



DEPARTAMENTO DE  
INFORMÁTICA TRELEW  
FACULTAD DE INGENIERÍA

---

## **IA y el Análisis del Lenguaje Natural como soporte en el proceso de Desarrollo de Software**

---

**Alumnos:**

**Aguila Barré, Maximiliano Andrés  
De Marco Andrada, Emiliano**

**Tutor:**

**Dr. Firmenich, Diego Andrés**

**Co-Tutor:**

**Lic. Pazos, Bruno**

**Proyecto de Investigación:**

**Res. CDFI N° 233/22**

**2024**

# Agradecimientos

Queremos dedicar este trabajo a nuestros familiares, amigos, compañeros y profesores que nos apoyaron y motivaron durante todas las etapas del desarrollo de nuestra tesina y a lo largo de todo nuestro proceso en la universidad. Gracias a su confianza y apoyo, encontramos la motivación para desarrollar este proyecto.

Un agradecimiento especial a nuestro tutor, Diego Firmenich, quien nos guió con su experiencia, dedicación y compromiso a lo largo de este proceso y de toda nuestra carrera. También queremos mencionar a Bruno Pazos, por su valiosa orientación, asistencia y apoyo en todo momento. Agradecemos a ambos, quienes nos brindaron la posibilidad de formar parte de un proyecto de investigación y aportar nuestro granito de arena, el cual evolucionó en nuestro trabajo de tesina.

También queremos agradecer a Mutisia Ríos por prestar su mirada profesional para el diseño de la interfaz de usuario y del logo de la extensión desarrollada.

El logo original de Git es creación de Jason Long y está licenciado bajo Creative Commons Attribution 3.0 Unported License.

Por último, queremos expresar nuestro agradecimiento al jurado por el tiempo y esfuerzo que dedicaron para evaluar nuestra tesina.

# Resumen

El presente trabajo de tesina aborda la relación entre la Inteligencia Artificial (IA) y el Análisis del Lenguaje Natural (NLP) a fin de evaluarlas como potenciales herramientas de soporte en el proceso de desarrollo de software. A lo largo del documento, se exploran los conceptos fundamentales de estas dos tecnologías, así como su aplicación en América Latina, su enfoque particular en el idioma español y el panorama del mercado actual, para luego concluir en el desarrollo de un proyecto de software que nos permitiera documentar nuestras experiencias en la materia.

Al inicio de este proceso, el estado del arte era muy diferente al existente hoy día. A lo largo del periodo que abarcó desde la concepción original del tema de tesina y del objetivo planteado hasta el inicio formal del desarrollo y su finalización, la oferta de IA al público general creció de manera exponencial.

A principios del año 2021, durante los primeros pasos que dimos incursionando en estas tecnologías, en el marco del proyecto de investigación "Herramientas de Desarrollo y Procesamiento de Lenguaje Natural e Inteligencia Artificial en la Ingeniería de Software Web", previo a la presentación de la propuesta y al inicio formal del trabajo de tesina, herramientas como Github Copilot y ChatGPT, altamente populares y al alcance del público actualmente, aún estaban en construcción y no eran de masivo conocimiento. El repentino y acelerado avance en torno a estas innovaciones nos obligó a adaptarnos respecto al desarrollo que teníamos proyectado producir; sin embargo, siempre mantuvimos un mismo objetivo y un enfoque principal: evaluar el estado del arte de las tecnologías de IA y NLP y exponer su potencial como grandes herramientas para mejorar y facilitar el trabajo de los desarrolladores.

Una vez iniciado formalmente este trayecto, a fines de 2023, y durante el paso de los meses, las tecnologías no pararon de evolucionar, como era de esperarse. A pesar de la desafiante tarea de documentar un estado del arte tan dinámico y constantemente cambiante, fuimos capaces de recopilar una cantidad satisfactoria de datos y herramientas y de sacar provecho de estos mismos avances para lograr concretar un documento con resultados más que satisfactorios que cumplieran con todos los objetivos que planteamos para nuestra tesina.

De este modo, podemos afirmar que el avance en las herramientas de IA y NLP, por un lado, nos jugó en contra añadiendo una dificultad extra al esfuerzo por mantener nuestra investigación actualizada y relevante pero, por otro, nos benefició permitiéndonos aprovechar la situación para alcanzar nuestros objetivos. Pero principalmente, sobre todo, podemos asegurar que nos ofreció una experiencia enriquecedora al sumergirnos en las turbulentas corrientes del cambio impulsadas por los avances en tecnologías que, sin poder preverlo en su momento, se posicionarían bajo la mira de las más grandes organizaciones de ciencia e innovación, en una carrera tecnológica por lanzar el producto más revolucionario. Mismas tecnologías las cuales, hoy más que nunca, están en boca de todos. Todo esto coincidentemente y en simultáneo al desarrollo de nuestra tesina.

