

Universidad de La Habana
Facultad de Matemática y Computación



MoLex: un prototipo de plataforma para la asistencia a la escritura científica basada en LLMs

Autor:

Sebastián Suárez Gómez

Tutor:

Dr. Yudivian Almeida Cruz

Trabajo de Diploma
presentado en opción al título de
Licenciado en Ciencia de la Computación

Febrero de 2025

<https://github.com/sebas-suarez01/MolexEditor>

A mi tío Salazar, que hubiese querido estar presente en este momento.

Agradecimientos

Mis agradecimientos más importantes son para mis padres, quienes han sido mi pilar fundamental en cada etapa de mi vida. A ellos les debo no solo este logro, sino todos los valores, el esfuerzo y la perseverancia que me han inculcado desde siempre.

Ocasiones difíciles se han presentado y agradezco a mis hermanos, tíos, abuelos y otros familiares por estar presente y apoyarme en ellas y durante este proceso. Su preocupación y acompañamiento han sido un aliento constante durante esta etapa.

Sin duda, agradezco a mi esposita por todo el apoyo y el amor que me brindó durante la realización de esta tesis y me faltarán las palabras para expresar lo afortunado que soy por tenerla a mi lado.

A mis amigos, Héctor, Karen, Luis y Javier, quienes me han apoyado en los momentos más exigentes de la carrera. Agradezco todas las experiencias que hemos vivido a lo largo de estos cinco años y sé que siempre podré contar con ellos, como ellos conmigo.

A mi tutor por guiarme durante este proceso y a todos los profesores que me ayudaron a llegar hasta aquí.

A todos ustedes, muchas gracias de corazón.

Opinión del tutor

Título de la tesis: MoLex: un prototipo de plataforma para la asistencia a la escritura científica basada en LLMs

Estudiante: Sebastián Suárez Gómez

Tutor: Dr. Yudivián Almeida Cruz

El estudiante Sebastián Suárez Gómez desarrolló satisfactoriamente el trabajo de diploma titulado “MoLex: un prototipo de plataforma para la asistencia a la escritura científica basada en LLMs”. En este trabajo el estudiante propuso el diseño e implementación de un prototipo de un sistema para la asistencia a la escritura científica a partir de la integración de un editor texto con un Modelo de Lenguaje (LLM) aplicando técnicas tanto de *prompt engineering* como de *retrieval-augmented generation* (RAG).

El trabajo propone un acercamiento a la utilización de modelos de lenguajes como una herramienta de soporte a la generación de texto, particularizado en el ámbito científico. La propuesta se basa en una arquitectura modular con un *front-end* basado en la plataforma *Lexical* que interactúa para tener nuevas funcionalidades con un modelo de lenguaje a través de una API desarrollada con FastAPI. La interacción es enriquecida a partir del uso de técnicas de *prompt engineering* que se enfocan en mejorar la precisión de los textos que se pueden generar o modificar así como un módulo de RAG para la generación de referencias que son una necesidad en la escritura científica. De esta manera se llega a un prototipo que permite la redacción, revisión y estructuración de documentos científicos, la generación automática de contenido, corrección gramatical y estilística, y gestión de referencias bibliográficas.

Para poder afrontar el trabajo, el estudiante tuvo que revisar literatura científica relacionada con la temática así como soluciones existentes y bibliotecas de software que pueden ser apropiadas para su utilización. Todo ello con sentido crítico, determinando las mejores aproximaciones y también las dificultades que presentan.

Todo el trabajo fue realizado por el estudiante con una buena dosis de constancia, capacidad de trabajo individual y habilidades, tanto de gestión, como de desarrollo y de investigación.

Por estas razones pedimos que le sea otorgada al estudiante Sebastián Suárez

Gómez una calificación que le permita obtener el título de Licenciado en Ciencia de la Computación y así, por derecho propio, se pueda integrar al gremio de los profesionales de la computación.

Dr. Yudivián Almeida Cruz

Resumen

Este trabajo propone el desarrollo de un prototipo de editor de texto científico que integre Modelos de Lenguaje a Gran Escala (LLMs) con técnicas de *prompt engineering* y *Retrieval-Augmented Generation* (RAG). El prototipo diseñado permite la redacción, revisión y estructuración de documentos científicos, la generación automática de contenido, corrección gramatical y estilística, y gestión de referencias bibliográficas.

La implementación del sistema se basa en una arquitectura modular con un *frontend* desarrollado en React y un *backend* construido en Python utilizando FastAPI. Se integran modelos LLM y su uso es optimizado con técnicas de *prompt engineering* para mejorar la precisión de las respuestas y un módulo RAG para la generación de referencias a partir de fuentes seleccionadas. Los resultados obtenidos demuestran la viabilidad del uso de inteligencia artificial generativa para asistir en la escritura académica, mejorando la productividad y calidad de los textos científicos.

Abstract

This research proposes the development of a prototype of a scientific text editor that integrates Large Language Models (LLMs) with prompt engineering and Retrieval-Augmented Generation (RAG) techniques. The designed prototype enables the writing, reviewing, and structuring of scientific documents, automatic content generation, grammatical and stylistic correction, and bibliographic reference management.

The system implementation is based on a modular architecture, with a frontend developed in React and a backend built in Python using FastAPI. LLM models are integrated, and their use is optimized with prompt engineering techniques to improve response accuracy, along with a RAG module for generating references from selected sources. The obtained results demonstrate the feasibility of using generative artificial intelligence to assist in academic writing, enhancing productivity and the quality of scientific texts.

Índice general

Introducción	1
1. Estado del Arte	5
1.1. La Redacción Científica	5
1.1.1. Concepto y características principales	5
1.1.2. Importancia de la Redacción Científica	6
1.1.3. Desafíos comunes en la redacción científica	7
1.1.4. Importancia de las herramientas computacionales en la redac- ción científica	7
1.2. Procesamiento del Lenguaje Natural (NLP)	8
1.2.1. Definición y características	8
1.2.2. Relevancia del NLP en la Escritura Científica	9
1.3. Modelos de Lenguaje a Gran Escala (LLMs)	10
1.3.1. Definición y características	10
1.3.2. Aplicaciones de los LLMs en la escritura científica	10
1.3.3. Limitaciones de los LLMs	11
1.4. Prompt Engineering	12
1.4.1. Definición	12
1.4.2. Importancia del Prompt Engineering	12
1.4.3. Principios Fundamentales	12
1.4.4. Técnicas Comunes de Prompt Engineering	13
1.4.5. Aplicaciones del Prompt Engineering en la Escritura Científica	13
1.5. Retrieval-Augmented Generation (RAG)	14
1.5.1. Definición	14
1.5.2. Funcionamiento	14
1.5.3. Ventajas y Aplicaciones de RAG en la escritura científica . . .	14
1.6. Revisión de propuestas computacionales para la escritura científica . .	15
1.6.1. Herramientas Actuales para la Escritura Científica	15
1.6.2. Identificación de Vacíos Tecnológicos	19

2. Diseño	20
2.1. Componente Visual	21
2.1.1. Estructura del Componente Visual	21
2.2. Componente API	22
2.2.1. Estructura del Componente API	23
2.2.2. Flujo de Comunicación	23
2.2.3. Núcleo	24
3. Implementación	31
3.1. Componente Visual	31
3.1.1. Edición de Texto	31
3.1.2. Importación y exportación de documentos	34
3.2. Componente API	35
3.2.1. Procesamiento	35
3.3. Limitaciones de las Funcionalidades del Programa	45
Conclusiones	47
Recomendaciones	48
Bibliografía	49

Índice de figuras

2.1.	Arquitectura	20
2.2.	Arquitectura de la interfaz gráfica	22
2.3.	Arquitectura de la API	24
2.4.	Diseño del módulo de prompt engineering	26
2.5.	Diseño del flujo lógico del módulo de prompt engineering	27
2.6.	Módulo de RAG	28
2.7.	Diseño del flujo lógico del módulo de RAG.	29
3.1.	Barra de herramientas	33
3.2.	Insertar ecuación.	34
3.3.	Jerarquía de prompts	36
3.4.	Vista de la herramienta para la búsqueda de documentos.	40
3.5.	Vista de la herramienta de revisión textual.	42
3.6.	Vista de la herramienta para la construcción de referencias.	43
3.7.	Constructor de prompts.	44

Introducción

La evolución de las herramientas de escritura ha sido fundamental en el desarrollo de la comunicación científica. Desde las primeras máquinas de escribir hasta los procesadores de texto modernos, cada avance ha permitido a los investigadores plasmar y compartir sus hallazgos de manera más eficiente. La transición de la máquina de escribir al procesador de texto en la década de 1980 marcó un hito significativo, facilitando la edición y distribución de documentos científicos. En la actualidad, la integración de tecnologías avanzadas, como los Modelos de Lenguaje de Gran Escala (LLMs), está transformando la manera en que se redactan y perfeccionan los textos científicos, ofreciendo herramientas que ayudan en la generación y revisión de contenido especializado.

La redacción científica se caracteriza por su rigor, precisión y estructura formal, aspectos que a menudo implican retos importantes para investigadores y estudiantes, especialmente en términos de coherencia, claridad y adherencia a estándares específicos. Si bien existen herramientas que los investigadores utilizan para la redacción científica, muchas no son del todo eficientes.

Por ejemplo, los procesadores de texto convencionales como *Microsoft Word* son más accesibles, pero carecen de sugerencias de estilo técnico, corrección estructural o generación automática de contenido. Los correctores gramaticales como *Grammarly*¹ ofrecen ayuda en gramática y estilo general, pero no están adaptados al lenguaje técnico y al tono formal que requiere la redacción científica. Los editores de texto especializados como *Overleaf*² son ideales para la creación de documentos estructurados, pero tienen una curva de aprendizaje pronunciada y carecen de asistencia integrada para generar contenido lingüístico. Por último, herramientas como *ChatGPT* están diseñadas para adaptarse a un amplio rango de tareas y contextos y ya ofrecen capacidades avanzadas para asistir en la escritura, pero son herramientas generalistas y externas, por lo que no están optimizadas para ajustarse a las necesidades específicas del investigador.

¹URL de Grammarly: <https://www.grammarly.com/>

²URL de Overleaf: <https://es.overleaf.com/>

Motivación

Los LLMs son herramientas con gran potencial en la redacción científica, facilitando la generación de contenido, mejorando la coherencia textual y agilizando la estructuración de documentos académicos. Según Koga (2023), "los LLMs pueden mejorar la productividad en la escritura académica, pero su implementación aún enfrenta limitaciones relacionadas con la precisión de las referencias y la contextualización del contenido generado"[34].

Para abordar estas limitaciones, se utilizan enfoques como el *prompt engineering* y RAG, que permiten optimizar la generación de texto mediante la incorporación de información relevante extraída de bases de datos científicas. Boyko et al. (2023) afirman que "la combinación de LLMs con técnicas de recuperación mejora la precisión y coherencia de los textos científicos, permitiendo una mejor integración del conocimiento en la redacción académica"[5].

Existen herramientas bastante completas para la escritura académica, como Effidit³ y Zettlr⁴, pero ambas están lejos de ser soluciones ideales. Effidit está altamente optimizada para la corrección y sugerencias de texto, pero su aplicabilidad está limitada principalmente a usuarios chinos. Zettlr, por su parte, es una opción eficiente para la organización de referencias y la estructuración de documentos, pero carece de integración con LLMs para mejorar el contenido generado.

Es por eso que es relevante el desarrollo de propuestas que combinen las capacidades de edición de texto con la potencia de los LLMs para contribuir al perfeccionamiento en la escritura científica, con el objetivo de facilitar y mejorar la calidad de la producción académica.

Antecedentes

En la Universidad de La Habana, el uso de LLMs ha sido explorado en diversas investigaciones dentro del campo del Procesamiento del Lenguaje Natural (NLP). En trabajos previos, se ha abordado la integración de estos modelos en la generación automática de resúmenes científicos, optimizando la identificación y síntesis de información relevante a partir de grandes volúmenes de documentos. Estas investigaciones han demostrado la capacidad de los LLMs para estructurar textos de manera coherente y contextualizada, facilitando la revisión de literatura de manera más eficiente[45].

Otro enfoque se centró en la evaluación de las habilidades cognitivas de los LLMs. En este sentido, se han desarrollado conjuntos de datos específicos que permiten analizar el desempeño de estos modelos en tareas que requieren comprensión de lectura, razonamiento lógico y generación de respuestas en lenguaje natural. A partir de estos

³URL de Effidit: <https://effidit.qq.com/en>

⁴URL de Zettlr: <https://www.zettlr.com/>

estudios, se ha podido identificar fortalezas y limitaciones en diferentes modelos, contribuyendo a un mejor entendimiento de su aplicabilidad en distintos contextos[57].

En línea con estos estudios previos, el trabajo actual continúa explorando las capacidades de estos modelos, aplicándolos en un nuevo contexto para abordar un problema específico dentro del procesamiento de información.

Problemática

Actualmente, existe carencia de herramientas que combinen capacidades de generación y revisión de contenido especializado con funcionalidades adaptadas a las necesidades de los investigadores en la escritura científica. Estas necesidades incluyen la generación asistida de texto académico, la detección y corrección de errores estructurales y gramaticales, la gestión de referencias bibliográficas, la mejora de la claridad y cohesión del contenido, y la adaptación del estilo a estándares editoriales específicos. Con el auge de los modelos de lenguaje comienzan a florecer desarrollos de herramientas que contribuyan a incrementar la productividad en la edición de textos, sin embargo todavía no existe una que sea la solución de referencia para el trabajo con textos científicos. La falta de una solución que integre estas funcionalidades representa una barrera significativa para la productividad académica, ya que los investigadores deben recurrir a múltiples herramientas para obtener un resultado óptimo, lo que ralentiza el proceso de redacción y aumenta el margen de error en la estructuración del contenido.

Objetivos

El objetivo general de esta investigación es crear un prototipo de editor de textos científicos que integre LLMs para agilizar la redacción de textos científicos y optimice su uso mediante *prompt engineering* y RAG, con el fin de asistir a los investigadores en la redacción de documentos académicos de alta calidad.

Para lograrlo, es necesario cumplimentar los siguientes objetivos específicos:

- Realizar la revisión del Estado del Arte en editores de textos científicos con integración de LLMs.
- Identificar las limitaciones de las herramientas actuales, incluyendo procesadores de texto convencionales y sistemas de asistencia y edición que hagan uso de LLMs, con el fin de identificar oportunidades de mejora.
- Diseñar e implementar una propuesta de arquitectura para un prototipo de sistema de asistencia a la escritura científica que permita la integración eficiente de LLMs.

- Proponer e implementar modelos que permitan la integración eficiente de técnicas de prompt engineering y RAG en el sistema propuesto.
- Proponer e implementar funcionalidades en el sistema que se ajusten a las exigencias de la redacción científica.

Propuesta de Solución

En este trabajo se hace una propuesta que consiste en desarrollar un prototipo de editor de textos científicos que ofrezca funcionalidades avanzadas que permitan generar y revisar contenido especializado, adaptado a las necesidades de los investigadores, proporcionándoles una herramienta que facilite el proceso de investigación y redacción. La implementación del sistema se desarrollará con una arquitectura modular que permitirá la integración eficiente de sus componentes. Se utilizará un *framework* de edición de texto para facilitar la interacción del usuario y mejorar la experiencia de escritura. Además, se incorporarán LLMs cuyo uso será optimizado con *prompt engineering* para generar respuestas precisas y relevantes, junto con un módulo de RAG para recuperar información de fuentes seleccionadas, citarlas e incorporarlas a la bibliografía.

Estructura de la Tesis

Finalmente, en cuanto a la estructura de esta tesis, se organiza de la siguiente manera: introducción, tres capítulos, conclusiones, recomendaciones y referencias.

En el primer capítulo, se presentan los conceptos y definiciones necesarios para comprender el trabajo, junto con una revisión del estado del arte relacionada con editores de texto, LLMs y su aplicación en la escritura científica.

