz y versátil para analizar el rendimiento de modelos de lenguaje tanto locales como basados en API, como es el caso de Gemini.

A lo largo del proyecto, se ha validado la importancia de emplear un enfoque dual de evaluación que combine métricas automáticas como BERTScore con evaluaciones cualitativas más completas como las que ofrece Gemini. Esta combinación ha evidenciado que, si bien los modelos más pequeños pueden obtener puntuaciones aceptables en métricas tradicionales, su rendimiento real frente a tareas más exigentes y contextos complejos es limitado. En cambio, modelos de última generación como Gemini han mostrado una capacidad superior para generar respuestas coherentes, relevantes y contextualmente apropiadas, confirmando así la ventaja de las arquitecturas LLM en escenarios más complejos.

Asimismo, el enfoque iterativo adoptado durante el desarrollo ha permitido no solo mejorar continuamente el sistema, sino también adaptar sus funcionalidades a distintos tipos de usuarios y necesidades. La integración del framework SPADE ha resultado clave para construir una arquitectura distribuida basada en agentes, capaz de ejecutar pruebas de manera concurrente, flexible y escalable. Esta capacidad de orquestar diferentes modelos y comparar sus resultados de forma automatizada aporta un valor significativo al análisis de rendimiento, especialmente en entornos de investigación o desarrollo donde el volumen de pruebas es elevado.

Otro aspecto a destacar ha sido la importancia de ofrecer una interfaz accesible e intuitiva, lo cual facilita el uso de la herramienta por parte de usuarios no especializados. Gracias a esta interfaz, cualquier persona interesada puede explorar las capacidades de los modelos, comparar resultados, aportar valoraciones subjetivas y entender, de forma visual y clara, las diferencias entre ellos.

En definitiva, este trabajo no solo ha cumplido los objetivos planteados inicialmente, sino que ha abierto nuevas líneas de mejora y ampliación. La herramienta desarrollada puede servir como base para futuros proyectos relacionados con la evaluación, personalización o entrenamiento de modelos de lenguaje, especialmente en un momento en que la inteligencia artificial está cada vez más presente en nuestras vidas. Además, invita a seguir reflexionando sobre cómo medir de forma justa y completa el rendimiento de estos sistemas, fomentando una aproximación crítica y consciente al desarrollo de tecnologías basadas en lenguaje natural.

Bibliografía

- [1] Tom Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared D Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, et al. Language models are few-shot learners. *Advances in neural information processing systems*, 33:1877–1901, 2020.
- [2] Tom B. Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, Sandhini Agarwal, Ariel Herbert-Voss, Gretchen Krueger, Tom Henighan, Rewon Child, Aditya Ramesh, Daniel M. Ziegler, Jeffrey Wu, Clemens Winter, Christopher Hesse, Mark Chen, Eric Sigler, Mateusz Litwin, Scott Gray, Benjamin Chess, Jack Clark, Christopher Berner, Sam McCandlish, Alec Radford, Ilya Sutskever, and Dario Amodei. Language models are few-shot learners, 2020.
- [3] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. Bert: Pre-training of deep bidirectional transformers for language understanding, 2019.
- [4] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. Bert: Pre-training of deep bidirectional transformers for language understanding. 2019.
- [5] Google AI. *Google Gemini Documentation*, 2024.
- [6] Nataliya Kosmyna, Eugene Hauptmann, Ye Tong Yuan, Jessica Situ, Xian-Hao Liao, Ashly Vivian Beresnitzky, Iris Braunstein, and Pattie Maes. Your brain on chatgpt: Accumulation of cognitive debt when using an ai assistant for essay writing task, 2025.
- [7] Javier Palanca, Andrés Terrasa, Vicente Julian, and Carlos Carrascosa. Spade 3: Supporting the new generation of multi-agent systems. *IEEE Access*, 8:182537–182549, 2020.
- [8] Kishore Papineni, Salim Roukos, Todd Ward, and Wei-Jing Zhu. Bleu: a method for automatic evaluation of machine translation. pages 311–318, 2002.
- [9] Alec Radford, Jeffrey Wu, Rewon Child, David Luan, Dario Amodei, Ilya Sutskever, et al. Language models are unsupervised multitask learners. *OpenAI blog*, 1(8):9, 2019.
- [10] Colin Raffel, Noam Shazeer, Adam Roberts, Katherine Lee, Sharan Narang, Michael Matena, Yanqi Zhou, Wei Li, and Peter J. Liu. Exploring the limits of transfer learning with a unified text-to-text transformer, 2023.
- [11] Ricardo Rei, Craig Stewart, Ana C Farinha, and Alon Lavie. Comet: A neural framework for mt evaluation, 2020.