# Índice

<b>Introducción.....</b>	<b>7</b>
Objetivos.....	7
Objetivos generales.....	7
Objetivos específicos.....	7
Motivación.....	7
Desarrollos Propuestos.....	9
Resultados Esperados.....	9
Inteligencia Artificial (IA).....	10
Definición.....	10
IAs Fuertes y Débiles.....	12
IA Débil.....	12
IA Fuerte.....	13
IA Débil vs IA Fuerte.....	13
IA en América Latina.....	14
Área de aplicación.....	15
El español como idioma de la Inteligencia Artificial.....	15
Procesamiento de lenguaje natural (NLP).....	16
Definición.....	16
Procesamiento del lenguaje.....	20
Funcionamiento del procesamiento de lenguaje natural (NLP).....	20
Data Processing.....	21
Stemming.....	21
Lemmatization.....	21
Sentence segmentation.....	22
Stop word removal.....	22
Tokenization.....	23
Extracción de características.....	23
Bag-of-Words.....	24
TF-IDF.....	24
Word2Vec.....	25
GLoVe.....	27
Modelado (Modeling).....	27
Procesamiento de lenguaje natural y sus técnicas.....	27
Técnicas tradicionales de NLP de aprendizaje automático.....	28
Regresión Logística.....	28
Naive Bayes.....	29
Árboles de Decisión.....	30
Asignación de Dirichlet latente (LDA).....	31

Modelo oculto de Markov.....	31
Técnicas de NLP de aprendizaje profundo.....	32
Red neuronal convolucional (CNN).....	32
Red neuronal recurrente (RNN).....	34
Auto-Encoder.....	35
Encoder-Decoder Seq2Seq.....	36
Transformers.....	37
NLP y sus controversias.....	38
Usos del procesamiento de lenguaje natural (NLP).....	39
Estado del arte y contexto.....	41
LLMs.....	43
Modelos de lenguaje.....	43
Capacidades y aplicaciones de los LLMs.....	45
Capacidades de los LLMs.....	45
Aplicaciones de los LLMs.....	46
Conclusiones y perspectivas futuras.....	46
Prompts en IA.....	47
¿Qué es un Prompt?.....	47
Propósito del Prompt.....	47
Importancia de prompts precisos.....	48
¿Cómo obtener un buen Prompt?.....	48
Relevancia del Prompt en la IA.....	49
Consideraciones Adicionales.....	49
Importancia de “prompt engineering”.....	50
Fundamentos de Prompt Engineering.....	51
Factores que influyen en la elección de instrucciones (prompts).....	51
Buenas prácticas.....	52
GPT-4 - OpenAI.....	55
PaLM 2 y Gemini - Google DeepMind.....	56
LLaMA 3.1 - Meta.....	56
Claude 3.5 - Anthropic.....	57
StarCoder.....	58
MPT (MosaicML).....	58
Modelos Open Source.....	59
LLaMA 3.1.....	59
Mistral.....	59
Falcon 2.....	60
StarCoder.....	60
MPT (MosaicML Pretrained Transformer).....	60
Herramientas.....	60
GitHub Copilot.....	60

Tabnine.....	61
Kite.....	61
CodeT5.....	62
Amazon CodeWhisperer.....	62
<b>Desarrollo.....</b>	<b>64</b>
Dominio.....	64
Alcance.....	65
Relevancia.....	66
Modelos LLM y prompt.....	66
Modelo: OpenAI - GPT3.5.....	68
Modelo: Google - Gemini.....	69
Meta - Llama3.....	70
Elección de modelo LLM.....	72
Funcionalidades.....	73
Asistente de Mensajes.....	74
Catálogo de Commits.....	77
Experimentación.....	81
Caso 1.....	82
Caso 2.....	88
Caso 3.....	90
Mejoras Posibles.....	91
Solucionar la Limitación de Tokens.....	91
Estandarización de mensajes.....	93
Personalización del Prompt.....	93
Personalización de otros parámetros.....	94
Refinamiento de prompts.....	94
Implementación de un modelo local.....	94
Edición de Sugerencias.....	94
Pre-cargado de código para evaluar contexto.....	94
<b>Conclusiones.....</b>	<b>95</b>
<b>Bibliografía.....</b>	<b>96</b>
<b>Glosario.....</b>	<b>100</b>

# Introducción

## Objetivos

### Objetivos generales

El objetivo general de la presente propuesta de tesina es el estudio del potencial que significa el uso de la Inteligencia Artificial y el Análisis del Lenguaje Natural como herramientas de soporte para el desarrollo de aplicaciones.

### Objetivos específicos

- Investigar, estudiar y documentar el marco teórico de referencia.
- Analizar en profundidad el estado actual del mercado en relación a la temática que se propone.
- Estudiar y documentar herramientas disponibles de Inteligencia Artificial que hagan uso del Análisis de Lenguaje Natural y sus distintos campos de aplicación.
- Elaborar una herramienta integrada a un IDE que haciendo uso de *análisis de lenguaje natural* pueda ser de utilidad para facilitar la creación de aplicaciones de software durante la etapa de desarrollo de código, ya sea a nivel de codificación o desde un punto de vista metodológico.
- Realizar y documentar una experimentación con el software desarrollado.

## Motivación

En la actualidad, el campo de la inteligencia artificial (AI, por sus siglas en inglés) está creciendo día a día a un ritmo cada vez más acelerado. Tal es el motivo por el cual progresivamente se encuentran más herramientas disponibles públicamente para el uso tanto del usuario común como para expertos informáticos y desarrolladores de software. Dentro de esta tendencia se destacan herramientas que hacen uso del análisis del lenguaje natural como pilar para la comunicación entre el usuario y la aplicación. Dichos son los casos de ChatGPT, Perplexity y Codex, entre otros.