En el segundo, se describe el proceso de diseño del sistema, incluyendo los usuarios objetivo, la interfaz y el flujo de datos entre los componentes del sistema. También se detallan los casos de uso, las funcionalidades clave y la arquitectura general del prototipo.

En el tercero, se describe el desarrollo de la plataforma propuesta, su implementación, funcionalidades y la integración de los LLMs.

El documento concluye con un análisis de los resultados obtenidos y recomendaciones para futuros trabajos en el área.

Capítulo 1

Estado del Arte

La escritura científica desempeña un papel fundamental en la generación y difusión del conocimiento, siendo el medio por excelencia para comunicar resultados de investigaciones, teorías y hallazgos innovadores. Sin embargo, los procesos tradicionales de redacción presentan desafíos significativos para los investigadores, especialmente en términos de claridad, coherencia y adherencia a los estándares académicos.

En este capítulo se presenta el contexto que sustenta el desarrollo de esta investigación, abordando los conceptos clave y las tecnologías relevantes que guían la propuesta de esta tesis. En particular, se explorarán las bases de la redacción científica, los procesadores de textos tradicionales, el Procesamiento de Lenguaje Natural (NLP), los Modelos de Lenguaje de Gran escala (LLMs) y el *prompt engineering*. Estos elementos forman el núcleo teórico sobre el cual se articula el desarrollo del prototipo de software propuesto. En el siguiente apartado, se profundizará en el primer concepto clave: la redacción científica, sus características esenciales y los desafíos que enfrenta, estableciendo así, el punto de partida para la discusión de las tecnologías que buscan abordarlos.

1.1. La Redacción Científica

1.1.1. Concepto y características principales

La redacción científica es una forma de comunicación técnica y formal utilizada por investigadores, académicos y profesionales para transmitir conocimientos y descubrimientos de manera precisa y estructurada. Se distingue de otros tipos de redacción por su énfasis en la claridad, el rigor y la adherencia a los estándares establecidos. Busca expresar ideas complejas de manera comprensible para una audiencia especializada, como científicos, académicos o estudiantes avanzados. Es fundamental para la comunicación y el avance del conocimiento, permitiendo el intercambio y la validación

de ideas a través de publicaciones en revistas científicas, congresos y tesis. Como lo señala Katz (2009): “la característica esencial de la escritura científica es la claridad. No hay lugar para la ambigüedad, el lenguaje arcano o adornos innecesarios en los archivos de la ciencia” [33].

Se caracteriza por la claridad, ya que los textos deben ser directos y fáciles de entender, eliminando ambigüedades. “La calidad de la escritura mejora la calidad del pensamiento” [23], lo que resalta la importancia de una comunicación científica efectiva. Además, el rigor es crucial: el contenido debe estar basado en datos verificables y debe presentarse de manera objetiva. Como afirma Gernsbacher (2018), “la escritura empírica debe permitir transparencia, reproducibilidad y claridad” [22]. Asimismo, la estructura formal es fundamental en la redacción científica, ya que permite una comprensión más accesible y ordenada de los textos. Finalmente, el uso de referencias es indispensable, pues permite citar adecuadamente las fuentes que respaldan las afirmaciones y dan crédito a investigaciones previas.

1.1.2. Importancia de la Redacción Científica

La redacción científica no solo se utiliza para difundir conocimientos, sino que también es crucial para validar los hallazgos a través de la revisión por pares y fomentar el progreso en diferentes disciplinas. La revisión por pares es un proceso esencial en la publicación académica, ya que garantiza la calidad y credibilidad de los trabajos científicos, además de permitir la retroalimentación de expertos en el campo [1]. Su impacto académico se evidencia en las publicaciones de las investigaciones en un formato comprensible y riguroso que amplía la influencia del trabajo científico y mejora la visibilidad de los investigadores y las instituciones. La publicación en revistas científicas no solo valida los hallazgos, sino que también incrementa la reputación de los autores y contribuye a la consolidación del conocimiento dentro de una comunidad disciplinaria [30].

La redacción científica también facilita el aprendizaje, ya que los textos científicos son clave para la formación de estudiantes e investigadores. La formación en escritura científica es fundamental para que los investigadores puedan comunicar sus ideas de manera efectiva y difundir sus resultados de manera precisa [46]. También promueve el intercambio de ideas a nivel global, facilitando colaboraciones interdisciplinarias. Finalmente, la redacción científica establece un registro de avances científicos, asegurando que los conocimientos generados sean documentados y puedan ser utilizados como base para futuras investigaciones. La documentación sistemática del conocimiento contribuye a la evolución del saber científico y permite que los hallazgos se preserven y sean replicables en el tiempo [62].

1.1.3. Desafíos comunes en la redacción científica

La redacción científica representa un desafío tanto para investigadores experimentados como para principiantes. Un primer obstáculo radica en la estructuración del contenido y el logro de un flujo lógico y coherente entre ideas, lo cual se vuelve particularmente complicado en textos extensos y altamente especializados [29]. La interpretación y organización de los resultados también demandan precisión, ya que la claridad de las ideas técnicas debe mantenerse sin comprometer el rigor del lenguaje científico [10].

Otro desafío importante es la adaptación a formatos específicos de revistas o instituciones, cada una con sus propias normativas y estilos de citación, como APA¹, IEEE² o Chicago³. Esta diversidad de formatos implica una curva de aprendizaje constante y un esfuerzo significativo en la edición de los manuscritos [19]. Además, la revisión por pares exige un nivel de escritura altamente depurado, lo que prolonga el tiempo de publicación y revisión de los trabajos [8].

En este contexto, las herramientas computacionales han cobrado gran relevancia en la escritura científica. Modelos de lenguaje como ChatGPT y otras soluciones basadas en inteligencia artificial han demostrado su capacidad para generar borradores, mejorar la coherencia textual y sugerir reformulaciones en tiempo real [11][49]. Estas tecnologías agilizan la redacción y edición [28].

1.1.4. Importancia de las herramientas computacionales en la redacción científica

El avance de las herramientas computacionales ha revolucionado la escritura científica y el proceso de investigación en múltiples disciplinas. Estas herramientas no solo han optimizado la redacción y la gestión de referencias, sino que también han mejorado la organización de datos, la reproducibilidad de los estudios y la eficiencia en la comunicación científica [47].

Uno de los principales beneficios de las herramientas computacionales es su capacidad para mejorar la precisión y la coherencia de los textos científicos. Procesadores de texto avanzados, como Overleaf⁴, han facilitado la estructuración de documentos científicos, permitiendo a los investigadores enfocarse en el contenido en lugar del formato [43]. Además, correctores gramaticales inteligentes, como Grammarly⁵, ayudan a mejorar la claridad y la calidad del lenguaje académico [55].

¹URL de APA: <https://www.apa.org/>

²URL de IEEE: <https://iee-collabratec.ieee.org/>

³URL de Chicago: <https://www.chicagomanualofstyle.org/>

⁴URL de Overleaf: <https://es.overleaf.com/>

⁵URL de Grammarly: <https://www.grammarly.com/>

La automatización de la gestión de referencias es otro aspecto clave. Herramientas como Mendeley⁶, Zotero⁷ y EndNote⁸ han simplificado la organización de citas y la aplicación de estilos bibliográficos, reduciendo significativamente el tiempo dedicado a esta tarea [51].

Por otro lado, la inteligencia artificial generativa ha comenzado a desempeñar un papel fundamental en la escritura científica. Según Liefke et al. (2021), "los modelos de lenguaje como ChatGPT y BERT han demostrado ser útiles para la generación de resúmenes, la reformulación de ideas y la detección de inconsistencias en manuscritos académicos"[40]. Estas herramientas han sido especialmente valiosas en revisiones sistemáticas y metaanálisis, donde la identificación y síntesis de grandes volúmenes de literatura requieren precisión y eficiencia [12].

El acceso a bases de datos y plataformas de análisis computacional ha facilitado la integración de información proveniente de múltiples fuentes, lo que ha permitido realizar descubrimientos científicos más sólidos. En este sentido, la combinación de literatura científica con herramientas computacionales ha impulsado la generación de nuevas hipótesis y ha mejorado la reproducibilidad de los estudios [54].

A continuación, se explorará el Procesamiento del Lenguaje Natural (NLP) como tecnología empleadas en los asistentes de redacción científica.

1.2. Procesamiento del Lenguaje Natural (NLP)

1.2.1. Definición y características

El procesamiento del lenguaje natural (NLP, por sus siglas en inglés) es una rama de la inteligencia artificial (IA) que se enfoca en la interacción entre las computadoras y los lenguajes humanos. Su objetivo es permitir que las máquinas comprendan, interpreten, generen y respondan al lenguaje humano de manera útil y significativa. Para lograr esto, el NLP combina conceptos de lingüística computacional, aprendizaje automático y redes neuronales profundas, proporcionando herramientas esenciales para diversas aplicaciones relacionadas con el lenguaje [16][38].

El NLP abarca varias áreas clave que trabajan en conjunto para procesar el lenguaje. Entre ellas se encuentra el análisis léxico, que identifica palabras y sus características como raíces, prefijos y sufijos. Asimismo, el análisis sintáctico permite comprender la estructura gramatical de las oraciones, mientras que el análisis semántico se enfoca en el significado de las palabras y frases dentro de su contexto. Finalmente, el análisis pragmático se encarga de interpretar el texto considerando el contexto más amplio, como la interacción detrás de las palabras del usuario [36].

⁶URL de Mendeley: <https://www.mendeley.com/>

⁷URL de Zotero: <https://www.zotero.org/>

⁸URL de EndNote: <https://endnote.com/>

Además, el NLP incluye componentes esenciales para su funcionamiento. Por ejemplo, la tokenización divide el texto en unidades más pequeñas llamadas tokens, facilitando el análisis posterior. Otro componente importante es el etiquetado de partes del discurso (POS tagging), que clasifica las palabras según su categoría gramatical, como sustantivos, verbos o adjetivos. También destaca el reconocimiento de entidades nombradas (NER), que identifica elementos específicos como nombres propios, fechas u organizaciones dentro del texto. A esto se suma el análisis de sentimientos, el cual evalúa las emociones expresadas en el texto, y la traducción automática, que convierte el texto de un idioma a otro. Estas tareas han mejorado significativamente con la incorporación de modelos de aprendizaje profundo, que han permitido grandes avances en la comprensión y generación de lenguaje natural [63].

1.2.2. Relevancia del NLP en la Escritura Científica

En el ámbito de la escritura científica, el NLP tiene un impacto considerable, ya que facilita la creación de contenido y mejora su calidad. Una de sus aplicaciones más destacadas es la generación de resúmenes automáticos, que sintetizan información clave de documentos extensos, ahorrando tiempo y esfuerzo a los investigadores. El uso de técnicas de NLP para la automatización de resúmenes ha demostrado mejorar la eficiencia en la lectura de artículos científicos y la extracción de información relevante, permitiendo a los investigadores centrarse en el análisis crítico del contenido [3]. También desempeña un papel crucial en la corrección gramatical y estilística, ayudando a identificar y corregir los errores en tiempo real para garantizar precisión y claridad. Además, el NLP puede analizar la coherencia y organización del texto, asegurando que las ideas se presenten de forma lógica y fluida.

Otra ventaja importante del NLP en la escritura científica es su capacidad para generar contenido. Por ejemplo, puede asistir en la redacción de secciones específicas como introducciones, desarrollo o conclusiones, basándose en datos iniciales proporcionados por el usuario. Asimismo, permite personalizar el estilo del texto para adaptarlo a las necesidades del autor o a los estándares de revistas científicas, lo que simplifica el proceso de publicación. Estas aplicaciones están evolucionando con el desarrollo de modelos de lenguaje que permiten una mayor precisión en la generación de contenido académico, mejorando la estructuración y coherencia del texto generado [39].

El NLP proporciona la base para integrar funcionalidades avanzadas en un editor de texto científico y es el pilar sobre el cual se construyen tecnologías más avanzadas como los Modelos de Lenguaje a Gran Escala (LLMs).

1.3. Modelos de Lenguaje a Gran Escala (LLMs)

1.3.1. Definición y características

Los Modelos de Lenguaje a Gran Escala (LLMs, por sus siglas en inglés) representan un avance significativo en el campo de la inteligencia artificial, particularmente en el NLP. Estos modelos están diseñados para procesar, comprender y generar texto de manera coherente y contextual, imitando la fluidez y estilo del lenguaje humano.

Los LLMs son redes neuronales profundas entrenadas con vastos conjuntos de datos contextuales y miles de millones de parámetros. Este entrenamiento masivo les permite identificar patrones complejos en el lenguaje, lo que les otorga la capacidad de realizar tareas como completar frases, responder preguntas, resumir contenido, y traducir. Modelos como GPT (*Generative Pre-trained Transformer*)[7] y BERT (*Bi-directional Encoder Representations from Transformers*)[18] son ejemplos destacados que han demostrado su versatilidad en aplicaciones prácticas [32].

1.3.2. Aplicaciones de los LLMs en la escritura científica

Una de las características más relevantes de los LLMs es su capacidad para generar textos de alta calidad. Estos modelos no solo pueden imitar estilos de escritura específicos, sino también adaptarse a diferentes contextos, como la escritura científica, donde la precisión y el rigor son esenciales[41]. Por ejemplo, un LLM puede generar una introducción académica basada en un conjunto de datos iniciales o sugerir mejoras estilísticas en un manuscrito. Esto hace que sean herramientas ideales para complementar los esfuerzos de los investigadores. Como lo menciona Cohan et al. (2020), los modelos de lenguaje como SPECTER pueden utilizar información proveniente de citas académicas para generar representaciones de documentos que ayuden a mejorar la escritura científica, facilitando la redacción coherente y el análisis del contenido en un artículo de investigación [13].

Además de su capacidad para generar texto, los LLMs sobresalen en tareas relacionadas con la mejora del contenido. Estos modelos pueden identificar y corregir errores gramaticales y estilísticos en tiempo real, proporcionando retroalimentación inmediata al autor [15]. También pueden analizar la coherencia lógica de un texto, señalando inconsistencias y sugiriendo formas de reorganizar las ideas para mejorar la claridad del mismo [48]. Según el informe técnico de OpenAI (2023), GPT-4 ha demostrado una notable capacidad para revisar y optimizar textos científicos, permitiendo la reestructuración de argumentos y el refinamiento del lenguaje para lograr mayor claridad y precisión en publicaciones académicas [44].

Asimismo, los modelos como BERT han permitido mejorar la escritura científica mediante el aprendizaje profundo de estructuras lingüísticas, ayudando a estructurar textos de manera más clara y lógica. Tenney, Das y Pavlick (2019) explican que

BERT puede redescubrir la estructura tradicional del lenguaje académico, ayudando a los autores a mejorar la cohesión de sus escritos mediante sugerencias basadas en datos previos de redacción científica [58]. Esta capacidad ha permitido a los investigadores optimizar la presentación de sus ideas, evitando ambigüedades y reforzando la argumentación en sus publicaciones.

Por último, como señala Heaven (2023), los LLMs como ChatGPT han comenzado a desempeñar un papel clave en la escritura científica, desde la redacción inicial de secciones de artículos hasta la revisión estilística y estructural. Según su análisis, los investigadores han utilizado estos modelos para generar resúmenes más precisos y mejorar la legibilidad de sus textos, lo que les permite reducir el tiempo necesario para la preparación de manuscritos científicos [27].

1.3.3. Limitaciones de los LLMs

A pesar de sus numerosas ventajas, los LLMs también presentan ciertas limitaciones. Uno de los desafíos más destacados es su dependencia del contexto proporcionado; si el contexto inicial es ambiguo o insuficiente, el modelo puede generar respuestas incorrectas o irrelevantes [50]. Esta característica limita su precisión en tareas complejas donde la interpretación depende de múltiples factores externos y del conocimiento no explícitamente incluido en su entrenamiento [4]. Además, al estar entrenados en grandes conjuntos de datos, los LLMs pueden reflejar sesgos inherentes a esos datos, lo que podría afectar la calidad e imparcialidad del contenido generado [61]. Otro aspecto clave es el alto costo computacional asociado a estos modelos. Entrenar y ejecutar LLMs requiere una cantidad significativa de recursos computacionales, lo que puede limitar su implementación en dispositivos con capacidad reducida [9]. Este problema genera barreras de acceso para investigadores y desarrolladores con menos recursos [24].