-
- [12] Victor Sanh, Lysandre Debut, Julien Chaumond, and Thomas Wolf. Distilbert, a distilled version of bert: smaller, faster, cheaper and lighter, 2020.
 - [13] Thibault Sellam, Dipanjan Das, and Ankur P. Parikh. Bleurt: Learning robust metrics for text generation, 2020.
 - [14] Thomas Wolf, Lysandre Debut, Victor Sanh, Julien Chaumond, Clement Delangue, Anthony Moi, Pierric Cistac, Tim Rault, Rémi Louf, Morgan Funtowicz, et al. Transformers: State-of-the-art natural language processing. In *Proceedings of the 2020 conference on empirical methods in natural language processing: system demonstrations*, pages 38–45, 2020.
 - [15] Tianyi Zhang, Varsha Kishore, Felix Wu, Kilian Q. Weinberger, and Yoav Artzi. Bertscore: Evaluating text generation with bert. *arXiv preprint arXiv:1904.09675*, 2019.
 - [16] Yizhe Zhang, Siqi Sun, Michel Galley, Yen-Chun Chen, Chris Brockett, Xiang Gao, Jianfeng Gao, Jingjing Liu, and Bill Dolan. Dialogpt: Large-scale generative pre-training for conversational response generation, 2020.

APÉNDICE A

ODS

OBJETIVOS DE DESARROLLO SOSTENIBLE

Objetivos de Desarrollo Sostenible	Alto	Medio	Bajo	No procede
ODS 1. Fin de la pobreza.				x
ODS 2. Hambre cero.				x
ODS 3. Salud y bienestar.				x
ODS 4. Educación de calidad.	x			
ODS 5. Igualdad de género.				x
ODS 6. Agua limpia y saneamiento.				x
ODS 7. Energía asequible y no contaminante.				x
ODS 8. Trabajo decente y crecimiento económico.				x
ODS 9. Industria, innovación e infraestructuras.	x			
ODS 10. Reducción de las desigualdades.	x			
ODS 11. Ciudades y comunidades sostenibles.				x
ODS 12. Producción y consumo responsables.				x
ODS 13. Acción por el clima.				x
ODS 14. Vida submarina.				x
ODS 15. Vida de ecosistemas terrestres.				x
ODS 16. Paz, justicia e instituciones sólidas.				x
ODS 17. Alianzas para lograr objetivos.	x			

El presente Trabajo Fin de Grado no solo se enmarca dentro del ámbito de la ingeniería informática y la inteligencia artificial, sino que también se alinea con varios de los Objetivos de Desarrollo Sostenible (ODS) establecidos por la ONU en la Agenda 2030. A continuación, se detallan los ODS más relevantes y cómo este proyecto contribuye a su cumplimiento: ODS 4: Educación de calidad. Este proyecto contribuye al desarrollo de herramientas accesibles para la evaluación de modelos de lenguaje, lo cual puede aplicarse en entornos educativos mediante sistemas inteligentes de apoyo al aprendizaje. La posibilidad de integrar modelos ligeros (SLM) permite que centros educativos o proyectos con recursos limitados puedan beneficiarse del uso de tecnologías de procesamiento del lenguaje natural, fomentando así una educación más personalizada y equitativa. ODS 9: Industria, innovación e infraestructura. La propuesta técnica del trabajo, basada en una arquitectura multiagente desarrollada con SPADE y la integración de métricas avanzadas como BERTScore y Gemini, representa una contribución directa a la innovación tecnológica. El sistema desarrollado promueve la investigación y experimentación con distintos tipos de modelos de lenguaje, contribuyendo a la mejora de las infraestructuras digitales

que sostienen aplicaciones inteligentes. ODS 10: Reducción de las desigualdades. Al permitir comparar el rendimiento de modelos locales y modelos por API, esta herramienta favorece la equidad tecnológica, ya que ofrece alternativas viables a desarrolladores y organizaciones que no disponen de acceso a modelos comerciales de gran escala. Esto ayuda a reducir la brecha tecnológica entre quienes tienen acceso a recursos avanzados y quienes no. ODS 17: Alianzas para lograr los objetivos. El uso de plataformas abiertas como GitHub y Hugging Face, así como la documentación detallada y el enfoque modular del desarrollo, promueven la colaboración entre desarrolladores, investigadores y comunidades interesadas en la inteligencia artificial. Estas alianzas y el uso de recursos compartidos son clave para fomentar la innovación responsable y sostenible.