Cuando hablamos de lenguaje natural, nos referimos a aquel que ha sido desarrollado naturalmente como método de comunicación entre las personas (Cohen, D. I. (1996). Introduction to computer theory. John Wiley & Sons). Dicho esto, el análisis de lenguaje natural consiste en un conjunto de técnicas que combinan modelos lingüísticos, estadísticos, de *machine learning* (ML) y de *deep learning* (DL) con el fin de darle a los sistemas informáticos la capacidad de comprender el lenguaje hablado. En la práctica, esto implica la posibilidad de proveer a una computadora una entrada en formato de lenguaje natural -oral o escrito- y que, tras procesarla, la misma sea capaz de interpretar su significado en términos inherentes a su función, permitiendo dar una respuesta acorde.

El uso de estas tecnologías nos brinda una facilidad de comunicación cada vez mayor entre el usuario y los sistemas, haciendo más natural la interacción entre los mismos y abriendo un abanico de posibilidades en el conjunto de instrucciones que el usuario común puede dar. Algunos ejemplos de sistemas emergentes que podemos observar como consecuencia de las facilidades que brindan estas técnicas, son los sistemas de generación de imagen mediante textos descriptivos y los asistentes virtuales comandados por voz.

Resulta importante no pasar por alto que el potencial en el uso del lenguaje natural en sistemas informáticos, no se acota únicamente a la posibilidad de ser implementado al momento de brindar directivas. Las IAs pueden, además, contar con la capacidad de analizar textos u otras fuentes de información expresadas en lenguaje natural con el fin de utilizarlos como base para desempeñar su función. Las IAs de análisis de sentimiento y aquellas que recaban información de internet u otras fuentes para responder a preguntas concretas, son ejemplos de ello.

Tras todo lo mencionado anteriormente, resulta claro el potencial de estas tecnologías para facilitar tareas y procesos de todo tipo, entre los que podríamos destacar aquellos realizados por los desarrolladores de software.

Dentro del extenso proceso de trabajo que realizan los desarrolladores, una gran parte es realizada mediante el uso de las *Ambientes de Desarrollo Integrados* (IDEs) que combinan herramientas comunes de programación en una interfaz de usuario única y que comúnmente consisten en un editor de código, utilidades de "construcción" (build) y depuradores (debuggers).

Actualmente y desde hace tiempo, estos IDEs cuentan con la posibilidad de agregar plugins específicos que extiendan la funcionalidad de los mismos con el fin de brindar nuevas facilidades a los usuarios a la hora de trabajar con ellos. Tal es el caso, por ejemplo, de los plugins Path IntelliSense (con 107,802 instalaciones), Prettier (con 1,257,556 instalaciones) o GitLens (con 1,237,438 instalaciones) para uno de los entornos más utilizados como lo es Visual Studio Code<sup>1</sup>; el primero siendo un plugin orientado a brindar herramientas de completado, asistencia y sugerencia de código, el segundo trata de un plugin que ayuda al formateo de código, mientras que el tercero se trata de un complemento desarrollado a fin de proveer a los programadores de información referida a los repositorios Git de sus proyectos, incluyendo funcionalidades para visualizar los autores de cada archivo, el historial de modificaciones, búsqueda y exploración de commits por mensaje, autor, archivo o cambios específicos, entre otras cosas. El uso de estos plugins en conjunto con la emergente tendencia

---

<sup>1</sup> <https://code.visualstudio.com/>

de la tecnología hacia la IA y el análisis del lenguaje natural, hacen que el potencial de combinar estas tres herramientas resulte atractivo.

Los IDEs y los plugins son herramientas importantes para el desarrollo de software, pero su efectividad depende en gran medida del lenguaje utilizado. Algunos datos de noticias tecnológicas reflejan que tras el chino mandarín, el español es la segunda lengua más hablada en el mundo, su uso en la inteligencia artificial es limitado y apenas está presente en aproximadamente el 25% de modelos del lenguaje mundial. Esto se debe en parte a la escasa inversión y protagonismo del lenguaje Español en la inteligencia artificial y la ventaja del Inglés como lengua por excelencia.

Otro dato a recalcar, es que los algoritmos de IA necesitan aprender y lo hacen mayoritariamente en idioma Inglés, siendo éste el idioma en que mejor se comportan, mientras que el idioma Español o castellano representa menos del 30% del mercado mundial de las tecnologías del procesamiento de lenguaje natural (López et al., 2019; La Cabeza, 2021; Carretero, 2021).

Este contraste o retraso de nuestra lengua frente al Inglés no es atribuible a una única causa. Por un lado, obedece a una razón histórica y en estos días el idioma Inglés se ha consolidado como la lengua por excelencia a nivel global. Otras razones son la variabilidad y las particularidades del idioma Español, que en sus variantes latinoamericanas pueden generar dificultades.

Si bien estos son elementos a tener en cuenta a la hora de introducir nuestro idioma y lenguaje al mundo de la IA, el factor determinante es la inversión y recursos para poder crear soluciones en español y no solo adaptadas a nuestro lenguaje.

## Desarrollos Propuestos

La propuesta de trabajo de tesina plantea el desarrollo de un complemento para un Ambiente Integrado de Desarrollo (IDE) que, haciendo uso de herramientas basadas en el Procesamiento de Lenguaje Natural en el marco de la Inteligencia Artificial, extienda las utilidades y/o funcionamiento del mismo de manera que brinde soporte al proceso de desarrollo de software.