En la escritura científica, los LLMs ofrecen una solución innovadora para superar muchos de los desafíos asociados con los métodos tradicionales. Sin embargo, para aprovechar al máximo sus capacidades es necesario guiarlos mediante técnicas específicas, como el *prompt engineering* y el uso RAG. El prompt engineering permite optimizar las entradas al modelo, asegurando que las salidas sean relevantes, precisas y adaptadas a las necesidades del autor [20]. Por su parte, RAG mejora la generación de contenido al combinar la capacidad de los modelos de lenguaje con el acceso a fuentes externas de información, lo que permite generar respuestas más actualizadas, fundamentadas y contextualizadas.

1.4. Prompt Engineering

1.4.1. Definición

El *prompt engineering* es una técnica avanzada dentro del campo del NLP, y, en particular, del uso de los LLMs. Esta técnica consiste en diseñar entradas específicas y optimizadas que guíen el comportamiento de los LLMs para obtener resultados precisos, coherentes y adaptados a una tarea determinada. A medida que los LLMs han evolucionado y ganado capacidad, el diseño de prompts efectivos se ha convertido en un componente crucial para maximizar su utilidad en aplicaciones prácticas.

1.4.2. Importancia del Prompt Engineering

La interacción con un modelo de lenguaje depende en gran medida de las instrucciones que se le proporcionen a través de un prompt. Un prompt mal diseñado puede generar respuestas confusas, irrelevantes o inadecuadas, mientras que uno bien estructurado puede aprovechar al máximo las capacidades del modelo. El *prompt engineering* es esencial porque permite controlar y personalizar la salida del modelo sin necesidad de modificar directamente su estructura interna o parámetros [10].

1.4.3. Principios Fundamentales

Para diseñar prompts efectivos, es fundamental seguir ciertos principios básicos que garantizan la calidad de las respuestas generadas por el modelo [35]:

Especificidad: Los prompts deben ser claros y detallados, evitando ambigüedades. Por ejemplo, en lugar de solicitar “Escribe un resumen”, un prompt específico podría ser “Resume el siguiente texto en menos de 100 palabras, destacando los puntos principales relacionados con el tema científico”.

Contextualización: Proporcionar suficiente información en el prompt para que el modelo comprenda el contexto y genere una respuesta relevante. Por ejemplo, al redactar un artículo científico, incluir el tema, el público objetivo y los objetivos del documento.

Iteración: Los prompts deben ajustarse y pulirse basándose en las respuestas previas del modelo. Este proceso iterativo asegura que el modelo aprenda a generar respuestas más precisas y útiles.

1.4.4. Técnicas Comunes de Prompt Engineering

Existen varias estrategias que se pueden emplear para optimizar los prompts:

Zero-shot Prompting: Pedir al modelo que realice una tarea sin proporcionar ejemplos previos. Por ejemplo, “Explica las aplicaciones del NLP en la redacción científica” [31].

Few-shot Prompting: Incluir ejemplos en el prompt para que el modelo entienda mejor la tarea. Por ejemplo, “Aquí tienes dos ejemplos de introducciones científicas. Basándote en ellos, escribe una introducción sobre este tema” [31].

In-context Learning: Proporcionar un contexto detallado en el prompt para mejorar la comprensión del modelo. Por ejemplo, al solicitar un análisis, incluir antecedentes o información adicional relevante [60].

1.4.5. Aplicaciones del Prompt Engineering en la Escritura Científica

El *prompt engineering* desempeña un papel fundamental en la creación de un asistente de escritura de textos científicos, pues permite aprovechar al máximo las capacidades de los LLMs para tareas específicas del ámbito académico. Una de sus aplicaciones más relevantes es la generación de contenido especializado. Gracias a prompts cuidadosamente diseñados, los LLMs pueden redactar introducciones, capítulos, epígrafes y conclusiones que se alineen con el estilo y los estándares propios de la comunicación científica. Estos prompts pueden incluir indicaciones claras sobre el tema, el objetivo del texto y el público al que está dirigido, asegurando que la salida sea precisa y contextualizada [64][6].

Otra área clave de aplicación es la corrección y mejora de textos. Utilizando prompts enfocados en la revisión, los LLMs pueden identificar y sugerir correcciones para errores gramaticales, incoherencias estilísticas y problemas de claridad en textos científicos. Este proceso no solo facilita la producción de documentos de alta calidad, sino que también reduce la necesidad de revisiones manuales exhaustivas, ahorrando así tiempo a los autores [6].

Además, es fundamental para personalizar el estilo del texto según las necesidades del autor o los requisitos específicos de una revista académica. Mediante instrucciones detalladas el modelo puede adaptar el tono, el nivel de formalidad y la estructura del texto, asegurando que cumpla con los parámetros y expectativas de los revisores y editores.

Por último, una aplicación particularmente útil es la generación de citas y referencias bibliográficas. Con prompts adecuados, los LLMs pueden automatizar la creación de referencias en diversos formatos, como APA, IEEE o Chicago. Esto no solo simplifica el proceso de redacción, sino que también minimiza errores comunes en la elaboración de bibliografías.

1.5. Retrieval-Augmented Generation (RAG)

1.5.1. Definición

Retrieval-Augmented Generation (RAG) es un enfoque innovador en el campo del NLP que combina dos capacidades fundamentales: la recuperación de información relevante y la generación de texto contextualizado. Este método integra un sistema de recuperación de información con un modelo generativo, como un LLM, para proporcionar respuestas más precisas, completas y actualizadas [37]. A diferencia de los modelos generativos tradicionales, que basan sus respuestas únicamente en los datos utilizados durante su entrenamiento, RAG permite incorporar información externa en tiempo real, lo que amplía significativamente sus capacidades [2]. Este enfoque mejora la capacidad de los modelos para responder preguntas de manera más informada y precisa, reduciendo la alucinación de datos y aumentando la confianza en los resultados generados [42].

1.5.2. Funcionamiento

El modelo RAG opera en dos etapas principales:

Recuperación de Información (*Retrieval*): En esta etapa, el sistema consulta una base de datos o un conjunto de documentos externos para localizar la información más relevante en función de una consulta específica. Estas bases pueden ser locales o en línea, y están estructuradas en formatos accesibles para el modelo [37].

Generación de Texto (*Generation*): Una vez recuperada la información relevante, el modelo generativo utiliza esos datos como contexto para generar una respuesta. Esto asegura que las salidas del modelo estén basadas en datos actualizados y específicos, mejorando la precisión y relevancia de los textos generados [37].

1.5.3. Ventajas y Aplicaciones de RAG en la escritura científica

En el contexto de la escritura científica, RAG ofrece una solución innovadora para integrar información recuperada de documentos específicos en la generación de texto contextualizado y relevante. A diferencia de los modelos generativos tradicionales,

RAG utiliza una bibliografía o un conjunto de documentos relacionados con el tema seleccionados por el usuario para generar texto fundamentado, garantizando que el contenido esté alineado con las fuentes originales y, además, debidamente citado. Este enfoque mejora significativamente la precisión y la utilidad del texto generado, convirtiéndolo en una herramienta valiosa para autores académicos[56]. Una de las principales ventajas de RAG en este ámbito es su capacidad para contextualizar la información proporcionada. Al limitar la generación de texto a un conjunto definido de documentos, el modelo asegura que el contenido generado sea coherente con la temática y las referencias específicas incluidas en la bibliografía. Esto no solo reduce la posibilidad de introducir información irrelevante o inexacta, sino que también asegura que el texto producido esté respaldado por evidencias documentales, algo esencial en la redacción científica.

Además, RAG permite automatizar la citación. Cuando el modelo genera texto a partir de los documentos recuperados, incluye referencias directas a las fuentes relevantes, lo que simplifica el proceso de redacción y asegura que las citas estén correctamente integradas en el texto. Esta funcionalidad no solo ahorra tiempo a los autores, sino que también mejora la transparencia y la trazabilidad del contenido, contribuyendo a la calidad académica del manuscrito [56].

1.6. Revisión de propuestas computacionales para la escritura científica

1.6.1. Herramientas Actuales para la Escritura Científica

En el ámbito de la escritura o redacción científica, las herramientas más utilizadas han sido tradicionalmente procesadores de texto como Microsoft Word, Google Docs y LaTeX. Aunque estas herramientas han facilitado enormemente la redacción de documentos, presentan limitaciones significativas cuando se trata de abordar las necesidades específicas de la escritura científica.

Microsoft Word es uno de los procesadores de texto más utilizados debido a su accesibilidad y facilidad de uso. Esta herramienta ofrece un entorno intuitivo que permite la redacción básica y formatos avanzados mediante plantillas prediseñadas. Además, su integración con gestores de referencia como Zotero y EndNote simplifica la gestión de citas y bibliografías. No obstante, Word presenta limitaciones significativas en la gestión de documentos complejos, ya que puede experimentar problemas de estabilidad al manejar múltiples referencias cruzadas, gráficos o tablas extensas. Además, aunque su funcionalidad colaborativa ha mejorado con herramientas como OneDrive, sigue siendo menos eficiente para la edición simultánea comparado con plataformas como Google Docs. Según Gastel y Day (2022), Word es ideal para in-

vestigadores que prefieren una herramienta sencilla y versátil, pero es menos adecuada para disciplinas que requieren un control tipográfico riguroso o fórmulas complejas [21].

Por otro lado, **Google Docs** se ha posicionado como una solución ideal para trabajos colaborativos, especialmente en entornos académicos. Su capacidad para permitir ediciones en tiempo real lo hace altamente eficiente para equipos distribuidos geográficamente. Además, al estar basado en la nube, permite un acceso constante desde cualquier dispositivo con conexión a Internet, eliminando la necesidad de gestionar archivos. Entre sus puntos débiles destaca su dependencia de la conectividad, lo que puede limitar su utilidad en entornos con acceso inestable a la red. Asimismo, su capacidad para manejar documentos técnicos complejos es limitada, especialmente en comparación con herramientas como LaTeX. Hames (2008) destaca que Google Docs es particularmente útil para revisiones rápidas y colaboraciones en proyectos pequeños o intermedios, pero su funcionalidad es limitada para necesidades científicas más especializadas [25].

Por su parte, **LaTeX**, ampliamente reconocido por su excelencia en la tipografía científica, se ha convertido en el estándar para la escritura técnica y académica, especialmente en campos como las matemáticas, la física y la ingeniería. La principal fortaleza de LaTeX radica en su capacidad para manejar documentos extensos con alta precisión, ofreciendo control absoluto sobre el formato y la estructura del documento. A pesar de estas ventajas, presenta una pronunciada curva de aprendizaje, ya que los usuarios deben familiarizarse con un lenguaje de marcado y comandos específicos.

En el caso de **Markdown**, herramientas como Typora⁹ y Obsidian¹⁰ destacan por su simplicidad y eficiencia. Markdown utiliza etiquetas mínimas para aplicar formatos básicos, lo que lo convierte en una opción ligera y fácil de utilizar. Además, su compatibilidad con sistemas de control de versiones como Git lo hace ideal para investigadores que necesitan mantener un historial detallado de cambios. Sin embargo, su funcionalidad es limitada en comparación con LaTeX, particularmente en lo que respecta al manejo de fórmulas matemáticas complejas o la automatización de bibliografías. Heard (2022) sugiere que Markdown es una alternativa adecuada para investigadores que priorizan la productividad y no requieren formatos complejos, destacando su capacidad para exportar documentos en múltiples formatos como PDF y HTML [26].

De igual manera basado en Markdown, **Zettlr** es un editor de texto diseñado específicamente para académicos y escritores profesionales que necesitan organizar grandes cantidades de información y manejar referencias de manera eficiente. Permite a los usuarios crear contenido con un formato ligero y limpio, lo que facilita la

⁹URL de Typora: <https://typora.io/>

¹⁰URL de Obsidian: <https://obsidian.md/>

edición rápida y la compatibilidad con múltiples plataformas. Entre sus principales ventajas se destaca su integración nativa con gestores de referencias como Zotero, lo que simplifica la incorporación de citas y bibliografías y que incluye herramientas para búsquedas avanzadas y organización de notas, lo que lo convierte en una buena opción para trabajos extensos y estructurados.[14] Sin embargo, una de sus principales limitaciones es que carece de soporte avanzado para fórmulas matemáticas, lo que representa un obstáculo para disciplinas técnicas como matemáticas o física. Además, no incluye funcionalidades de inteligencia artificial para la mejora del texto o la generación automática de contenido.

Por otra parte, **Lex**¹¹ es un editor de texto basado en la web que se caracteriza por su enfoque minimalista y su integración con tecnologías de inteligencia artificial. Este editor proporciona una experiencia de escritura simplificada y libre de distracciones, ofreciendo además la capacidad de generar sugerencias de texto en tiempo real mediante el uso de modelos de lenguaje avanzados, como GPT. Su interfaz limpia y su capacidad para generar contenido predictivo resultan útiles para superar bloqueos creativos y fomentar el desarrollo eficiente de ideas [59]. Sin embargo, Lex presenta limitaciones significativas cuando se considera su aplicación en el contexto de la escritura académica. Aunque su funcionalidad de generación de texto es eficiente, carece de la personalización necesaria para ajustarse a los estándares rigurosos de la redacción científica, como el manejo de formatos específicos de citación. Asimismo, no cuenta con integración para gestores de referencias ni herramientas avanzadas para la gestión de bibliografías, lo que restringe su utilidad en proyectos académicos complejos. Estas características hacen de Lex una herramienta más adecuada para tareas de escritura creativa o la preparación de borradores preliminares, en lugar de una solución integral para la elaboración de documentos científicos.

En el contexto de herramientas más especializadas, **MathTOUCH**¹² representa una solución para abordar la complejidad de la entrada de expresiones matemáticas. Este editor de texto utiliza un algoritmo de conversión predictiva que transforma texto coloquial en expresiones matemáticas en formato bidimensional, permitiendo a los usuarios introducir fórmulas de manera más intuitiva. Esta funcionalidad elimina la necesidad de comandos sintácticos complejos, como los requeridos en LaTeX, y reduce la carga cognitiva del usuario. Además, MathTOUCH ha demostrado ser un 50 % más rápido que las interfaces estándar y ha incrementado la satisfacción de los usuarios en estudios realizados [53]. Implementado como un complemento para Moodle, esta herramienta combina accesibilidad y eficiencia al integrar funciones como visualización WYSIWYG y una interacción fluida entre texto y expresiones matemáticas.

También, existen asistentes de escritura más básicos como Grammarly y Quill-

¹¹URL de Lex: <https://lex.page/>

¹²URL de MathTOUCH: <https://mathtouch.org/en/>

bot¹³. **Grammarly** se especializa en la corrección de gramática, ortografía y puntuación. Esta herramienta identifica eficientemente errores en tiempo real y cuenta con capacidades adicionales como la detección de plagio. Además, su integración con plataformas populares como Microsoft Word, Google Docs, correos electrónicos y navegadores web lo convierte en un recurso versátil para estudiantes, profesionales y escritores en general. Sin embargo, su mayor limitación es que sus funciones avanzadas están restringidas a su versión premium, lo que la hace poco accesible.

Por otra parte, **Quillbot** se destaca por su enfoque en la reescritura y síntesis de textos, ofreciendo funciones como parafraseo, resumen y generación de contenido. Esta herramienta es ideal para reformular ideas, condensar información y mejorar la fluidez del texto para usuarios que buscan expresar ideas de manera más creativa. Entre sus limitaciones, se pueden mencionar la dependencia de los textos de entrada para generar resultados precisos y que en ocasiones un parafraseo puede alterar el significado original del texto. Además, sus capacidades de corrección gramatical son más básicas en comparación con Grammarly.