APÉNDICE B

Manual de instalación

A continuación se explicará de manera clara como poder descargar la herramienta y utilizarla. Es importante destacar que funciona en cualquier sistema operativo que pueda tener Python instalado.

B.1 Descarga del trabajo

B.1.1. Acceso a github

Para poder acceder al código de este trabajo, deberemos acceder al repositorio de GitHub¹(figura B.1).

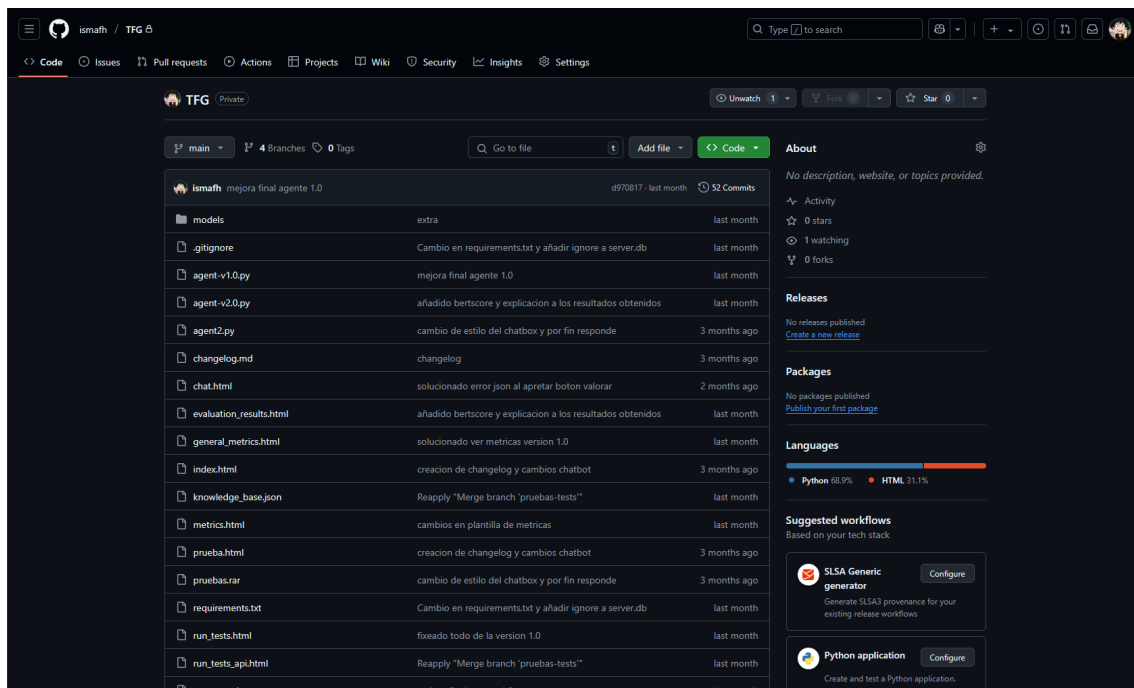


Figura B.1: Repositorio de GitHub

¹<https://github.com/ismafh/TFG>

B.1.2. Clonación del repositorio

Una vez tenemos el repositorio localizado, deberemos copiar el link para poder clonarlo en nuestro dispositivo. Podemos observar de donde obtener el link en la figura B.2