## Resultados Esperados

Luego de llevar a cabo un proceso de investigación, desarrollo y análisis, se espera poder finalizar el presente trabajo de tesina y documentar el marco teórico de referencia, como así también los resultados y conjeturas obtenidos, de manera que nos lleven finalmente a poder analizar el impacto que la incorporación de estas tecnologías pudiera tener en el proceso de desarrollo de software.

# Marco Teórico

## Inteligencia Artificial (IA)

### Definición

La Inteligencia Artificial (IA) es una rama de la ciencia informática que tiene como objetivo diseñar tecnología que emule la inteligencia humana. La IA aprovecha computadoras y máquinas para imitar las capacidades de resolución de problemas y toma de decisiones de la mente humana. Las máquinas pueden llevar a cabo procesos propios de la inteligencia humana, como aprender, razonar o autocorregirse.

Al contrario de la concepción arcaica que se tenía hace algunos años, en la actualidad se sabe que la IA no busca reemplazar a los humanos, sino contribuir al desarrollo de sus capacidades y su impacto. Sus diferentes aplicaciones tienen como meta mejorar tanto el desempeño como la experiencia de sus usuarios.

Es importante mencionar algunos de los debates actuales en torno al desarrollo de tecnologías avanzadas de IA, especialmente en cuanto a los riesgos que representan para la sociedad. Un ejemplo clave de esta preocupación es la carta abierta firmada en marzo de 2023 por más de mil expertos, investigadores y personalidades influyentes de todo el mundo, incluidos Elon Musk y Steve Wozniak, que llamaron a una pausa de seis meses en el desarrollo de sistemas de inteligencia artificial avanzados. (Yudkowsky, 2023)

Esta carta abierta destacó que, si bien la IA tiene el potencial de transformar muchas industrias y mejorar la calidad de vida, también podría generar consecuencias imprevistas y desestabilizadoras si no se maneja adecuadamente. La falta de transparencia en el funcionamiento interno de los modelos de IA, junto con la velocidad con la que están evolucionando, plantea riesgos serios, desde la proliferación de información errónea hasta la posible automatización masiva de empleos. Este grupo de expertos advirtió que las empresas tecnológicas están inmersas en una carrera desenfrenada por crear modelos cada vez más poderosos sin un control adecuado, lo que podría desencadenar efectos adversos que ni siquiera sus propios creadores pueden predecir o controlar.

La controversia en torno a este tema radica en la creciente brecha entre el desarrollo de capacidades tecnológicas y la falta de regulaciones y protocolos de seguridad adecuados para su implementación. Los firmantes de la carta señalan que, aunque las tecnologías de IA pueden ser extremadamente beneficiosas, su avance debe estar alineado con la seguridad y el bienestar global. En este sentido, solicitaron una pausa para reflexionar sobre el impacto de la IA, y recalcaron que esta detención no significa cesar el progreso, sino asegurarse de que los sistemas sean más transparentes, seguros y confiables antes de continuar.

La discusión generada por esta carta abierta subraya la necesidad de un equilibrio entre la innovación y la regulación, y resalta la creciente conciencia de que el desarrollo desmedido de tecnologías avanzadas de IA podría tener consecuencias tanto positivas como negativas.

Por otra parte, Stuart Russell y Peter Norvig en su libro “Artificial Intelligence - A Modern Approach” definen a la Inteligencia Artificial en torno a cuatro conceptos; Pensamiento Humano, Pensamiento Racional, Accionar Humano y Accionar Racional.

**Accionar Humano:** Para desarrollar este concepto, los autores traen a colación al Test de Turing (Turing, 1950). Este test surgió para dar una definición operacional satisfactoria del concepto de inteligencia. Según este, una computadora supera esta prueba si un agente humano no es capaz de diferenciar si una serie de interrogantes, planteadas previamente por él, fueron contestadas por un humano o una máquina. Para ello, la computadora debería tener las capacidades de:

- *Procesamiento de Lenguaje Natural (NLP)*: para la comprensión del lenguaje.
- Representación de Conocimiento: para almacenar su base de conocimiento.
- Razonamiento Automatizado: para contestar las interrogantes presentadas haciendo uso del conocimiento almacenado.
- Machine Learning: para adaptarse a nuevas circunstancias y detectar y extrapolar patrones.

Este modelo, se abstrae de la interacción física entre el humano y la máquina. En un Test de Turing “Total”, sería necesario incorporar reconocimiento de imagen y robótica a las capacidades anteriormente nombradas. Sin embargo, los investigadores y desarrolladores de IA no buscan emular el comportamiento humano al 100%, sino más bien explotar las capacidades que la tecnología propone.

**Pensamiento Humano:** Es necesario comprender primero el funcionamiento del pensamiento humano previo a pretender emularlo. Por ello, se plantean tres acercamientos; introspección del pensamiento, experimentos psicológicos y análisis del cerebro en acción. Una vez logrado este propósito, resultaría posible plasmarlo programáticamente. Frecuentemente, se desea que la búsqueda de la solución por parte de la computadora sea implementando una línea de pensamiento análoga a la de la mente humana, por sobre llegar al mismo resultado de la manera más óptima operacionalmente.

La Ciencia de la Cognitividad es un campo multidisciplinario complejo que busca construir teorías precisas sobre la mente humana y lo hace utilizando conjuntamente modelos computacionales de IA y técnicas experimentales de la psicología.