Por otro lado, **Effidit** es un asistente de escritura impulsado por inteligencia artificial que ofrece un enfoque integral para mejorar la calidad y eficiencia de la redacción académica en inglés y chino. A diferencia de Grammarly o Quillbot, Effidit amplía las capacidades típicas de los asistentes al integrar cinco módulos principales: completado de texto, corrección de errores, pulido de texto, generación de oraciones a partir de palabras claves (*Keywords-to-Sentences*, K2S) y métodos de entrada en la nube (Cloud IME). Su módulo de completado de texto destaca por combinar métodos de generación y recuperación, permitiendo tanto la creación de frases como de oraciones completas, además de ofrecer estilos específicos para escritura creativa en chino, como ciencia ficción y fantasía. El módulo de corrección de errores sobresale en la detección y corrección de errores de gramática, ortografía y sintaxis, especialmente en chino. En cuanto al pulido de texto, Effidit permite reformular oraciones manteniendo su significado, pulir frases contextualizadas y expandir texto, con la particularidad de soportar conversiones entre chino clásico y moderno. El módulo K2S genera oraciones relacionadas semánticamente a partir de palabras claves, utilizando tanto recuperación basada en corpus como generación con modelos avanzados de lenguaje. Por último, su Cloud IME proporciona sugerencias instantáneas contextuales de palabras y frases durante la escritura. Sin embargo, algo que lo hace poco atractivo para los usuarios globales es que gran parte de sus innovaciones están orientadas a usuarios chinos, limitando su aplicación a otros mercados; por ejemplo, los estilos creativos solo están disponibles en chino, lo que restringe su potencial en inglés u otros idiomas [52].

Finalmente, herramientas basadas en inteligencia artificial como **ChatGPT**¹⁴ y

¹³URL de Quillbot: <https://quillbot.com/>

¹⁴URL de ChatGPT: <https://chatgpt.com/>

Jasper AI¹⁵, han comenzado a transformar el panorama de la escritura científica. Estas plataformas son particularmente útiles para acelerar tareas rutinarias, como la generación de borradores, la corrección gramatical y las sugerencias estilísticas. Sin embargo, su funcionalidad es limitada en términos de especialización, ya que no son herramientas diseñadas para la redacción. Según Dergaa et al (2023), su mayor aporte radica en la optimización de procesos [17].

A pesar de estos avances, existe una escasez de herramientas accesibles que integren de forma nativa tecnologías de inteligencia artificial que permitan asistir en la redacción científica mediante sugerencias contextuales o mejoras estilísticas en tiempo real. Este vacío tecnológico constituye una de las principales motivaciones de esta investigación.

1.6.2. Identificación de Vacíos Tecnológicos

A partir de esta revisión, se identifican varios vacíos que justifican la propuesta de esta investigación:

- Falta de herramientas que integren LLMs para la redacción científica.
- Ausencia de funcionalidades específicas para generar contenido adaptado a formatos académicos.
- Escasez de soluciones que combinen capacidades de corrección gramatical y estilística con generación de texto contextualizado en tiempo real.
- Necesidad de plataformas que ofrezcan una experiencia de usuario accesible y eficiente, sin requerir conocimientos técnicos avanzados.

Es por eso que la propuesta de esta investigación se posiciona como una solución para abordar estos vacíos. Mediante la integración de LLMs optimizados a través de técnicas de *prompt engineering* y RAG, el prototipo que se desarrollará permitirá a los usuarios generar, editar y mejorar textos científicos de manera más eficiente y precisa. Esta herramienta no solo facilitará la redacción, sino que hará que el proceso de materializar la investigación sea más eficiente.

En el próximo capítulo, se describirá el diseño lógico del prototipo.

¹⁵URL de Jasper AI: <https://www.jasper.ai/>

Capítulo 2

Diseño

El diseño del prototipo se estructura en torno a dos componentes fundamentales. El primero de ellos es el componente visual, el cual desempeña un papel central en la interacción del usuario con la plataforma. Este componente abarca la edición de texto, la importación y exportación de documentos, así como un conjunto de herramientas de asistencia interactivas basadas en LLMs. El segundo es el componente API, cuya función principal es gestionar y ejecutar las peticiones generadas desde el componente visual. Este módulo actúa como el núcleo, procesando la información enviada por el usuario, gestionando los modelos de lenguaje (LLM) y asegurando la integración con fuentes externas necesarias para el enriquecimiento del contenido (Fig. 2.1).

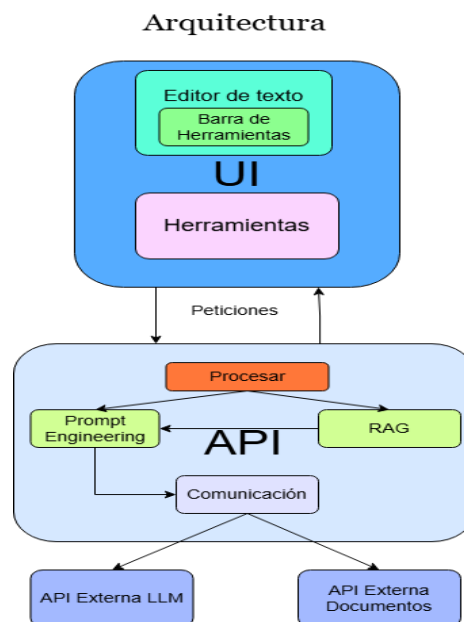


Figura 2.1: Arquitectura

2.1. Componente Visual

El componente visual del prototipo está compuesto por una interfaz gráfica cuya estructura principal incluye un editor de texto y un conjunto de herramientas que interactúan con la API (Fig. 2.2).

2.1.1. Estructura del Componente Visual

A continuación, se describen los principales componentes que conforman la interfaz:

- **Editor de Texto:** Elemento central de la interfaz donde se presentan y manipulan los contenidos textuales. Diseñado para admitir interacciones del usuario relacionadas con la edición de texto.
- **Barra de Herramientas:** Conjunto de elementos necesarios para realizar las acciones de edición de texto tanto básicas como cambiar la fuente o complejas como insertar una ecuación, organizados en una disposición accesible para facilitar su uso.
- **Barra de navegación de herramientas:** Sección que agrupa múltiples funcionalidades adicionales. Permite la interacción con la API para ejecutar acciones específicas.

Interacción del Componente Visual con la API

El diseño lógico contempla una comunicación estructurada entre la interfaz gráfica y la API, permitiendo que las herramientas disponibles en la interfaz operen a través de solicitudes y respuestas gestionadas por la API.

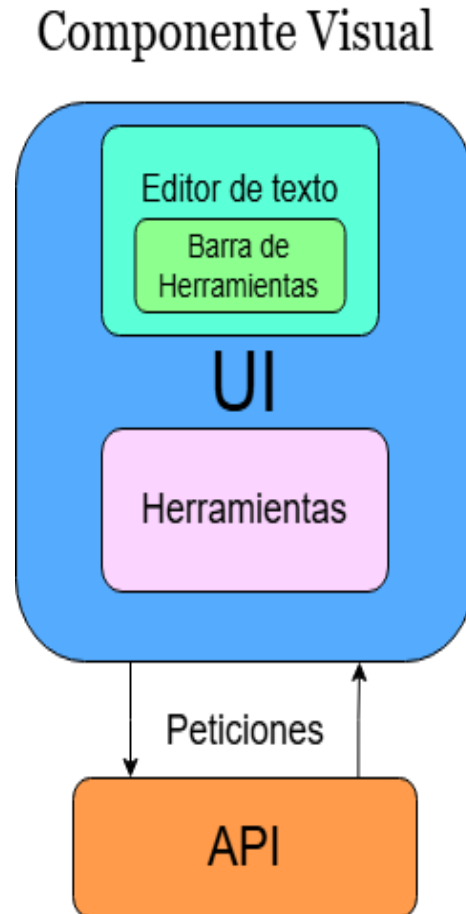


Figura 2.2: Arquitectura de la interfaz gráfica

2.2. Componente API

La API expone los servicios y funcionalidades de la aplicación, permitiendo la comunicación entre la interfaz gráfica y los servicios externos a través de un conjunto de endpoints definidos. Su arquitectura sigue un modelo RESTful¹ basado en la gestión de peticiones, procesamiento de datos y comunicación con otras API (Fig. 2.3).

¹API RESTful es una interfaz de programación de aplicaciones (API) que sigue los principios de REST (Representational State Transfer), un estilo de arquitectura para la comunicación entre sistemas en la web.

2.2.1. Estructura del Componente API

Aquí se muestra la estructura de la API:

- **Gestión de Peticiones:** Recibe las solicitudes provenientes de la interfaz gráfica. Estructura y organiza los datos para su procesamiento.
- **Procesamiento de Datos:** Interpreta y transforma la información recibida según los requerimientos de la aplicación. Prepara los datos para su posterior consulta o generación de contenido.
- **Núcleo:** Responsable de procesar solicitudes, generar información relevante y gestionar las operaciones fundamentales del programa, asegurando su correcto funcionamiento y la toma de decisiones clave. Para ello, emplea:
 - **Prompt Engineering:** Estructura dinámicamente las consultas para optimizar la interacción con modelos de lenguaje, asegurando respuestas precisas y contextualizadas.
 - **RAG:** Recupera información de bases de datos antes de generar contenido, proporcionando contexto adicional para mejorar la calidad de las respuestas.
- **Comunicación con APIs Externas:** Establece conexión con proveedores de modelos de lenguaje (LLM) como OpenAI, Gemini o Hugging Face para recuperar documentación o complementar la generación de contenido. Gestiona el envío y recepción de datos asegurando que la información procesada sea coherente con los objetivos de la aplicación.

2.2.2. Flujo de Comunicación

El diseño lógico del núcleo establece un flujo estructurado de información en el que las peticiones siguen un ciclo definido:

- La interfaz gráfica envía una solicitud a la API del núcleo.
- La API procesa la solicitud y determina la acción requerida.
- Si es necesario, se consulta a una API externa para obtener documentación o generar contenido.
- Los datos obtenidos se estructuran y se envían a la interfaz gráfica.

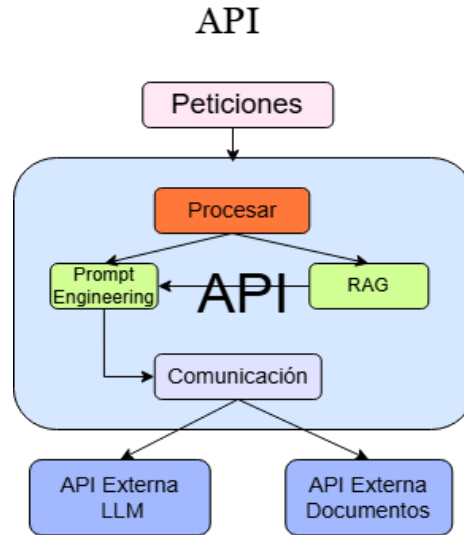


Figura 2.3: Arquitectura de la API

2.2.3. Núcleo

El núcleo de la aplicación constituye la base de su funcionamiento, integrando los módulos esenciales que permiten su operatividad. En esta sección, se describen los componentes principales que lo conforman, incluyendo los módulos de *Prompt Engineering* y RAG, y cómo estos se articulan para garantizar un desempeño eficiente del sistema.

Prompt Engineering

El módulo de *Prompt Engineering* se encarga de procesar las peticiones que requieren la construcción de *prompts* estructurados. Su diseño está basado en una arquitectura jerárquica que organiza los distintos tipos de prompts dentro de una conversación estructurada (Fig. 2.4).

Estructura del Módulo de Prompt Engineering:

Aquí se muestra la estructura del módulo:

- **BasePrompt:** Es la estructura base a partir de la cual se definen los distintos tipos de prompts. Define reglas de redacción académica, como evitar lenguaje coloquial, mejorar la claridad y usar términos técnicos adecuados.
- **Tipos de Prompts:** A partir de BasePrompt, se derivan tres principales tipos de prompts:

- **UserPrompt:** Representa las entradas del usuario dentro de la conversación. Se puede extender para realizar cualquier tipo de tarea específica como resumir, extender o modificar cierta parte del documento y aplicar diferentes acciones sobre la forma en que se escribe el texto, por ejemplo, precisión terminológica, estructura y cohesión.
 - **AssistantPrompt:** Define las respuestas generadas dentro del flujo de interacción.
 - **SystemPrompt:** Contiene instrucciones o configuraciones previas que guían la generación de respuestas. Se puede extender para crear distintas directrices académicas, por ejemplo, uso de voz impersonal, citas en formato específico, coherencia en la terminología específica de la disciplina.
- **Construcción de la Conversación:** Dependiendo de la petición recibida, se generan distintos tipos de prompts. Los prompts creados se organizan en una estructura de conversación, preservando la continuidad de la interacción.
 - **Contexto:** Algunos prompts necesitan de un contexto para ejecutarse. En algunas ocasiones dicho contexto puede ser muy grande, excediendo el límite de tokens del modelo. Para tratar este inconveniente, el contexto es particionado en fragmentos de forma que entren dentro del límite de tokens del modelo. Luego, los prompts se ejecutan utilizando todos los fragmentos o una parte de ellos como contexto de forma individual.

Este módulo está diseñado para asistir en la redacción, corrección y mejora de textos científicos, asegurando que el lenguaje, la estructura y la coherencia cumplan con los estándares académicos. Facilita la generación de prompts específicos para mejorar claridad, precisión terminológica y estructura argumentativa en artículos de investigación. Para lograr esto, se extendieron los distintos, se crearon *UserPrompts* especializados en tareas específicas y múltiples *SystemPrompts* para garantizar diferentes estilos gramaticales y académicos; y coherencia.

Extensibilidad: El usuario puede personalizar el comportamiento del módulo ajustando los *SystemPrompts* para adaptarse a distintos estilos y formatos de publicación. Además, puede definir reglas específicas para distintos campos científicos, garantizando un nivel de personalización alto para cada disciplina.

Módulo de Prompt Engineering

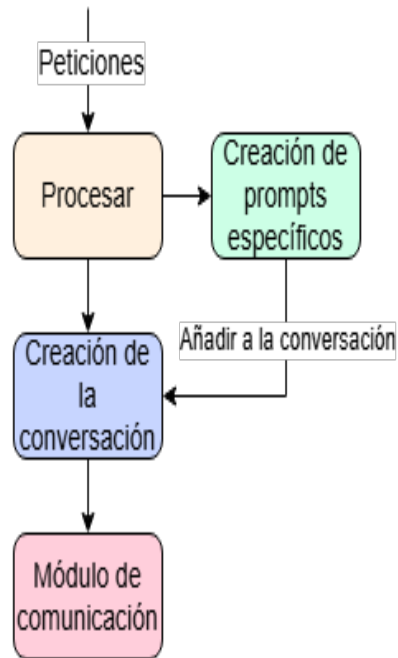


Figura 2.4: Diseño del módulo de prompt engineering

Flujo de Procesamiento

El módulo opera siguiendo un flujo lógico definido (Fig. 2.5):

- Se recibe una petición que requiere la generación de *prompts*.
- Se determina qué tipo de *prompt* es necesario en función del contexto.
- Los *prompts* generados se agregan a la estructura de conversación.
- La conversación se prepara para ser enviada al siguiente módulo de procesamiento.