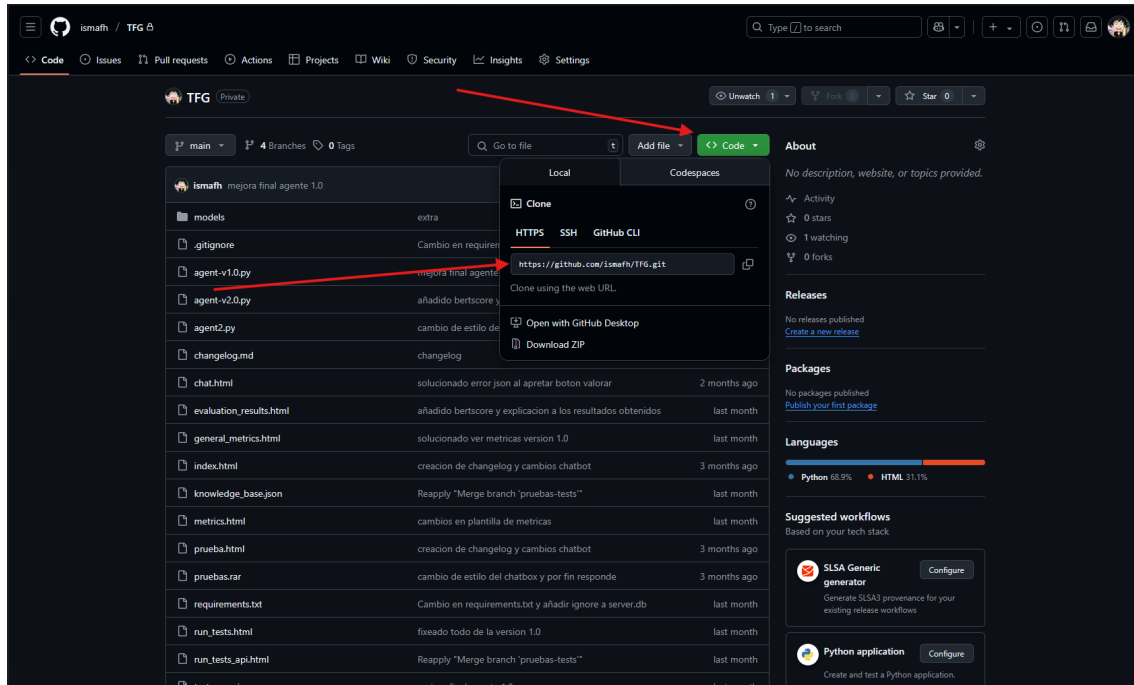


Figura B.2: Link para poder clonar repositorio

Acto seguido, deberemos abrirnos una terminal para poder ejecutar el siguiente comando: `git clone https://github.com/ismafh/TFG` (figura B.3)

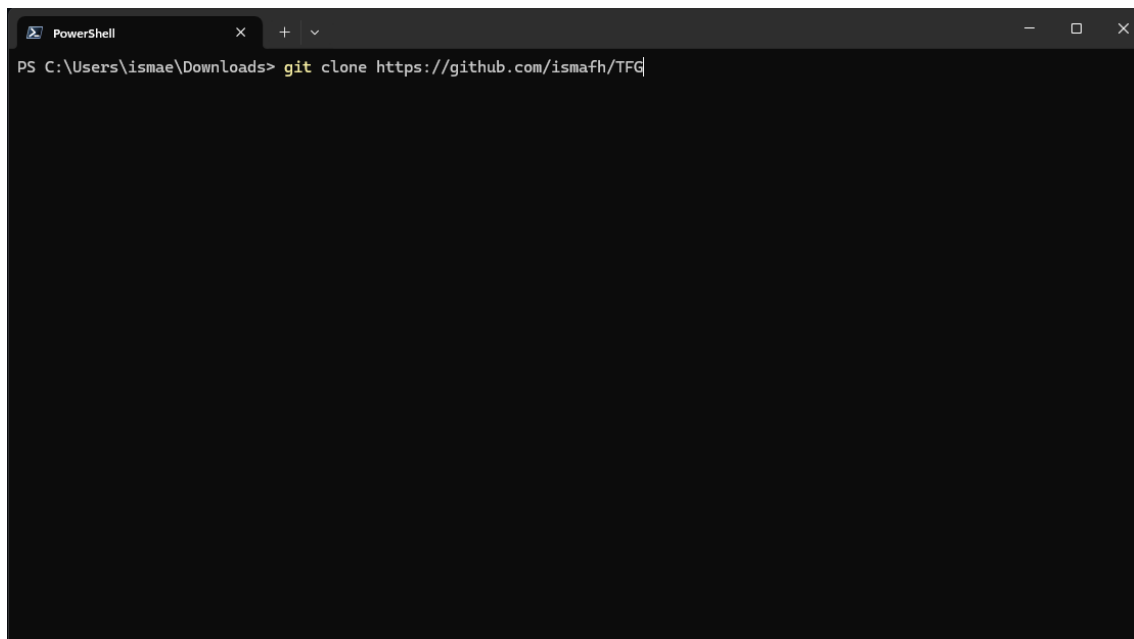


Figura B.3: Comando para clonar el repositorio

Una vez tenemos clonado el repositorio podemos pasar a la siguiente sección.

B.2 Puesta en marcha

Ahora deberemos acceder a la carpeta del repositorio y abrir una terminal en ella. Podemos observar en la figura B.4 el paso a seguir para ello.