**Pensamiento Racional:** Basado en los conceptos teóricos de la Lógica, se busca el desarrollo de programas informáticos que sean capaces de resolver cualquier problema descrito en notaciones lógicas. Frente a este objetivo se presentan dos obstáculos. Por un lado, la dificultad de trasladar conocimiento informal a términos formales representables dentro de

las notaciones de la lógica. Por otro lado, el poder computacional necesario para resolver las premisas planteadas, el cual en la realidad es limitado y frecuentemente insuficiente, a menos que se cuente con algún tipo de guía de procedimiento.

**Accionar Racional:** Un agente racional es aquel que acciona con el objetivo de llegar al mejor resultado, pero que, a su vez, en casos de incertidumbre, es capaz de operar llegando a un buen desenlace esperado. En la realidad, sin embargo, una completa racionalidad no siempre es posible debido a los límites computacionales.

## IAs Fuertes y Débiles

La inteligencia artificial se puede dividir o clasificar en dos grandes categorías: IA Débil o Estrecha e IA Fuerte o Sólida.

La inteligencia artificial débil sustenta la mayor parte de la IA que nos rodea hoy en día y es aquella que incorpora un sistema diseñado para realizar un trabajo en particular. "Estrecha" podría ser un calificativo más preciso para este tipo de IA, ya que es cualquier cosa menos débil, permite algunas aplicaciones complejas y avanzadas, como Siri de Apple<sup>2</sup>, Alexa de Amazon<sup>3</sup>, IBM watsonx<sup>4</sup> y los vehículos autónomos.

Los sistemas de inteligencia artificial fuerte o sólida son sistemas que llevan a cabo las tareas que se consideran similares a las humanas. La IA fuerte sigue siendo totalmente teórica y no existen ejemplos prácticos en la actualidad.

### IA Débil

- **Ejemplos más específicos:**
  - **En el mundo empresarial:** Sistemas de recomendación en plataformas de streaming, chatbots para atención al cliente, análisis de datos para tomar decisiones estratégicas.
  - **En la vida cotidiana:** Filtros de spam en correos electrónicos, reconocimiento facial en teléfonos móviles, asistentes virtuales como Google Assistant<sup>5</sup>.
- **Limitaciones:**
  - **Falta de comprensión profunda:** Aunque pueden realizar tareas específicas de manera excelente, no comprenden el significado detrás de las tareas. Por ejemplo, un chatbot puede responder preguntas sobre un producto, pero no entiende realmente qué es ese producto.

---

<sup>2</sup> <https://www.apple.com/es/siri/>

<sup>3</sup> <https://www.amazon.es/b?node=15823644031>

<sup>4</sup> <https://www.ibm.com/es-es/watsonx>

<sup>5</sup> [https://assistant.google.com/intl/es\\_es/](https://assistant.google.com/intl/es_es/)

- **Dependencia de datos:** Necesitan grandes cantidades de datos para entrenar y mejorar su desempeño.
- **Incapacidad para generalizar:** Si se les presenta una situación que no han visto antes, pueden tener dificultades para adaptarse.
- **Aplicaciones futuras:**
  - **Automatización de tareas repetitivas:** Liberando a los humanos para que se enfoquen en tareas más creativas y estratégicas.
  - **Mejora de la eficiencia en diversos sectores:** Desde la medicina hasta la agricultura.
  - **Desarrollo de nuevas tecnologías:** Como la realidad virtual y aumentada.
- **Sesgos:**
  - Los algoritmos de IA pueden perpetuar y amplificar los sesgos presentes en los datos de entrenamiento, lo que por ejemplo, puede llevar a discriminación en áreas como la contratación, la justicia penal y la publicidad.

#### IA Fuerte

- **Características:**
  - **Conciencia:** Capacidad de ser consciente de sí misma y de su entorno.
  - **Aprendizaje continuo:** Capacidad de aprender y adaptarse a nuevas situaciones de manera autónoma.
  - **Razonamiento complejo:** Capacidad de resolver problemas complejos y tomar decisiones basadas en la lógica.
  - **Creatividad:** Capacidad de generar ideas nuevas y originales.
- **Desafíos:**
  - **Computación:** Requeriría una capacidad de procesamiento increíblemente grande.
  - **Algoritmos:** Desarrollar algoritmos capaces de simular la complejidad del cerebro humano es un desafío enorme.
  - **Datos:** La cantidad y calidad de los datos necesarios para entrenar una IA fuerte son inimaginables en la actualidad.

#### IA Débil vs IA Fuerte

A modo resumen, a continuación veremos una tabla comparativa entre las categorías descritas anteriormente.