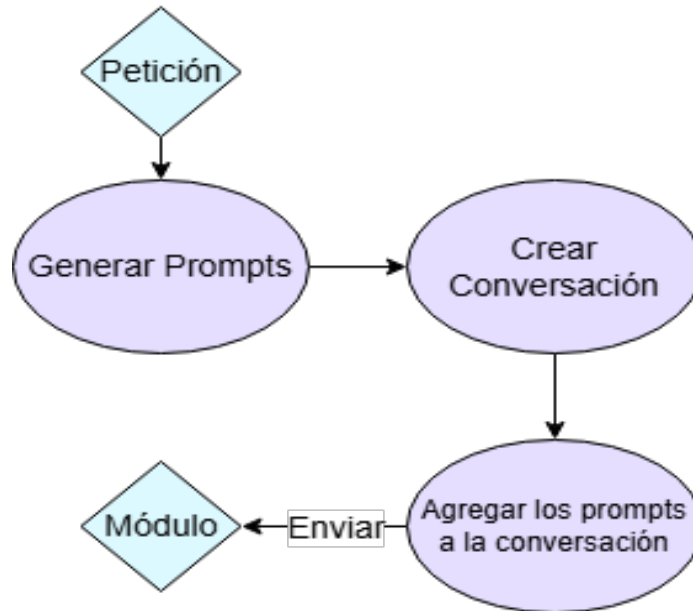


Figura 2.5: Diseño del flujo lógico del módulo de prompt engineering

RAG

El módulo RAG se centra en la recuperación y gestión de información a partir de la bibliografía seleccionada por el usuario. Para ello, procesa los documentos de referencia, dividiéndolos en fragmentos que luego almacena y organiza en una base de datos vectorial. Cuando el usuario realiza una consulta, el sistema identifica y recupera los fragmentos más relevantes, permitiendo integrar citas y referencias de manera automática. Al estar basado casi exclusivamente en la bibliografía proporcionada, este módulo facilita la incorporación de información precisa y fundamentada, mejorando la coherencia y solidez argumentativa en la redacción de textos científicos (Fig. 2.6).

Estructura del Módulo de RAG:

Aquí se muestra la estructura del módulo:

- **Base de Datos Vectorial:** Almacena y organiza los documentos seleccionados por el usuario para su bibliografía en forma de representaciones vectoriales. Esto permite realizar búsquedas semánticas y recuperar fragmentos de información relevantes para la redacción científica.
- **Segmentación e Indexación de Documentos:** Al agregar un nuevo documento de referencia, este se particiona en fragmentos de tamaño específico. Cada fragmento se almacena de manera independiente en la base de datos vectorial.

- **Recuperación de Información:** Ante una solicitud del usuario, se realiza una búsqueda en la base de datos vectorial y se extraen los fragmentos más relevantes de los documentos bibliográficos. Esto permite acceder a información fundamentada y contextualizada dentro del marco de referencia establecido por el usuario.

Generación de la Petición para el Próximo Módulo: Los fragmentos recuperados son utilizados para construir una nueva solicitud estructurada. A través del módulo de *Prompt Engineering*, se extienden los tipos principales de prompts basados específicamente en la tarea a realizar (por ejemplo, generar cita en línea), para generar una petición optimizada para su procesamiento en el siguiente módulo.

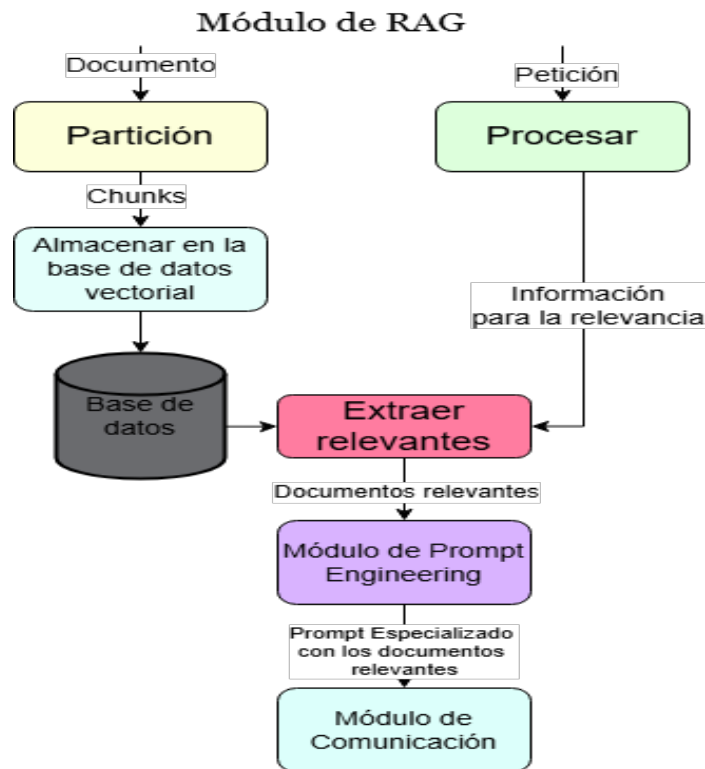


Figura 2.6: Módulo de RAG

Flujo de Procesamiento

El módulo opera bajo un flujo lógico(Fig. 2.7) en el que :

- Se recibe un documento para su almacenamiento.
- Se eliminan los caracteres que representen imágenes u otro tipo de texto enriquecido, luego se genera la cita en línea del documento.
- La información del documento es almacenada junto a su cita en línea.
- Se particiona el documento en fragmentos para su posterior almacenamiento.
- Los fragmentos son almacenados junto a su correspondiente cita en línea en la base de datos vectorial.
- Ante una solicitud, se extraen los fragmentos más relevantes.
- Se genera una nueva petición basada en la información recuperada.

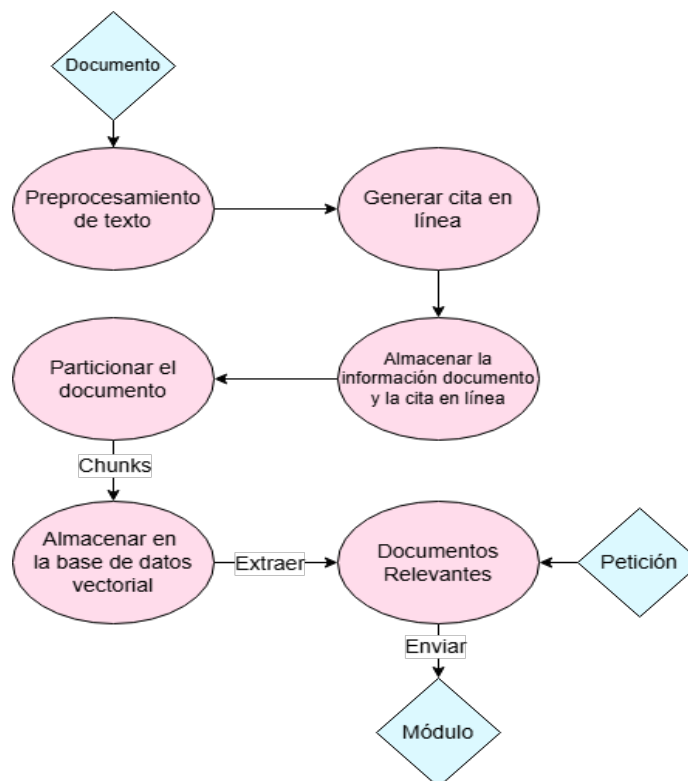


Figura 2.7: Diseño del flujo lógico del módulo de RAG.

Herramientas

Haciendo uso de los módulos anteriores, se desarrollaron distintas herramientas accesibles desde una interfaz gráfica, donde el usuario puede utilizar estas funciones para la asistencia con la escritura científica:

- Resumir: Se encarga de resumir un texto seleccionado o extraído por el usuario utilizando el módulo de *prompt engineering*.
- Extender: Se encarga de ampliar un texto seleccionado o extraído por el usuario, haciendo uso de los módulos de *prompt engineering* y RAG. Para así, ampliar el texto teniendo en cuenta la bibliografía del usuario.
- Constructor de Prompts: Herramienta para construir prompts personalizados por el usuario, los cuales son almacenados para su posterior uso.
- Chequeo: Herramienta que dado un conjunto de acciones seleccionadas por el usuario, ejecuta dichas acciones sobre el texto, haciendo uso de *prompt engineering*.
- Chat: Herramienta para interactuar en tiempo real con el modelo de lenguaje elegido.
- Extracción y Construcción de Bibliografía: Se basa en un buscador donde se interactúa con una API externa para obtener los documentos relacionados con la búsqueda. Luego el usuario puede seleccionar los documentos para añadirlos a su bibliografía.
- Generador de Bibliografía: Dado los documentos seleccionados por el usuario, se genera en el documento la bibliografía.
- Generador de Referencias: Dado los datos proporcionados y el tipo de referencia que se quiera generar, se genera la cita en línea y la referencia, haciendo uso de *prompt engineering*.

Capítulo 3

Implementación

En este capítulo se aborda la implementación de cada componente que conforma el diseño descrito en el anterior capítulo.

3.1. Componente Visual

Para la construcción de esta interfaz gráfica se utilizó React con Javascript y para el aspecto visual TailwindCss¹, se seleccionaron estas herramientas por tener gran experiencia con el trabajo con ellas.

3.1.1. Edición de Texto

Para la construcción de este módulo de la parte visual se necesitó un framework para edición de textos. Se evaluaron varias opciones, Lexical², Slate³, Quill⁴ y ProseMirror⁵. Finalmente, se escogió Lexical debido a las siguientes ventajas comparativas:

- **Eficiencia y rapidez:** A diferencia de otros editores como Quill o Slate, que utilizan estructuras de datos basadas en modelos de documentos tradicionales, Lexical emplea un sistema basado en nodos con actualizaciones optimizadas. Esto permite evitar renders innecesarios y mejora el rendimiento, especialmente en aplicaciones con contenido dinámico o grandes volúmenes de texto.
- **Arquitectura modular:** Mientras que Quill y ProseMirror ofrecen una gran cantidad de funcionalidades predefinidas, pueden resultar más pesados si solo se

¹URL de TailwindCss: <https://tailwindcss.com/>

²URL de Lexical: <https://lexical.dev/>

³Documentación de Slate: <https://docs.slatejs.org/>

⁴Documentación de Quill: <https://quilljs.com/docs/quickstart>

⁵Documentación de ProseMirror: <https://prosemirror.net/docs/>

requieren ciertas características específicas. Lexical adopta un enfoque modular que permite agregar únicamente los módulos necesarios, optimizando el tamaño y rendimiento de la aplicación.

- **Extensibilidad mediante plugins:** Aunque otros editores como ProseMirror también permiten la creación de plugins, Lexical simplifica este proceso gracias a su diseño basado en React y su API amigable. Esto facilita la personalización y la integración con otras tecnologías del ecosistema moderno.
- **Soporte y modernidad:** Lexical es desarrollado por Meta (Facebook), lo que le proporciona un respaldo sólido y actualizaciones frecuentes. En contraste, Slate y Quill han tenido periodos de desarrollo más irregulares y menos documentación reciente.

El módulo de edición presenta dos partes. La entrada de textos y la barra de herramientas. Para la implementación de elementos en Lexical se crean *plugins*, que pueden representar elementos complejos como puede ser una barra de herramientas o sencillos como algún tipo de evento. Para la implementación de estos se crea un comando⁶, haciendo uso del método *createCommand* predefinido en Lexical, en el cual se le pasa el identificador como argumento y luego son registrados utilizando el método *registerCommand* con su lógica específica. En la construcción del prototipo se crearon varios *plugins* los cuales son:

- ***PageBreakPlugin*:** Representa la división entre dos páginas del documento.
- ***DeleteNodePlugin*:** Representa un evento el cual permite la eliminación de nodos completos del documento(listas, títulos, subtítulos, divisor de páginas, ecuaciones).
- ***ToolbarPlugin*:** Representa la barra de herramientas, en la cual se encuentran las herramientas de edición del texto.
- ***CustomCommandsPlugin*:** Representa un *wrapper*⁷ que permite añadir múltiples eventos simples como: modificar el texto a mayúsculas, a minúsculas, capitalizarlo y alinearlos al inicio como al final.
- ***EquationPlugin*:** Representa la herramienta para escribir las ecuaciones en formato Latex (Fig. 3.2).

⁶Identificador de un evento

⁷Un wrapper es un programa o código que encapsula y facilita el uso de otros componentes del programa

Para el trabajo con el texto Lexical presenta múltiples opciones, la principal sería *useLexicalComposerContext* porque permite obtener el objeto del estado actual del editor. Mediante sus métodos como *.read* y *.update* se puede tanto leer como modificar el texto y otro como *dispatchCommand* que permite ejecutar comandos específicos sobre la parte del texto seleccionada, existen predefinidos y se pueden crear nuevos personalizados, mediante el método *registerCommand*.

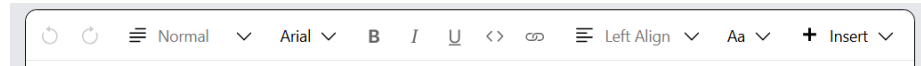


Figura 3.1: Barra de herramientas

En la barra de herramientas se implementaron múltiples tipos de opciones básicas para la edición del texto como:

- Las opciones de formato y personalización del texto (tamaño de fuente y de texto (mayúscula y minúscula), negrita, cursiva, subrayado, tachado, subíndice y superíndice). Para la implementación de estas opciones de negrita, cursiva, subrayado, tachado, subíndice, superíndice se hizo uso de los comandos predefinidos por Lexical, por ejemplo para subíndice se ejecuta

```
.dispatchCommand(FORMAT_TEXT_COMMAND, subscript),
```

luego para las otras opciones se crearon comandos específicos porque no existían predefinidos que los ejecutaran, por ejemplo para mayúscula se registra el comando

```
.registerCommand(TOGGLE_UPPERCASE_COMMAND, () => (código)).
```

- Alineación del texto, adaptándose a los requerimientos específicos de cada usuario. Para la implementación de todas las alineaciones se hizo uso de los comandos predefinidos por Lexical, por ejemplo para *Left Align*, se ejecuta

```
.dispatchCommand(FORMAT_ELEMENT_COMMAND, left)
```

.

- Creación de títulos, subtítulos que permitirá estructurar y jerarquizar el contenido, incluyendo también formato de listas tanto con orden como sin orden. Para la implementación de estas opciones se utilizaron comandos predefinidos por Lexical, por ejemplo para la creación de una lista ordenada se ejecuta,

```
.dispatchCommand(INSERT_ORDERED_LIST_COMMAND)
```

.

- Inserción de elementos enriquecidos expande las capacidades del sistema al permitir la inclusión de ecuaciones, códigos, hipervínculos y citas. Para la implementación de elementos mas complejos de debe crear *plugins*. Por ejemplo para las ecuaciones al estilo latex, se hizo uso de una biblioteca llamada *katex*⁸, en la cual con la expresión matemática de latex se obtiene el resultado y se renderiza en el texto (Fig. 3.2).

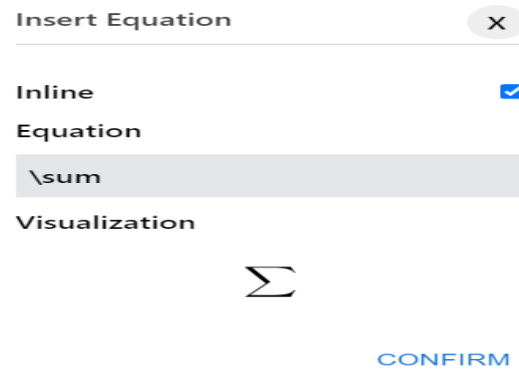


Figura 3.2: Insertar ecuación.

3.1.2. Importación y exportación de documentos

Importación

Para la importación de documentos son permitidos solamente dos tipos de archivos *.md* y *.lexical*. Se toman estos dos como únicos tipos, primero *.lexical* representa el formato interno del editor de texto por lo que preserva la estructura y el formato del contenido de manera óptima dentro del entorno de edición. y *.md* porque es fácil de analizar y convertir a otros formatos como *HTML*, *PDF* o *LaTeX* sin pérdida de información.