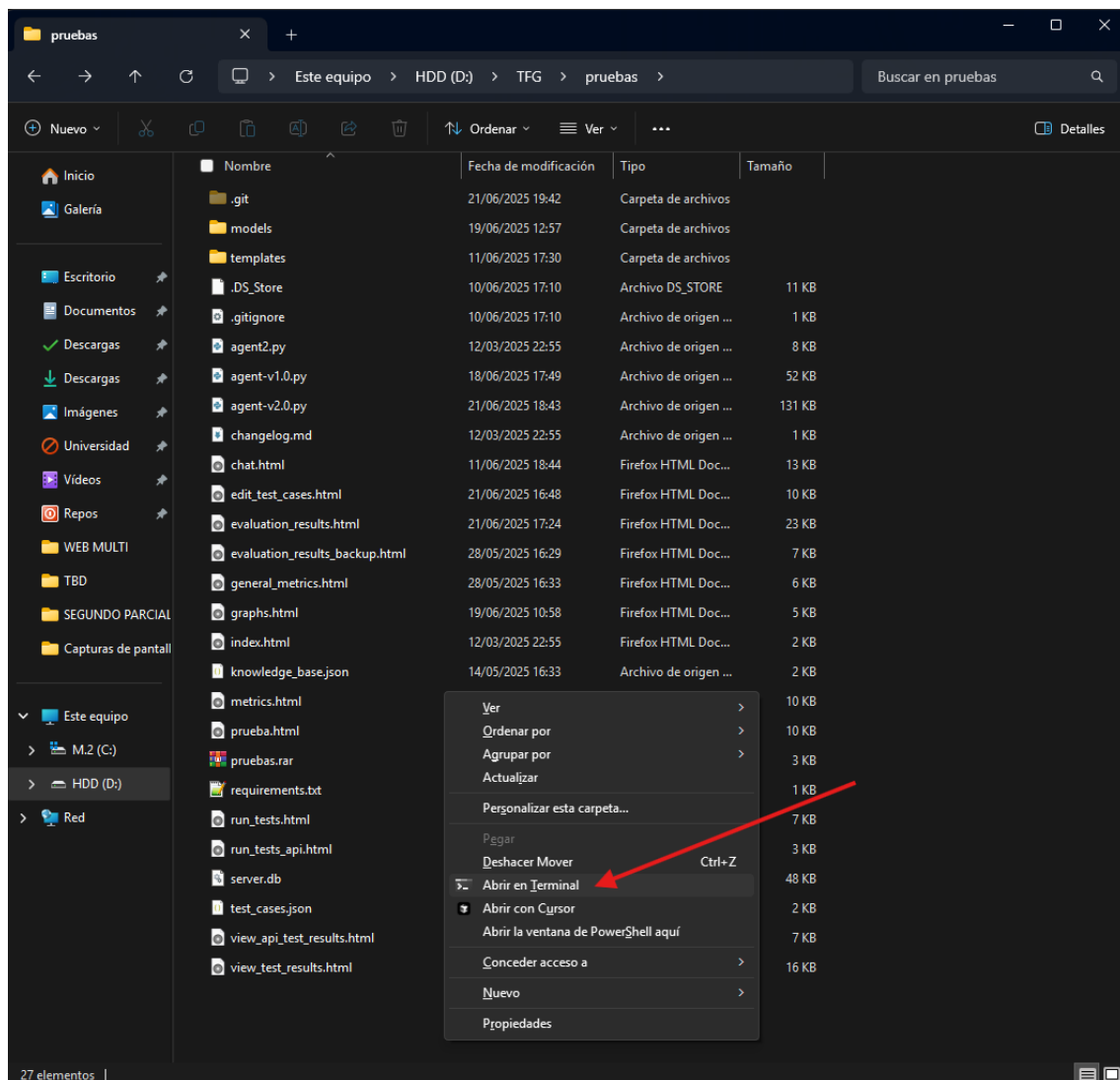


Figura B.4: Paso para abrir terminal en directorio

Una vez abierta la terminal, deberemos ejecutar el siguiente comando (figura B.5):
`python -m venv venv` (en el caso de estar en macOS el comando seria `python3 -m venv venv`)