Tabla 1. IA Débil vs IA Fuerte

<i>Característica</i>	<i>IA Débil (Estrecha)</i>	<i>IA Fuerte (General)</i>
Objetivo	Realizar tareas específicas y bien definidas	Poseer una inteligencia general comparable a la humana

Capacidad	Limitada a un dominio específico	Amplia y flexible, capaz de aprender y adaptarse a cualquier tarea
Conciencia	Sin conciencia de sí misma ni del mundo	Potencialmente consciente de sí misma y capaz de comprender su entorno
Aprendizaje	Basado en grandes cantidades de datos etiquetados	Aprendizaje continuo y autónomo, capaz de aprender de la experiencia y generalizar
Razonamiento	Lógico y basado en reglas predefinidas	Capaz de razonar de forma abstracta, creativa y causal
Ejemplos	Asistentes virtuales, reconocimiento facial, motores de recomendación, vehículos autónomos	No existe aún, pero se espera que sea capaz de realizar cualquier tarea intelectual humana
Desafíos	Requiere grandes cantidades de datos, puede ser vulnerable a sesgos, limitada a tareas específicas	Requiere una comprensión profunda de la inteligencia humana, plantea desafíos éticos y sociales
Implicaciones	Automatización de tareas, mejora de la eficiencia, personalización	Transformación radical de la sociedad, nuevas formas de trabajo y ocio, posibles riesgos existenciales

## IA en América Latina

La inteligencia artificial en América Latina a diferencia de los países más desarrollados, aún continúa en etapas iniciales. Sin embargo, existen proyectos ambiciosos que esperan estimular el desarrollo del talento tecnológico en la región.

Uno de los problemas clave es la brecha de acceso a la tecnología. No todos tienen la misma facilidad para acceder a la educación en IA o a las herramientas necesarias para desarrollarla. Además, la falta de inversión y financiamiento en investigación y desarrollo de tecnologías emergentes puede frenar el progreso. La colaboración entre el sector público y privado es crucial para impulsar la innovación.

En cuanto a la etapa de desarrollo, América Latina está en diferentes niveles. Algunos países han avanzado significativamente en la adopción de tecnologías de IA, mientras que otros están en las etapas iniciales. La falta de una estrategia regional unificada también puede ser un obstáculo.

Muchos países están tomando medidas para mejorar la infraestructura, la educación y la colaboración entre sectores. Con el tiempo, podríamos ver un crecimiento significativo en la adopción de la IA en América Latina (ILIA, 2024).

## Área de aplicación

La Inteligencia Artificial (IA) es un área multidisciplinaria. Desde sus fundamentos hasta sus aplicaciones más recientes, abarca una amplia gama de sectores de la economía, la ciencia y la sociedad en su conjunto. Las aplicaciones de la inteligencia artificial son infinitas. La tecnología se puede aplicar a muchos sectores e industrias diferentes. Son varias las disciplinas que han contribuido, con ideas, puntos de vista y técnicas, al desarrollo de la IA, al tiempo que la IA ha contribuido al desarrollo de ellas y otras muchas.

La manera en que opera una solución de IA varía en función de sus objetivos y aplicaciones. El abordaje de cada una suele ser tan disímil, que es difícil reconocerles un origen común.

A medida que la IA se ha vuelto más sofisticada, ha surgido una rama especializada que ha revolucionado la manera en que interactuamos con la tecnología y el mundo que nos rodea: el Procesamiento de Lenguaje Natural (NLP). Esta disciplina se centra en dotar a las máquinas de la capacidad de comprender, interpretar y generar lenguaje humano de manera natural. Desde asistentes virtuales hasta traductores automáticos, el NLP ha ampliado significativamente las capacidades de la IA en el ámbito del lenguaje.

## El español como idioma de la Inteligencia Artificial

El uso de estas tecnologías nos brinda una facilidad de comunicación cada vez mayor entre el usuario y los sistemas, haciendo más natural la interacción entre los mismos y abriendo un abanico de posibilidades en el conjunto de instrucciones que el usuario común puede dar.

Pero su efectividad depende en gran medida del lenguaje utilizado. Tras el chino mandarín, el español es la segunda lengua más hablada del planeta, su uso en la inteligencia artificial es limitado y apenas está presente en aproximadamente el 25% de modelos del lenguaje mundial. Esto se debe en parte a la escasa inversión y protagonismo del lenguaje Español en la inteligencia artificial y la ventaja del Inglés como lengua por excelencia.

Los algoritmos de IA necesitan aprender y lo hacen mayoritariamente en idioma Inglés, siendo éste el idioma en que mejor se comportan, mientras que el idioma Español o castellano

representa menos del 30% del mercado mundial de las tecnologías del procesamiento de lenguaje natural (Carretero, 2021).

Este contraste o retraso de nuestra lengua frente al Inglés no es atribuible a una única causa. Por un lado, obedece a una razón histórica y en estos días el idioma Inglés se ha consolidado como la lengua de investigación por excelencia.

Otras razones son la variabilidad y particularidades del idioma Español en sus variantes en Latinoamérica que pueden generar dificultades (el sarcasmo, las expresiones idiomáticas, la jerga, las metáforas, etc).

Si bien estos son elementos a tener en cuenta a la hora de introducir nuestro idioma y lenguaje al mundo de la IA, el factor determinante es la inversión y recursos para poder crear soluciones en español y no solo adaptadas a nuestro lenguaje. Por ejemplo, Siri, Cortana, Alexa y el asistente de Google hablan nuestro idioma. Pero no es su lengua materna. Esto se refleja en que su rendimiento y precisión son notablemente superiores cuando se utilizan en inglés, el idioma en el que fueron originalmente diseñados.

A pesar de todo, nos encontramos hoy en un momento de interés creciente por el desarrollo de la inteligencia artificial aplicada al lenguaje. Ha habido un incremento del 34,5 % de trabajos científicos sobre procesamiento del lenguaje natural e IA aplicado a lenguaje entre 2019 y 2020, que hace patente la creciente madurez de la tecnología (González-Blanco, n.d.).