Exportación

Se desarrolló la funcionalidad para la exportación de tres formatos de documentos externos:

Markdown (*.md*): Para la exportación en *.md* se utilizó un método incluido en la biblioteca de lexical *convertToMarkdownString*, el cual lee lo escrito en el editor y devuelve la cadena de texto de lo leído en el documento en formato markdown y luego es exportado a un archivo de su propio formato.

⁸URL de katex: <https://katex.org/>

Microsoft Word (.docx): Para la exportación en *.docx* se convirtió primeramente el documento a formato markdown con el método descrito anteriormente. Luego se mapearon todos los tipos de formato del markdown a Word utilizando una biblioteca llamada *docx*⁹. Se escogió por su facilidad a la hora de mapear los elementos.

Latex (.tex): Para la exportación en *.tex* se convirtió el documento a formato markdown con el método descrito en su sección. Luego con se convirtió a *.tex* utilizando la biblioteca *markdown-it*¹⁰.

3.2. Componente API

Para la construcción de esta API se utilizó Python, haciendo uso de FastAPI¹¹. Se escogieron Python y FastAPI por la experiencia que se tiene utilizando dichas tecnologías.

Al construir la API se optó por una arquitectura por capas para tener separadas las funciones de cada módulo de la API. Se implementaron patrones de diseño que complementan dicha estructura, como el *Factory Pattern* que se utilizó para crear instancias de objetos dependiendo de los requisitos de cada caso y para generar dinámicamente diferentes estrategias de interacción con APIs externas según las configuraciones o el tipo de solicitud. Por ejemplo, al utilizar una clase para instanciar diferentes conectores o adaptadores según la API externa que se vaya a utilizar, como el modelo de lenguaje o la base de datos científica. Además, el *Strategy Pattern* que permitió definir una familia de comportamientos y seleccionarlos dinámicamente según el contexto. Se refleja al implementar estrategias separadas para manejar solicitudes relacionadas con *prompt engineering* y RAG, asegurando que cada una use su propio enfoque especializado. De igual manera, el *Singleton Pattern* que garantiza que una clase tenga una única instancia en toda la aplicación y proporciona un punto de acceso global a ella. Se implementa al crear un objeto persistente que representa la conexión con la base de datos vectorial.

3.2.1. Procesamiento

Prompts: Para gestionar la interacción con los modelos de lenguaje, se desarrolló una arquitectura basada en clases. Se creó una clase principal, denominada Chat, que encapsula los mensajes entre el usuario, el asistente y el sistema, la cual permite añadir los *prompts* mediante el método *add_prompt* y devolver la conversación en un arreglo mediante el método *get*. Los mensajes están tipificados en tres categorías

⁹URL de docx: <https://docx.js.org/>

¹⁰URL de markdown-it: <https://github.com/markdown-it/markdown-it>

¹¹URL de FastAPI: <https://fastapi.tiangolo.com/>

principales: `UserPrompt`, `SystemPrompt` y `AssistantPrompt`. Estas clases son extensibles, lo que permite crear prompts más específicos según las necesidades del usuario, garantizando su escalabilidad y flexibilidad.

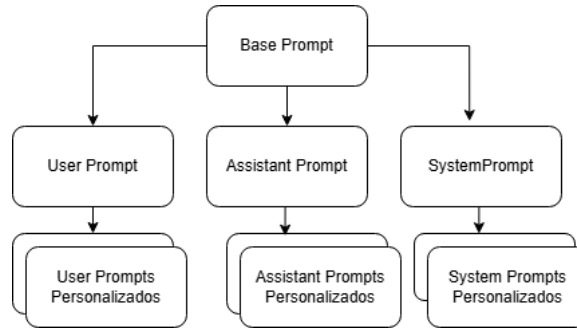


Figura 3.3: Jerarquía de prompts

Prompts Predefinidos

Se crearon prompts específicos para llevar a cabo tareas determinadas, los cuales se muestran a continuación:

UserPrompts:

- ***UserSummarizePrompt***: Habla de que el modelo resuma el texto suministrado, siguiendo un estilo, estructura y sintaxis especificadas para la escritura científica y que solo responda con el texto resumido.
- ***UserTopicPrompt***: Habla de que solo mencione la idea central del texto suministrado, sin información adicional.
- ***UserLanguageDetectPrompt***: Habla de que el modelo detecte el idioma del texto suministrado y que solo responda con el idioma.
- ***UserCheckPrompt***: Habla de realizar las acciones suministradas sobre el texto recibido, como la corrección errores ortográficos y gramaticales, omisión de palabras innecesarias, sustitución de palabras sobreutilizadas, simplificación de oraciones complejas y eliminación de palabras repetidas, y que solo devuelva el texto corregido.
- ***UserCheckCitationPrompt***: Habla de realizar las acciones suministradas sobre el texto recibido, como la corrección errores ortográficos y gramaticales,

omisión de palabras innecesarias, sustitución de palabras sobreutilizadas, simplificación de oraciones complejas, eliminación de palabras repetidas e identificación de afirmaciones que necesitan evidencias, y que solo devuelva el texto corregido.

- ***UserExtendTextInLineCitations***: Habla de que se extienda el texto suministrado siguiendo un estilo, estructura y sintaxis especificadas para la escritura científica, haciendo uso de un contexto (documentos relevantes) y que lo generado por el modelo tenga cita en línea basado en él y que solo responda con el texto extendido.
- ***UserArXivInLineReferencePrompt***: Habla de que genere la cita en línea con los datos especificados y que solo responda con la cita.
- ***UserBookReferencePrompt***: Habla de que genere la cita en línea y la referencia bibliográfica con los datos suministradas del libro y que responda solo con la cita en el texto y la referencia de una forma en específica.
- ***UserArticleReferencePrompt***: Habla de que genere la cita en línea y la referencia bibliográfica con los datos suministradas del artículo y que responda solo con la cita en el texto y la referencia de una forma en específica.
- ***UserWebsiteReferencePrompt***: Habla de que genere la cita en línea y la referencia bibliográfica con los datos suministradas del sitio web y que responda solo con la cita en el texto y la referencia de una forma en específica.
- ***UserReportReferencePrompt***: Habla de que genere la cita en línea y la referencia bibliográfica con los datos suministradas del informe y que responda solo con la cita en el texto y la referencia de una forma en específica.

SystemPrompts:

- ***SystemGenerateTextPrompt***: Indica al modelo que las respuestas generadas deben tener unas características específicas, basadas en la escritura científica. Como son la claridad y la precisión: que sería ser directo pero explicativo donde cada oración contribuya al contexto general; la cohesión: que representa la relación entre párrafo y cada idea entre sí; la coherencia: donde se mantenga una estructura clara, siguiendo un orden de lo que se quiera comentar.
- ***SystemReturnResultPrompt***: Indica al modelo que solo responda con lo que se pide, sin explicaciones previas.

Módulo de RAG: Para la implementación de este módulo, se utilizó *LangChain*¹², se escogió por su facilidad de integración con múltiples LLMs sin necesidad de grandes ajustes, por sus *chains* (cadenas de procesamiento) que combinan múltiples pasos (por ejemplo, recuperar documentos + generar respuesta) por su soporte nativo para embeddings¹³ y búsqueda semántica, integración con bases vectoriales como FAISS¹⁴, ChromaDB¹⁵. *LangChain* permite generar embeddings mediante el uso de modelos avanzados como los de OpenAI, Hugging Face o Cohere. Estos vectores son esenciales para identificar similitudes entre documentos y realizar búsquedas eficaces dentro de una base de datos vectorial.

En el caso de la presente investigación, se utilizó un modelo de embeddings base proporcionado por Hugging Face, */all-MiniLM-L6-v2* para convertir los documentos en representaciones vectoriales y se escogió Chroma para gestionar el almacenamiento local y realizar búsquedas rápidas y precisas dentro de la base de datos vectorial. Para seleccionar un modelo de embeddings diferente en *HuggingFaceEmbeddings*, es necesario proporcionar el nombre del modelo disponible en Hugging Face como argumento al constructor de la clase que lo gestiona, en este caso sería *ReferenceStoreContext*.

Las interacciones con la base de datos vectorial se hace a través de un *wrapper* con la siguiente estructura:

- *add_document* Su función es añadir el documento a la base de datos vectorial. Para ello el documento es particionado en fragmentos, luego se crean los ids de los fragmentos basado en el id del documento original, para luego identificarlos. Después son añadidos a la base de datos vectorial.
- *remove_document* Su función es eliminar el documento de la base de datos vectorial. Para ello, se elimina por los ids, al ser basado en el documento original son fácil de identificar.

Comunicación con los LLMs: La comunicación con los LLMs es completamente online, se utilizan tanto modelos de OpenAI como de Gemini. Para la comunicación

¹²LangChain es un framework de código abierto diseñado para facilitar la integración de modelos de lenguaje grande (LLMs) con fuentes de datos externas, bases de conocimiento y flujos de trabajo avanzados. Permite construir aplicaciones inteligentes mediante cadenas de procesamiento, recuperación de información con embeddings y automatización de agentes.

¹³Un embedding es una representación numérica de datos, generalmente en forma de un vector en un espacio de múltiples dimensiones

¹⁴FAISS (Facebook AI Similarity Search): Es una biblioteca de código abierto desarrollada por Meta AI para búsqueda eficiente de vectores de alta dimensión. Se usa principalmente en tareas de búsqueda semántica, recuperación de información y bases de datos vectoriales, permitiendo búsquedas rápidas en grandes volúmenes de datos.

¹⁵Es una base de datos vectorial optimizada para gestionar y buscar embeddings de texto. Se integra fácilmente con modelos de lenguaje y frameworks como LangChain, permitiendo almacenar, indexar y recuperar información de manera eficiente en aplicaciones de IA generativa.

se utiliza la biblioteca *openai* que interactúa con la API de OpenAI, la cual se incluyó un *wrapper* para que el cambio entre modelos y su comunicación sea sencillo, usando el método *post* de la clase *wrapper*. Para seleccionar el modelo a utilizar, el *wrapper* actúa como un recurso persistente. De este modo cuando el usuario en el apartado de ajustes de la interfaz gráfica elige el modelo a utilizar, se envía una solicitud al endpoint `/model/change/model_name` para cambiar de modelo, primero se recuperan las credenciales del modelo solicitado, almacenadas en el archivo *model_info.json*, y luego se actualizan en el *wrapper*. El usuario no tiene la posibilidad de añadir un nuevo modelo, solo podrá utilizar los que son mostrados en el apartado de ajustes.

Manejo de Contexto en los prompts: Se implementa un mecanismo de fragmentación del texto de entrada para optimizar la ejecución de los prompts dentro de los límites de tokens permitidos. Para ello, se hace uso de la biblioteca *tiktoken*¹⁶, que permite calcular el número de tokens de un texto antes de enviarlo al modelo de lenguaje.

- **Fragmentación del Texto:** Dado que los modelos de lenguaje tienen un límite de tokens por prompt, es necesario dividir el texto de entrada en fragmentos manejables. Para lograrlo, se desarrolló el método *divide_text*(text: str, max_tokens: int) ->List[str], que:

Utiliza *tiktoken* para calcular la cantidad de tokens del texto. Divide el texto en segmentos que no excedan el número máximo de tokens permitido (max_tokens). Se asegura que las divisiones se realicen en puntos lógicos, como puntos finales o saltos de línea, evitando cortes abruptos en medio de las palabras.

- **Ejecución de Prompts por Fragmento:** Cada fragmento, una vez generado, se procesa de forma independiente utilizando como contexto en la ejecución de los prompts. Según el tipo de petición, se crean los *prompts* empleando algunos o todos los fragmentos como parte del contexto. Luego son ejecutados como se explicó anteriormente haciendo uso del método *post*.
- **Unión de los Resultados:** Al obtener las respuestas de los prompts, estas se combinan en función de la naturaleza de la petición. Para realizar esta integración y generar una respuesta final coherente, se emplea el método *join_results*(answers: List[str]) ->str.

Peticiones: Para el proceso de las peticiones de la interfaz gráfica se construyeron distintos *endpoints*¹⁷, diseñados para implementar las diversas funcionalidades del sistema. Cada uno de estos endpoints está configurado para realizar tareas específicas,

¹⁶URL de tiktoken: <https://github.com/openai/tiktoken>

¹⁷Un endpoint es una URL específica dentro de una API que permite a los clientes enviar o recibir datos.

asegurando la comunicación eficiente entre el usuario, el modelo de lenguaje y los módulos internos del programa. Los endpoints que terminan interactuando con los LLMs aceptan como parámetros opcionales el idioma del texto escrito por el usuario (para generar la respuesta del modelo en dicho idioma) y los hiperparámetros requeridos para la comunicación con el modelo. A continuación, se describen los mas relevantes basados en las herramientas:

- **Endpoint para buscar documentos científicos** `/quotes/topic` Este recibe mediante un parámetro de query(topic) el tema que se quiere investigar, permitiendo al usuario buscar artículos científicos relacionados con un término o frase clave proporcionada. Para ello, utiliza la biblioteca de Python *arxiv*¹⁸, que facilita la conexión con la API de ArXiv y la realización de peticiones para recuperar los documentos relevantes. Se hace uso del método *Search*, al cual se le pasa el *query* y *max_results*(la cantidad máxima de resultados). Los resultados obtenidos son enviados al usuario para su revisión.

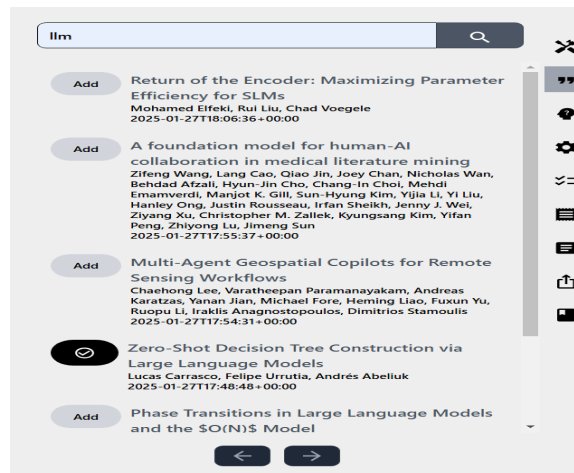


Figura 3.4: Vista de la herramienta para la búsqueda de documentos.

- **Endpoint para agregar documentos a la bibliografía** `/paper/save` Este recibe un json¹⁹ en el cuerpo de la petición con los datos del documento que se quiera agregar. Debido a que no se pudo encontrar una API que proporcione el texto completo de los artículos, se decidió descargar el archivo en formato PDF,

¹⁸URL de la biblioteca de arxiv: <https://pypi.org/project/arxiv/>

¹⁹JSON (JavaScript Object Notation) es un formato ligero de intercambio de datos, basado en texto, fácil de leer y escribir para los humanos, y sencillo de procesar para las máquinas. Se utiliza ampliamente en APIs y almacenamiento de datos debido a su estructura basada en clave-valor, compatible con múltiples lenguajes de programación.

haciendo uso de la biblioteca *arxiv*. Para luego, realizar un tarea en segundo plano donde se procesan los PDFs utilizando la biblioteca *PyPDF2*²⁰, elegida por su eficacia en la conversión de estos archivos. Durante este proceso, se eliminan caracteres especiales, que representan tanto ecuaciones o imágenes, mediante expresiones regulares con la biblioteca *re* que es parte de la biblioteca estándar de Python. Luego se genera la cita en línea del documento, haciendo uso de *prompt engineering* utilizando de una clase extendida de *UserPrompt*, la cual es *UserArXivInLineReferencePrompt*. Finalmente, el texto procesado y los metadatos relevantes del documento se almacenan en un archivo json y sus vectores se agregan a la base de datos vectorial en el módulo de RAG, que puede ser reutilizado posteriormente para generar bibliografías.