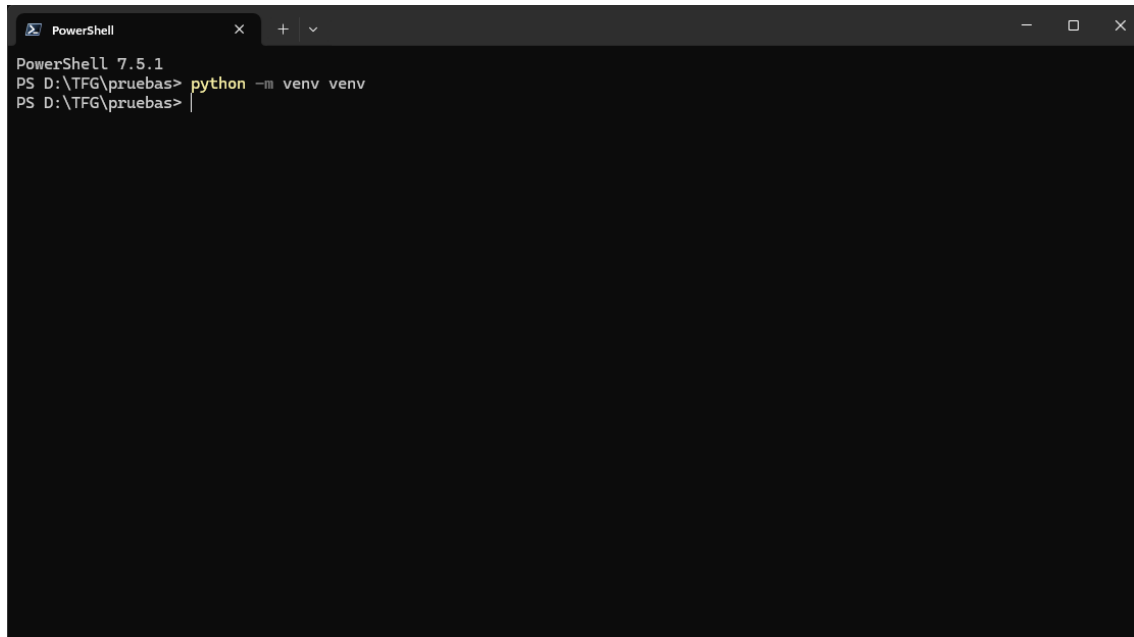


Figura B.5: Paso para ejecutar un entorno virtual

Esto sirve para crear un entorno virtual de Python donde poder instalar todas las librerías necesarias para la ejecución de la herramienta.

Una vez tengamos el entorno virtual de Python creado, debemos de acceder a el utilizando el comando `.\venv\Scripts\activate` (si estamos en macOS o en linux el comando es `source venv/bin/activate`) (figura B.6)

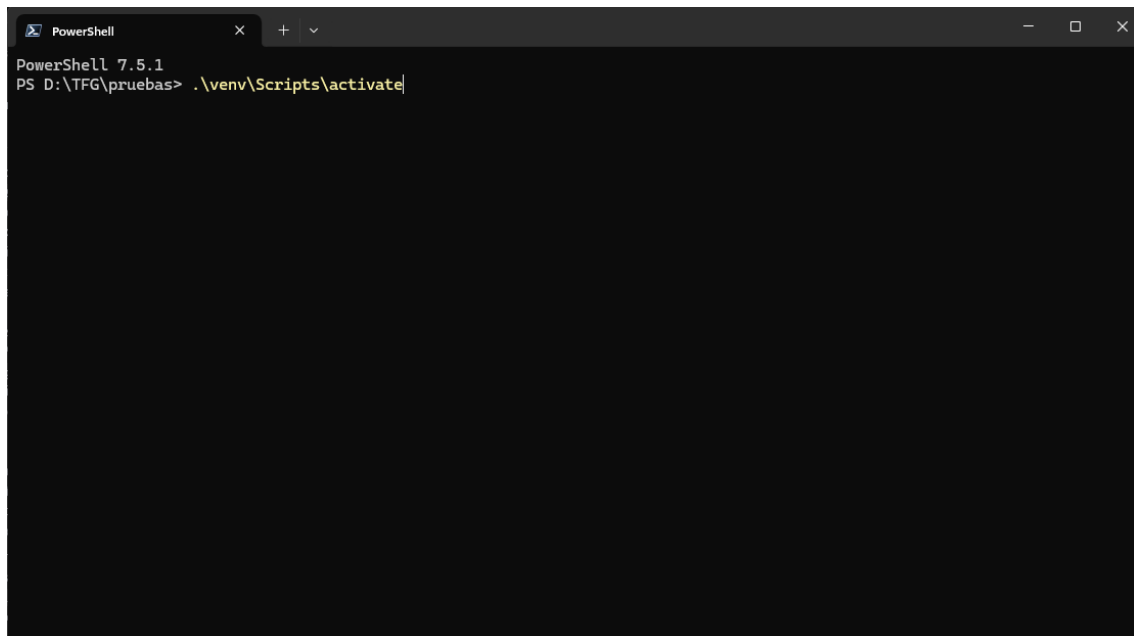
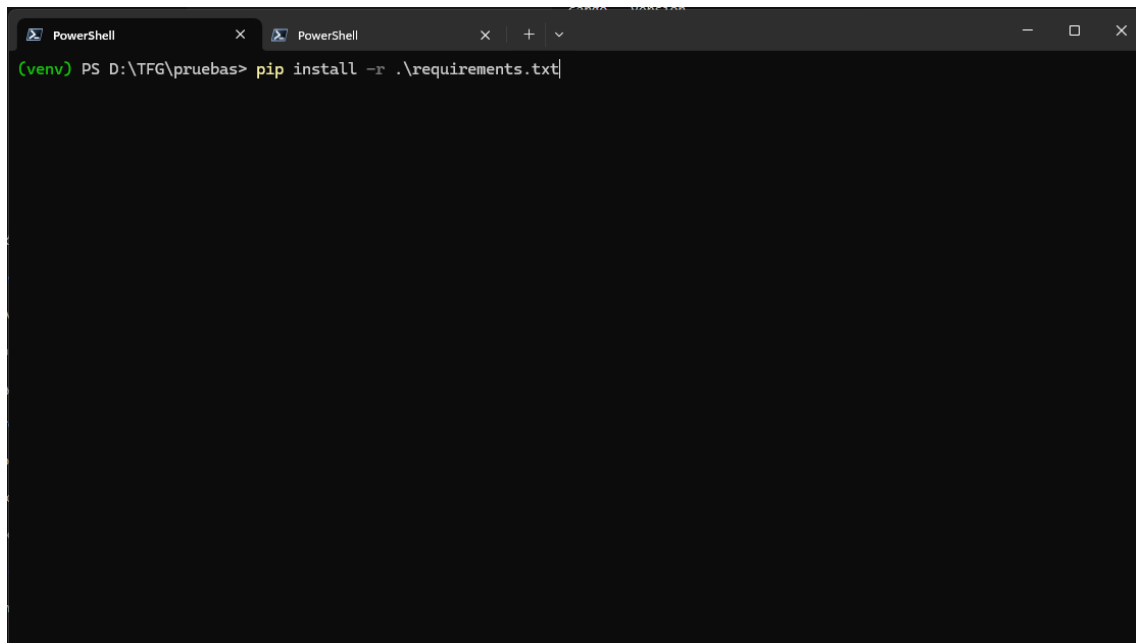