## Procesamiento de lenguaje natural (NLP)

### Definición

El procesamiento del lenguaje natural (NLP) es una rama de la inteligencia artificial (IA) que permite a las computadoras comprender, generar y manipular el lenguaje humano. Es el conjunto de métodos para hacer que el lenguaje humano sea accesible para las computadoras.

El procesamiento del lenguaje natural se ha incorporado a la vida cotidiana. La traducción automática está omnipresente en la web y en las redes sociales, la clasificación de texto evita spam en nuestros correos electrónicos, los motores de búsqueda han avanzado más allá de la coincidencia de cadenas hacia un alto grado de sofisticación lingüística, los sistemas de diálogo proporcionan una forma cada vez más común y efectiva de obtener y compartir información.

Estas diversas aplicaciones se basan en el procesamiento de lenguaje natural (NLP) el cual se nutre de muchas otras ideas que van desde la lingüística formal hasta la algorítmica y estadística.

*Lingüística Computacional*: Muchos autores o documentos que hablan sobre procesamiento del lenguaje natural tienen el concepto o llevan el nombre de "lingüística computacional", y los términos pueden considerarse esencialmente sinónimos. Pero mientras hay una considerable superposición entre estos conceptos, debido a una diferencia importante en el enfoque. En lingüística, el lenguaje es el objeto de estudio. Los métodos computacionales pueden ser empleados como formas de apoyo. En cambio, el procesamiento del lenguaje natural se centra en el diseño y análisis de algoritmos y representaciones computacionales para procesar el lenguaje humano natural. El objetivo del procesamiento del lenguaje natural es proporcionar nuevas capacidades computacionales en torno al lenguaje humano: por ejemplo, extraer información de textos, traducir idiomas, responder preguntas, sostener conversaciones, seguir instrucciones, entre otros. Los conocimientos lingüísticos fundamentales pueden ser cruciales para realizar estas tareas, pero el éxito se mide en última instancia por sí y cuán bien se realiza el trabajo.

*Aprendizaje Automático (Machine Learning)*: Los enfoques en el procesamiento del lenguaje natural dependen en gran medida del aprendizaje automático. El aprendizaje automático proporciona una variedad de técnicas o métodos para tareas como convertir una secuencia de tokens discretos en un vocabulario a otra secuencia de tokens discretos en otro vocabulario, una generalización de lo que se podría llamar informalmente "traducción". Gran parte de la investigación actual en procesamiento del lenguaje natural puede considerarse como aprendizaje automático aplicado. Sin embargo, el procesamiento del lenguaje natural tiene características que lo distinguen de muchos de los otros dominios de aplicación del aprendizaje automático.

A diferencia de las imágenes o el audio, los datos de texto son fundamentalmente discretos, con significado creado por disposiciones combinatorias de unidades simbólicas. Esto es particularmente importante para aplicaciones en las que el texto es el resultado, como la traducción y la sumarización, porque no es posible aproximarse gradualmente a una solución óptima.

Aunque el conjunto de palabras es discreto, siempre se están creando nuevas palabras. Además, la distribución sobre las palabras (y otros elementos lingüísticos) se asemeja a la de una cola de palabras que son raras. Una consecuencia es que los algoritmos de procesamiento del lenguaje natural deben ser especialmente robustos frente a observaciones que no ocurren en los datos de entrenamiento.

El lenguaje es compositivo, esto quiere decir que unidades como las palabras pueden combinarse para crear frases, que pueden combinarse según los mismos principios para crear frases más grandes. Por ejemplo, una frase nominal puede ser creada combinando una frase nominal más pequeña con una frase preposicional, como en "la blancura de la ballena". La

frase preposicional se crea combinando una preposición (en este caso, "de") con otra frase nominal (la ballena). De esta manera, es posible crear frases arbitrariamente largas, como:

*"... piezas globulares enormes de la ballena del tamaño de una cabeza humana"*

El significado de tal frase debe ser analizado de acuerdo con la estructura jerárquica subyacente. En este caso, "piezas globulares enormes de la ballena" actúa como una sola frase nominal, que se conjuga con la frase preposicional "del tamaño de una cabeza humana". La interpretación sería diferente si en su lugar "piezas globulares enormes" se conjugará con la frase preposicional "de la ballena del tamaño de una cabeza humana", lo que implicaría una ballena de tamaño irreal y carece de sentido. Aunque el texto aparece como una secuencia, los métodos de aprendizaje automático deben tener en cuenta su estructura recursiva implícita.

*Inteligencia Artificial:* El objetivo de la inteligencia artificial es construir software y robots con el mismo rango de habilidades que los humanos (Russell y Norvig, 2009). El procesamiento del lenguaje natural es relevante para este objetivo de varias maneras. En el nivel más básico, la capacidad del lenguaje es una de las características centrales de la inteligencia humana y, por lo tanto, es un requisito para la IA. En segundo lugar, gran parte de la investigación en inteligencia artificial se dedica al desarrollo de sistemas que pueden razonar desde premisas hasta una conclusión (Hubert Dreyfus 1994). El procesamiento del lenguaje natural es una solución potencial para el "cuello de botella del conocimiento", adquiriendo conocimiento de textos y, posiblemente, también de conversaciones. Esta idea se remonta al artículo "Computing Machinery and Intelligence", que propuso la prueba de Turing para determinar si se había logrado la inteligencia artificial (Turing, 1950).