- **Endpoint para el chequeo textual** */check*

Este endpoint recibe el texto del usuario junto con las acciones a realizar en el cuerpo de la petición, como chequeo ortográfico, revisión de tono o validación de citas. Para las acciones relacionadas con el chequeo, se utiliza *prompt engineering* en este caso dos prompts diferentes, para cuando no incluya cita se usa *UserCheckPrompt*, y con cita *UserCheckCitationPrompt*, este se trata diferente ya que utiliza el módulo RAG para identificar los documentos más relevantes en la bibliografía del usuario. En el prompt se explica de realizar las acciones suministradas incluyendo la cita, basada en los documentos relevantes suministrados, y que solo responda con el texto corregido y las partes que necesitan citado con la referencia correspondiente. Para el manejo del contexto en este caso el texto suministrado, se hace uso de la biblioteca *tiktoken* para contar los tokens y particionar dicho texto en fragmentos que no superen el límite de tokens de la solicitud del modelo. Estos fragmentos son procesados cada uno por separado y pasados como contexto a los prompts mencionados anteriormente. Luego los chequeos parciales obtenidos se unen, convirtiéndolo en un solo chequeo y se envía al usuario.

²⁰URL de PyPDF2: <https://pypi.org/project/PyPDF2/>

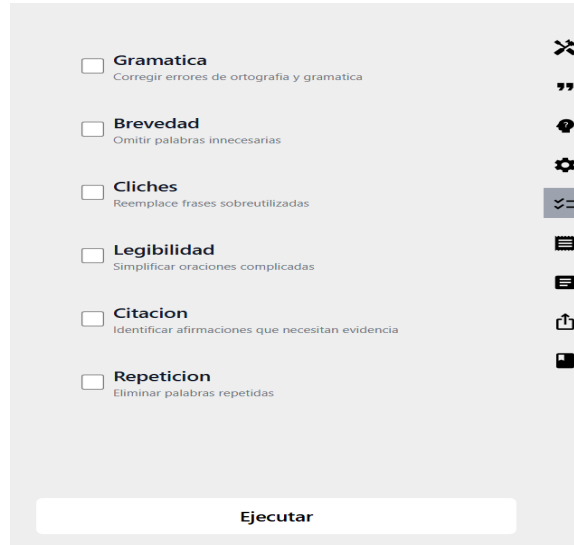


Figura 3.5: Vista de la herramienta de revisión textual.

- **Endpoint para generar temas relacionados y buscar documentos** */topic*

Este endpoint acepta texto del usuario en el cuerpo de la petición y utiliza *prompt engineering* para identificar el tema esencial del contenido con el prompt predefinido, *UserTopicPrompt*. Una vez identificado, se realiza una búsqueda de artículos científicos relacionados con el tema extraído y los resultados son enviados al usuario. Para el manejo del contexto en este caso el texto suministrado, se hace uso de la biblioteca *tiktoken* para contar los tokens y particionar dicho texto en fragmentos que no superen el límite de tokens de la solicitud del modelo. De estos fragmentos solo es tomado el primero y es pasado como contexto al prompt antes mencionado.

- **Endpoint para resumir texto:** */summarize*

Este endpoint recibe el texto proporcionado por el usuario en la solicitud. Una vez validado, se realiza un proceso de *prompt engineering*, para preservar el tono y las características del texto original en el resultado resumido se hace uso de un prompt predefinido, *UserSummarizePrompt* que hereda de *UserPrompt*. Posteriormente, el prompt generado se envía a una API externa de un modelo de lenguaje (LLM), que devuelve el texto resumido al usuario. Para el manejo del contexto en este caso el texto suministrado, se hace uso de la biblioteca *tiktoken* para contar los tokens y particionar dicho texto en fragmentos que no superen el límite de tokens de la solicitud del modelo. De estos fragmentos se generan los resúmenes parciales de cada fragmento. Luego estos resúmenes son unidos (si dicha unión excede el límite de tokens se realiza el paso anterior hasta

que no se exceda dicho límite) para luego generar un resumen de la unión de ellos.

- **Endpoint para extender texto** */extend*

Este endpoint recibe el texto del usuario en el cuerpo de la petición, lo valida y extrae el tema relacionado. Posteriormente, se utiliza el módulo RAG para identificar los documentos más relevantes relacionados con el tema. Estos documentos se toman como contexto para una solicitud a la API externa del LLM, que mediante *prompt engineering* con un prompt predefinido, *UserExtend-TextInLineCitations*. Para el manejo del contexto en este caso los documentos relevantes, se disminuye la cantidad y el tamaño de las particiones, que se pasan como contexto al prompt antes mencionado.

- **Endpoint para generar referencias y citas en texto** */reference/book;*
/reference/article; */reference/website;* */reference/report*

Este endpoint recibe los datos específicos del documento en el cuerpo de la petición (como tipo de documento y formato de cita) y genera la referencia bibliográfica correspondiente junto con su cita en línea, utilizando *prompt engineering* con los prompts predefinidos para cada tipo de archivos: para libro *UserBookReferencePrompt*, para artículo *UserArticleReferencePrompt*, para sitio web *UserWebSiteReferencePrompt*, para informes *UserReportReferencePrompt*.

The image shows a web interface for generating references. It features two dropdown menus at the top: 'Document Type' with 'Libro' selected, and 'Reference Style' with 'APA 7ma Edicion' selected. Below these are several input fields arranged in two columns: 'First name' (containing 'John'), 'Last Name' (containing 'Doe'), 'Title' (containing 'Title'), 'Year' (empty), 'Publisher' (containing 'Publisher'), 'Chapter Title' (containing 'Title'), 'First Page' (empty), and 'Last Page' (empty). At the bottom center is a button labeled 'Generar'. On the right side, there is a vertical sidebar with icons for deleting, quoting, settings, saving, and other functions.

Figura 3.6: Vista de la herramienta para la construcción de referencias.

- **Endpoint para el lenguaje** */language* La función de este endpoint es detectar el idioma del texto suministrado haciendo uso de *prompt engineering* con

un prompt predefinido siendo este, *UserLanguageDetectPrompt*. Se ejecuta antes de cada petición del usuario que incluya el texto, para conocer el idioma actualizado del documento y así devolver la respuesta de sus prompts en dicho idioma. Para el manejo del contexto en este caso el texto suministrado, se hace uso de la biblioteca *tiktoken* para contar los tokens y particionar dicho texto en fragmentos que no superen el límite de tokens de la solicitud del modelo. De estos fragmentos solo es tomado el primero y es pasado como contexto al prompt antes mencionado.

- **Endpoint para el chat** */chat*

La función de este endpoint es interactuar directamente con el modelo de lenguaje especificado, el usuario envía un mensaje y ese mismo mensaje es enviado al LLM y su respuesta es enviada al usuario.

- **Endpoint para la construcción de prompts** */prompt/save*

La función de este endpoint es almacenar el prompt creado por el usuario. Se almacena en un archivo *.json* manteniendo la estructura del prompt, tanto si es mensaje del usuario como del sistema, eso gracias la jerarquía de prompts.

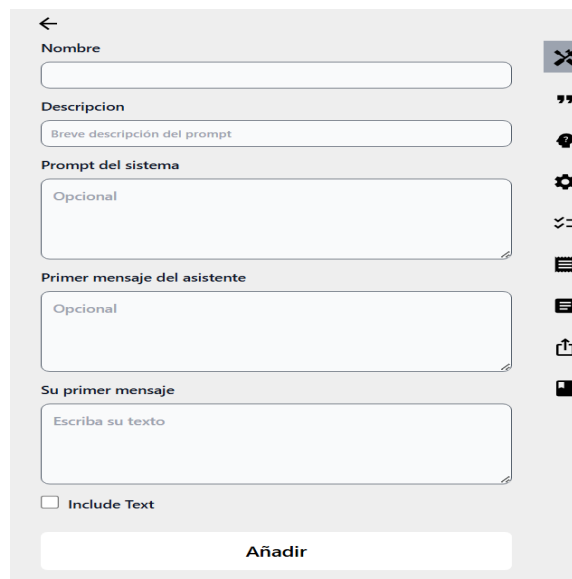


Figura 3.7: Constructor de prompts.

- **Endpoint para el cambio de modelo** */model/change/model_name*

La función de este *endpoint* es reemplazar el modelo anterior por el nuevo y actualizar las credenciales previas con las correspondientes al modelo seleccionado por el usuario.

Dado que el software no maneja un volumen significativo de información, no se consideró necesario el uso de una base de datos tradicional. En su lugar, se optó por almacenar la información de los documentos y otros tipos de información en archivos JSON. Este enfoque permitió una gestión sencilla y eficiente de los datos, acorde con las necesidades del sistema, sin comprometer el rendimiento ni la funcionalidad.

3.3. Limitaciones de las Funcionalidades del Programa

A pesar de las múltiples funcionalidades integradas en el programa, es importante reconocer las limitaciones inherentes a su diseño, implementación y entorno de operación. Estas limitaciones pueden influir en la experiencia del usuario y en la eficiencia general del sistema. A continuación, se describen las principales restricciones identificadas:

Limitación en la longitud de los textos procesados: Debido al uso de modelos de lenguaje (LLM) a través de APIs gratuitas, el sistema está restringido por el número máximo de tokens que puede manejar cada solicitud. Esto implica que textos largos deben ser fragmentados o simplificados. Para lograr esto, el texto es dividido en segmentos que no superan el límite de tokens permitido, para lograr esto se utilizó la biblioteca *tiktoken*.

Restricción en la cantidad de solicitudes por segundo: Las políticas de las APIs gratuitas online imponen límites estrictos en la cantidad de solicitudes que pueden enviarse por segundo o por minuto. Esto limita la capacidad del programa para procesar múltiples solicitudes de manera concurrente. Pero para un modelo cargado localmente la única restricción serían los recursos de la máquina.

Limitaciones en el acceso a texto completo de documentos científicos: Como no se pudo encontrar con una API gratuita que proporcione acceso directo al texto completo de los artículos científicos, se recurre a la descarga de archivos en formato PDF, los cuales son procesados localmente. Este enfoque puede ser limitado en casos donde los documentos contienen formatos complejos, como ecuaciones o gráficos, que dificultan la extracción precisa del contenido.

Dependencia de servicios externos: La funcionalidad del programa depende en gran medida de APIs externas, tanto para la interacción con LLM(no incluyendo local, ya que su única limitación son los recursos de la máquina con el modelo en específico) como para la búsqueda y procesamiento de documentos científicos. Cualquier

interrupción o cambio en las políticas de estos servicios podría afectar directamente el funcionamiento del sistema.

Conclusiones

En este trabajo se desarrolló un prototipo funcional de un editor de textos científicos que integra LLMs mediante técnicas de *prompt engineering* y RAG, abordando desafíos claves en la redacción académica. La combinación de estas tecnologías permitió la generación, corrección y contextualización de contenido académico, brindando mejoras en aspectos no cubiertos de herramientas tradicionales como Microsoft Word, LaTeX o correctores generalistas como Grammarly. La arquitectura modular implementada, facilitó la implementación de funcionalidades como la generación de texto especializado, la corrección estilística, la gestión de referencias y la asistencia interactiva por su capacidad para encajar dichos módulos. Además, el prototipo permite que los LLMs, guiados mediante *prompt engineering*, se adapten a los estándares de la escritura científica, lo que reduce el tiempo y esfuerzo requeridos por los investigadores al automatizar la generación de contenido técnico coherente, la corrección de errores en tiempo real y la creación de citas bibliográficas.

Asimismo, este trabajo contribuye a la integración nativa de LLMs en contextos académicos y la creación de funcionalidades específicas para la escritura científica. La implementación de RAG, respaldada por bases de datos vectoriales y modelos de *embeddings*, permite generar texto fundamentado en bibliografías seleccionadas, asegurando coherencia temática y trazabilidad de las fuentes.

No obstante, el desarrollo del prototipo presentó ciertas limitaciones, como las restricciones impuestas por el uso de APIs gratuitas de LLMs, que afectan la longitud de los textos procesados y dependen de la conectividad externa, además de dificultades en la extracción de texto desde documentos PDF con formatos complejos que incluyen ecuaciones y gráficos. A pesar de estos desafíos, la arquitectura modular implementada confirmó la viabilidad de integrar componentes heterogéneos en un sistema cohesivo, donde el uso de patrones de diseño y estrategias dinámicas para RAG garantizó la escalabilidad y mantenibilidad del código.

En síntesis, los resultados de esta investigación muestran que la combinación estratégica de LLMs, *prompt engineering* y RAG puede transformar la redacción científica, proporcionando un prototipo de herramienta especializada que optimiza la productividad y calidad académica.

Recomendaciones

A partir del desarrollo y análisis del prototipo se identifican diversas oportunidades de mejora y expansión que pueden ser abordadas en futuras iteraciones del sistema.

En primer lugar, se recomienda la implementación de una base de datos para almacenar documentos, referencias bibliográficas y configuraciones personalizadas de los usuarios de manera estructurada. La elección entre una base de datos relacional (como PostgreSQL) o una NoSQL (como MongoDB) dependerá de los requerimientos del sistema. La integración de una base de datos también permitiría optimizar el módulo de RAG, mejorando la precisión y relevancia de los textos generados a partir de fuentes seleccionadas por el usuario.

También se propone la incorporación de plantillas personalizables para formatos académicos específicos, tales como tesis, artículos científicos y pósteres de conferencias. Esto permitiría a los usuarios trabajar sobre una estructura predefinida que facilite la organización del contenido según las normas y estándares de cada tipo de documento.

Por último, en cuanto a la búsqueda de artículos científicos, actualmente el sistema se limita a la extracción de documentos desde arXiv. Para ampliar su funcionalidad, se recomienda extender la búsqueda a otras bibliotecas virtuales y plataformas académicas como Semantic Scholar. Esto proporcionaría a los usuarios un acceso más amplio a fuentes relevantes para sus investigaciones y permitiría mejorar la calidad de las referencias generadas en el sistema.

Estas recomendaciones servirán como base para futuros desarrollos, garantizando su evolución y adaptabilidad a las necesidades de los usuarios.

Bibliografía

- [1] K. Adeli. «Peer review in scientific publications: benefits, critiques, and a survival guide». En: *EJIFCC* (2014). URL: <https://www.ncbi.nlm.nih.gov/pmc/articles/PMC4975196/> (vid. pág. 6).
- [2] M. Altamimi e I. O. William. «Text Embedding Implementation Using Retrieval-Augmented Generation (RAG) Model Combined With Large Language Model». En: *ResearchGate* (2024). URL: https://www.researchgate.net/publication/381105654_Text_Embedding_Implementation_Using_Retrieval-Augmented_Generation_RAG_Model_Combined_With_Large_Language_Model (vid. pág. 14).
- [3] N.I. Altmami y M.E.B. Menai. «Automatic summarization of scientific articles: A survey». En: *Journal of King Saud University-Computer and Information Sciences* (2022). URL: <https://www.sciencedirect.com/science/article/pii/S1319157820303554> (vid. pág. 9).
- [4] Emily M. Bender et al. «On the Dangers of Stochastic Parrots: Can Language Models Be Too Big?» En: *Proceedings of the 2021 ACM Conference on Fairness, Accountability, and Transparency*. FAccT '21. Virtual Event, Canada: Association for Computing Machinery, 2021, págs. 610-623. ISBN: 9781450383097. DOI: 10.1145/3442188.3445922. URL: <https://doi.org/10.1145/3442188.3445922> (vid. pág. 11).
- [5] James Boyko et al. *An Interdisciplinary Outlook on Large Language Models for Scientific Research*. 2023. arXiv: 2311.04929 [cs.CL]. URL: <https://arxiv.org/abs/2311.04929> (vid. pág. 2).
- [6] A. Bozkurt. «Tell me your prompts and I will make them true: The alchemy of prompt engineering and generative AI». En: *Open Praxis* (2024). URL: <https://search.informit.org/doi/pdf/10.3316/informit.T2024041000014390073541090> (vid. pág. 13).
- [7] T. Brown y B. Mann. «Language models are few-shot learners». En: *Advances in Neural Information Processing Systems (NeurIPS)* (2020). URL: <https://arxiv.org/abs/2005.14165> (vid. pág. 10).