Figura B.6: Paso para entrar en el entorno virtual

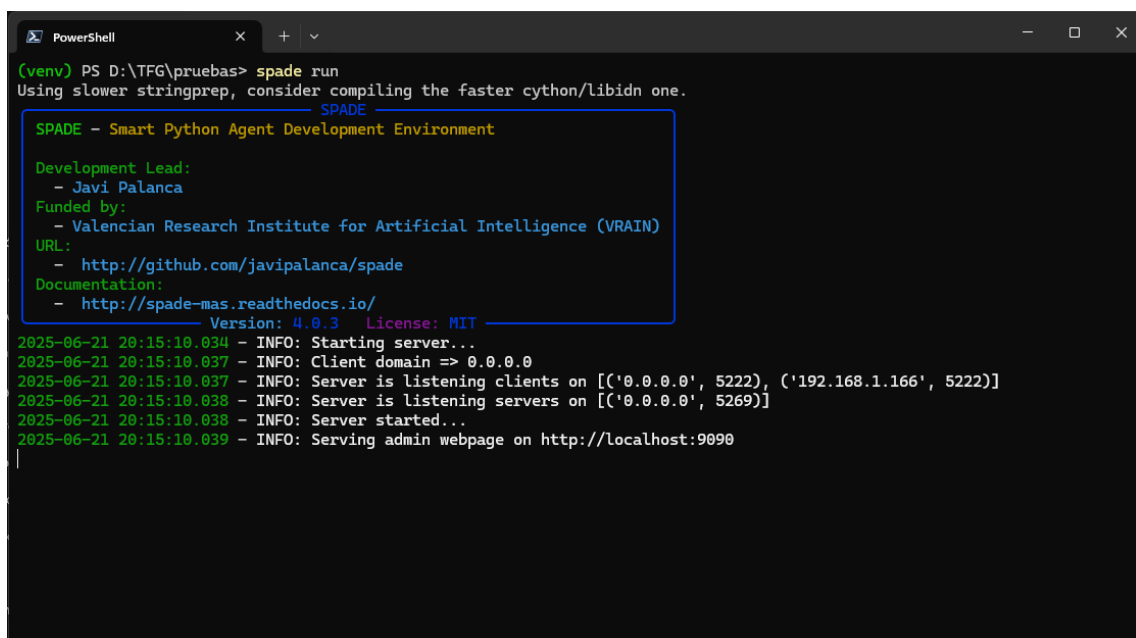
Cuando estemos dentro del entorno virtual lo único que nos falta es descargar las dependencias necesarias para poder hacer que la herramienta funcione. Estas dependencias están ya predefinidas en un archivo por lo que solo necesitaremos realizar un comando que nos las descargará automáticamente. El comando es `pip install -r requirements.txt` (figura B.7):



```
(venv) PS D:\TFG\pruebas> pip install -r .\requirements.txt
```

Figura B.7: Paso para descargar dependencias

Una vez descargadas todas las dependencias, podremos lanzar el servidor de SPADE con el siguiente comando (figura B.8): `spade run`



```
(venv) PS D:\TFG\pruebas> spade run
Using slower stringprep, consider compiling the faster cython/libidn one.

SPADE - Smart Python Agent Development Environment
Development Lead:
- Javi Palanca
Funded by:
- Valencian Research Institute for Artificial Intelligence (VRAIN)
URL:
- http://github.com/javipalanca/spade
Documentation:
- http://spade-mas.readthedocs.io/
Version: 4.0.3 License: MIT

2025-06-21 20:15:10.034 - INFO: Starting server...
2025-06-21 20:15:10.037 - INFO: Client domain => 0.0.0.0
2025-06-21 20:15:10.037 - INFO: Server is listening clients on [('0.0.0.0', 5222), ('192.168.1.166', 5222)]
2025-06-21 20:15:10.038 - INFO: Server is listening servers on [('0.0.0.0', 5269)]
2025-06-21 20:15:10.038 - INFO: Server started...
2025-06-21 20:15:10.039 - INFO: Serving admin webpage on http://localhost:9090
```

Figura B.8: Paso para lanzar SPADE

Con SPADE lanzado, ya solo queda lanzar la herramienta con el siguiente comando (figura B.9): `python agent-v2.0.py`

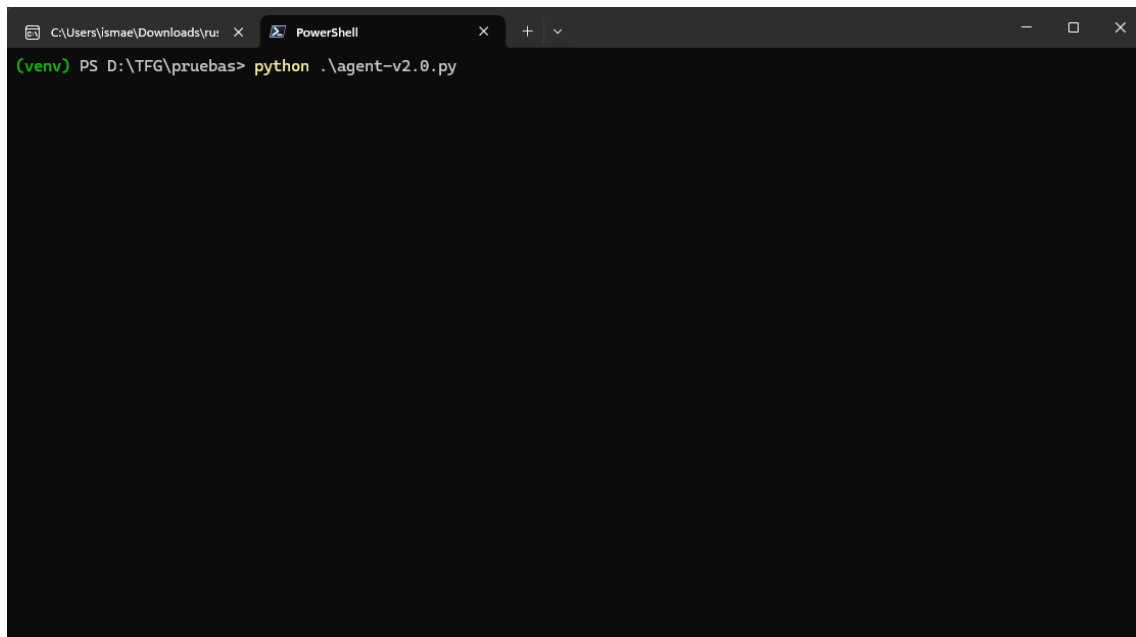


Figura B.9: Paso para lanzar la herramienta

Finalmente ya tendremos la herramienta operativa, para acceder a ella ahora solo falta entrar en el navegador a la IP `http://127.0.0.1:10000/index`