- [8] J. E. Casal y M. Kessler. «Can linguists distinguish between ChatGPT/AI and human writing? A study of research ethics and academic publishing». En: *Elsevier* (2023). URL: <https://www.sciencedirect.com/science/article/pii/S2772766123000289> (vid. pág. 7).
- [9] Y. Chang et al. «A survey on evaluation of large language models». En: *ACM Transactions on Intelligent Systems and Technology* (2024). URL: <https://dl.acm.org/doi/abs/10.1145/3641289> (vid. pág. 11).
- [10] B. Chen, Z. Zhang y S. Langrené N.and Zhu. «Unleashing the Potential of Prompt Engineering in Large Language Models: A Comprehensive Review.» En: *Arxiv* (2023). URL: <https://arxiv.org/abs/2310.14735> (vid. págs. 7, 12).
- [11] T. J. Chen. «ChatGPT and other artificial intelligence applications speed up scientific writing». En: *Journal of the Chinese Medical Association* (2023). URL: https://journals.lww.com/jcma/fulltext/2023/04000/chatgpt_and_other_artificial_intelligence.1.aspx (vid. pág. 7).
- [12] T. Cierco-Jimenez R.and Lee, N. Rosillo y R. Cordova. «Machine learning computational tools for systematic reviews». En: *BMC Medical Research Methodology* (2022). URL: <https://link.springer.com/content/pdf/10.1186/s12874-022-01805-4.pdf> (vid. pág. 8).
- [13] Arman Cohan et al. «SPECTER: Document-Level Representation Learning using Citation-Informed Transformers». En: *arXiv preprint arXiv:2004.07180* (2020). URL: <https://arxiv.org/abs/2004.07180> (vid. pág. 10).
- [14] Colaboratorio. *Zettlr: Más que un editor Markdown*. Recuperado el 14 de enero de 2025. 2020. URL: <https://colaboratorio.net/javierinsitu/program/2020/zettlr-mas-que-editor-markdown> (vid. pág. 17).
- [15] W. Dai et al. «Assessing the proficiency of large language models in automatic feedback generation: An evaluation study». En: *Elsevier* (2024). URL: <https://www.sciencedirect.com/science/article/pii/S2666920X24001024> (vid. pág. 10).
- [16] L. Deng. *Deep Learning in Natural Language Processing*. Google Books, 2018 (vid. pág. 8).
- [17] Ilias Dergaa et al. «From human writing to artificial intelligence-generated text: Examining the prospects and potential threats of ChatGPT in academic writing». En: *Biology of Sport* 40.2 (2023), págs. 555-564 (vid. pág. 19).

- [18] J. Devlin et al. «BERT: Pre-training of deep bidirectional transformers for language understanding». En: *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies* (2019). URL: <https://arxiv.org/abs/1810.04805> (vid. pág. 10).
- [19] Y. K. Dwivedi et al. «“So what if ChatGPT wrote it?” Multidisciplinary perspectives on opportunities, challenges and implications of generative conversational AI for research». En: *Elsevier* (2023). URL: <https://www.sciencedirect.com/science/article/pii/S0268401223000233> (vid. pág. 7).
- [20] R. Fok y D.S. Weld. «What can’t large language models do? The future of AI-assisted academic writing». En: *In2Writing Workshop at CHI* (2023). URL: https://cdn.glitch.global/d058c114-3406-43be-8a3c-d3afff35eda2/paper4_2023.pdf (vid. pág. 11).
- [21] Barbara Gastel y Robert A. Day. *How to Write and Publish a Scientific Paper*. 9th. Cambridge, UK: Cambridge University Press, 2022 (vid. pág. 16).
- [22] M.A. Gernsbacher. «Writing Empirical Articles: Transparency, Reproducibility, Clarity, and Memorability». En: *Sage Journals* (2018). URL: <https://journals.sagepub.com/doi/abs/10.1177/2515245918754485> (vid. pág. 6).
- [23] G.D. Gopen y J.A. Swan. «The Science of Scientific Writing». En: *American Scientist* (2018). URL: [http://staff.ustc.edu.cn/~msherk/Publishing%20English%20\(PhD\)/The%20Science%20of%20Scientific%20Writing.pdf](http://staff.ustc.edu.cn/~msherk/Publishing%20English%20(PhD)/The%20Science%20of%20Scientific%20Writing.pdf) (vid. pág. 6).
- [24] P. Hager et al. «Evaluation and mitigation of the limitations of large language models in clinical decision-making». En: *Nature Medicine* (2024). URL: <https://www.nature.com/articles/s41591-024-03097-1> (vid. pág. 11).
- [25] Irene Hames. *Peer Review and Manuscript Management in Scientific Journals: Guidelines for Good Practice*. Oxford, UK: Wiley-Blackwell, 2008 (vid. pág. 16).
- [26] Stephen B. Heard. *The Scientist’s Guide to Writing: How to Write More Easily and Effectively Throughout Your Scientific Career*. 2nd. Princeton, NJ, USA: Princeton University Press, 2022 (vid. pág. 16).
- [27] Douglas Heaven. «ChatGPT and scientific writing: Can AI improve academic texts?» En: *Nature* (2023). URL: <https://www.nature.com/articles/d41586-023-00191-1> (vid. pág. 11).
- [28] M. Hosseini y L. M. Rasmussen. «Using AI to write scholarly publications». En: *Taylor and Francis* (2024). URL: <https://www.tandfonline.com/doi/full/10.1080/08989621.2023.2168535> (vid. pág. 7).

- [29] G. J. Hwang et al. «Vision, challenges, roles and research issues of Artificial Intelligence in Education». En: *Elsevier* (2020). URL: <https://www.sciencedirect.com/science/article/pii/S2666920X20300011> (vid. pág. 7).
- [30] J.K. Iskander et al. «Successful scientific writing and publishing: a step-by-step approach». En: *Preventing Chronic Disease* (2018). URL: <https://www.ncbi.nlm.nih.gov/pmc/articles/PMC6016396/> (vid. pág. 6).
- [31] K. Kalluri. «Exploring Zero-Shot and Few-Shot Learning Capabilities in LLMS for Complex Query Handling». En: *ResearchGate* (2022). URL: https://www.researchgate.net/publication/387512656_Exploring_Zero-Shot_and_Few-Shot_Learning_Capabilities_in_LLMS_for_Complex_Query_Handling (vid. pág. 13).
- [32] K.S. Kalyan. «A survey of GPT-3 family large language models including ChatGPT and GPT-4». En: *Natural Language Processing Journal* (2024). URL: <https://www.sciencedirect.com/science/article/pii/S2949719123000456> (vid. pág. 10).
- [33] M.J. Katz. *From Research to Manuscript: A Guide to Scientific Writing*. 2009 (vid. pág. 6).
- [34] S. Koga. «The integration of large language models such as ChatGPT in scientific writing: harnessing potential and addressing pitfalls.» En: *Korean Journal of Radiology* (2023). URL: <https://www.ncbi.nlm.nih.gov/pmc/articles/PMC10462902/> (vid. pág. 2).
- [35] N.D. Kulkarni y P. Tupsakhare. «Crafting Effective Prompts: Enhancing AI Performance Through Structured Input Design». En: *Journal of Recent Trends in AI Research* (2024). URL: https://www.researchgate.net/publication/385591891_Crafting_Effective_Prompts_Enhancing_AI_Performance_through_Structured_Input_Design (vid. pág. 12).
- [36] I. Lauriola, A. Lavelli y F. Aioli. «An introduction to deep learning in natural language processing: Models, techniques, and tools». En: *Neurocomputing* (2022). URL: <https://www.sciencedirect.com/science/article/pii/S0925231221010997> (vid. pág. 8).
- [37] P. Lewis et al. «Retrieval-Augmented Generation for Knowledge-Intensive NLP Tasks». En: *Advances in Neural Information Processing Systems (NeurIPS)* (2024). URL: <https://proceedings.neurips.cc/paper/2020/hash/6b493230205f780e1bc26Abstract.html> (vid. pág. 14).
- [38] H. Li. «Deep learning for natural language processing: advantages and challenges». En: *National Science Review* (2018). URL: <https://academic.oup.com/nsr/article-pdf/5/1/24/31567231/nwx110.pdf> (vid. pág. 8).

- [39] L. Li, W.A. Geissinger J.and Ingram y E.A. Fox. «Teaching natural language processing through big data text summarization with problem-based learning». En: *Data and Information Management* (2020). URL: <https://www.sciencedirect.com/science/article/pii/S2543925122000572> (vid. pág. 9).
- [40] B. Liefke R.and Stielow y C. Simon. «Making fundamental scientific discoveries by combining information from literature, databases, and computational tools—An example». En: *Computational and Structural Biotechnology Journal* (2021). URL: <https://www.sciencedirect.com/science/article/pii/S2001037021001641> (vid. pág. 8).
- [41] Y. Liu et al. «Summary of ChatGPT-related research and perspective towards the future of large language models». En: *Elsevier* (2023). URL: <https://www.sciencedirect.com/science/article/pii/S2001037021001641> (vid. pág. 10).
- [42] Y. Lyu et al. «CRUD-RAG: A Comprehensive Chinese Benchmark for Retrieval-Augmented Generation of Large Language Models». En: *ACM Transactions on Information Systems* (2024). URL: <https://dl.acm.org/doi/abs/10.1145/3701228> (vid. pág. 14).
- [43] S. Mangul et al. «Systematic benchmarking of omics computational tools». En: *Nature Communications* (2019). URL: <https://www.nature.com/articles/s41467-019-09406-4> (vid. pág. 7).
- [44] OpenAI. «GPT-4 Technical Report». En: *arXiv preprint arXiv:2303.08774* (2023). URL: <https://arxiv.org/abs/2303.08774> (vid. pág. 10).
- [45] Javier Alejandro Oramas López. «Generación Automática de un borrador de Estado del Arte a partir de una colección de documentos científicos». Tesis de mtría. Universidad de La Habana, 2023 (vid. pág. 2).
- [46] Pettoello-Mantovani et al. «The Importance of scientific writing training courses in enhancing the dissemination of research findings». En: *Global Pediatrics* (2024). URL: <https://www.sciencedirect.com/science/article/pii/S2667009724000204> (vid. pág. 6).
- [47] S. R. Piccolo y M. B. Frampton. «Tools and techniques for computational reproducibility». En: *GigaScience* (2016). URL: <https://academic.oup.com/gigascience/article-abstract/5/1/s13742-016-0135-4/2720991> (vid. pág. 7).
- [48] Tyler Thomas Procko et al. «Leveraging Large Language Models on the Traditional Scientific Writing Workflow». En: *2024 Conference on AI, Science, Engineering, and Technology (AIxSET)*. 2024, págs. 154-161. DOI: 10.1109/AIxSET62544.2024.00028 (vid. pág. 10).

- [49] T. Rasul et al. «The role of ChatGPT in higher education: Benefits, challenges, and future research directions». En: *Journal of Applied Research* (2023). URL: https://research.aib.edu.au/files/35431301/787_Article_Text_3375_1_10_20230510.pdf (vid. pág. 7).
- [50] P. P. Ray. «ChatGPT: A comprehensive review on background, applications, key challenges, bias, ethics, limitations and future scope». En: *Elsevier* (2023). URL: <https://www.sciencedirect.com/science/article/pii/S266734522300024X> (vid. pág. 11).
- [51] N. Rosillo, R. Cordova y R. Cierco-Jimenez. «Machine learning computational tools to assist the performance of systematic reviews: A mapping review». En: *BMC Medical Research Methodology* (2022). URL: <https://link.springer.com/content/pdf/10.1186/s12874-022-01805-4.pdf> (vid. pág. 8).
- [52] Shuming Shi et al. «Effidit: An Assistant for Improving Writing Efficiency». En: *Proceedings of the 61st Annual Meeting of the Association for Computational Linguistics*. 2023, págs. 508-515. URL: <https://effidit.qq.com/> (vid. pág. 18).
- [53] Shizuka Shirai, Takahiro Nakahara y Tetsuo Fukui. «MathTOUCH Editor: Rich-text Editor for Math E-learning Using an Intelligent Math Input Interface». En: *Journal of Information Processing* 31 (2023), págs. 775-785. DOI: 10.2197/ipsjjip.31.775. URL: <https://doi.org/10.2197/ipsjjip.31.775> (vid. pág. 17).
- [54] B. Stielow, C. Simon y R. Liefke. «Combining literature, databases, and computational tools for scientific discovery». En: *Elsevier* (2021). URL: <https://www.sciencedirect.com/science/article/pii/S2001037021001641> (vid. pág. 8).
- [55] C. Strobl et al. «Digital support for academic writing: A review of technologies and pedagogies». En: *Computers and Education* (2019). URL: <https://www.academia.edu/download/98662057/j.compedu.2018.12.00520230214-1-r19ji8.pdf> (vid. pág. 7).
- [56] T. Susnjak y A. Mathrani. «Automating Systematic Literature Reviews with Retrieval-Augmented Generation: A Comprehensive Overview». En: *Applied Sciences* (2024). URL: <https://www.mdpi.com/2076-3417/14/19/9103> (vid. pág. 15).
- [57] Kevin Talavera Díaz. «Evaluación de habilidades cognitivas de diversos LLMs». Tesis de mtría. Universidad de La Habana, 2023 (vid. pág. 3).
- [58] Ian Tenney, Dipanjan Das y Ellie Pavlick. «BERT Rediscovered the Classical NLP Pipeline». En: *arXiv preprint arXiv:1905.05950* (2019). URL: <https://arxiv.org/abs/1905.05950> (vid. pág. 11).

- [59] Toolify. *Lex: La herramienta AI gratuita para superar el bloqueo del escritor*. Recuperado el 14 de enero de 2025. n.d. URL: <https://www.toolify.ai/es/ai-news-es/lex-la-herramienta-ai-gratuita-para-superar-el-bloqueo-del-escriptor-1689515> (vid. pág. 17).
 - [60] Xinyi Wang, Wanrong Zhu y William Wang. *Large Language Models Are Implicitly Topic Models: Explaining and Finding Good Demonstrations for In-Context Learning*. Ene. de 2023. DOI: 10.48550/arXiv.2301.11916 (vid. pág. 13).
 - [61] Laura Weidinger et al. *Ethical and social risks of harm from Language Models*. 2021. arXiv: 2112.04359 [cs.CL]. URL: <https://arxiv.org/abs/2112.04359> (vid. pág. 11).
 - [62] D. Whitehead y Z. Schneider. «Writing and presenting research findings for dissemination». En: *Nursing and Midwifery Research* (2013). URL: https://www.researchgate.net/publication/255965934_Writing_and_presenting_research_findings_for_dissemination (vid. pág. 6).
 - [63] T. Young et al. «Recent trends in deep learning-based natural language processing». En: *IEEE Computational Intelligence Magazine* (2018). URL: <http://sentitc.net/deep-learning-for-nlp-review.pdf> (vid. pág. 9).
 - [64] J. Zaghir et al. «Prompt engineering paradigms for medical applications: Scoping review». En: *Journal of Medical Internet Research* (2024). URL: <https://www.jmir.org/2024/1/e60501/> (vid. pág. 13).
- Öchsner, A. (2013). Introduction to scientific publishing: Backgrounds, concepts, strategies. Springer. <https://doi.org/10.1007/978-3-642-38646-6>
 - Bailey, S. (2014). Academic writing: A handbook for international students (4th ed.). Routledge. Öchsner, A. (2013). Introduction to scientific publishing: Backgrounds, concepts, strategies. Springer. <https://doi.org/10.1007/978-3-642-38646-6>
 - Lindsay, D. (2020). Scientific writing: Thinking in words (2nd ed.). CSIRO Publishing.
 - Strauss, P., & Manalo, E. (2023). “Most good papers are published in English”: Japanese academics’ perspectives on the benefits and drawbacks of writing and publishing in English. *Journal of English for Research Publication Purposes*, 4(1), 3–21. <https://doi.org/10.1075/jerpp.22009.str>