Las tecnologías de procesamiento del lenguaje natural (NLP) tiene la capacidad de procesar el lenguaje humano en forma de texto o voz y "comprender" su significado, junto con la intención y el sentimiento del mensaje.

El lenguaje humano está lleno de ambigüedades que hacen difícil determinar con precisión el significado deseado de los mensajes (ya sea texto o voz). Los homónimos, los homófonos, el sarcasmo, las expresiones idiomáticas, la jerga, las metáforas, las excepciones gramaticales y su uso o las variaciones en la estructura de la oración son sólo algunas de las irregularidades del lenguaje que los humanos tardan años en aprender. Las tecnologías de NLP son capaces de resolver estas diferencias.

El razonamiento es esencial para tareas básicas de procesamiento del lenguaje, como resolver un pronombre o una anáfora.

*"La notebook no cabe en el bolso azul porque es demasiado [pequeña/grande]."*

Cuando la última palabra es "pequeña", entonces el pronombre se refiere al bolso; cuando la última palabra es "grande", entonces se refiere a la notebook. Resolver este ejemplo requiere un razonamiento; otros esquemas requieren razonamiento sobre acciones y sus efectos, emociones e intenciones, y convenciones sociales.

Tales ejemplos demuestran que la comprensión del lenguaje natural no se puede lograr de forma aislada del conocimiento y el razonamiento.

Por ejemplo, NLP es la tecnología principal detrás de los asistentes virtuales, como Oracle Digital Assistant (ODA), Siri, Cortana o Alexa. Cuando hacemos preguntas sobre estos asistentes virtuales, NLP es lo que les permite no sólo comprender la solicitud del usuario, sino también responder con lenguaje natural. NLP se aplica tanto al texto escrito como al habla y busca ser aplicado para todos los idiomas.

Otros ejemplos de herramientas con tecnología NLP incluyen búsqueda web, filtrado de correo electrónico no deseado, traducción automática de texto o voz, resumen de documentos, análisis de sentimientos y comprobación de ortografía y gramática.

*Ciencia de la Computación:* La naturaleza discreta y recursiva del lenguaje natural se presta para la formulación y aplicación de ideas teóricas de la ciencia de la computación. Algunos lingüistas han demostrado cómo la teoría formal del lenguaje puede ayudar a explicar la sintaxis y la semántica del lenguaje natural (Chomsky N., 1957; Montague R., 1970). Modelos teóricos (como por ejemplo, autómatas finitos) son la base de muchos sistemas prácticos de procesamiento del lenguaje formal. El estudio de los sistemas informáticos también es relevante para el procesamiento del lenguaje natural.

*Speech Processing:* El lenguaje natural se comunica a menudo en forma hablada, y el reconocimiento del habla es la tarea de convertir una señal de audio a texto. Desde una perspectiva, este es un problema de procesamiento de señales, que podría considerarse como un paso de preprocesamiento antes de que se pueda aplicar el procesamiento del lenguaje natural. Sin embargo, el contexto juega un papel crítico en el reconocimiento del habla por parte de los oyentes humanos: el conocimiento de las palabras circundantes influye en la percepción y ayuda a corregir el ruido (Miller G.A., 1951). Por esta razón, el reconocimiento del habla a menudo se integra con el análisis de texto.

*Otros:* El procesamiento del lenguaje natural juega un papel significativo en campos interdisciplinarios emergentes como la ciencia computacional social y las humanidades digitales. La clasificación de texto, el agrupamiento y la extracción de información. La recuperación de información emplea herramientas y técnicas afines, como el análisis semántico, el cual se encuentra estrechamente vinculado con este campo.

## Procesamiento del lenguaje

Escribir patrones de respuesta para determinadas consultas es un procedimiento complejo, ya que cadenas muy similares pueden significar cosas muy distintas, mientras que cadenas muy diferentes pueden significar casi lo mismo (ambigüedad). Por ejemplo:

1. ¿Qué tan rápido es la X?
2. ¿Cuándo llegará mi X?
3. Por favor, dime cuándo puedo esperar el X que ordené.

Las consultas 1 y 2 se ven muy similares, pero significan algo completamente diferente, mientras que las 2 y 3 se ven muy diferentes, pero significan casi lo mismo.

Algunas tareas en NLP se pueden hacer adecuadamente sin tener en cuenta el significado, otras requieren que podamos construir representaciones detalladas que reflejen el significado subyacente en lugar del significado literal. De hecho, en los lenguajes naturales (a diferencia de los lenguajes de programación), la ambigüedad es omnipresente, por lo que exactamente la misma cadena puede significar cosas diferentes. Por ejemplo:

- ¿Venden laptops y discos duros de X?

El usuario puede estar preguntando o no sobre discos duros "X". Esta ambigüedad particular puede ser representada por diferentes estructuras, como por ejemplo:

- ¿Venden (laptops de X) y (discos duros)?
- ¿Venden (X (laptops y discos duros))?

Este tipo de ambigüedades existen constantemente en el mundo real. Los humanos tienen conocimiento del mundo que resuelve una posible ambigüedad, probablemente sin que el hablante o el oyente siquiera sean conscientes de que existe una ambigüedad. Pero codificar dicho conocimiento en aplicaciones de procesamiento del lenguaje natural ha resultado ser muy complejo.

## Funcionamiento del procesamiento de lenguaje natural (NLP)

Los modelos de NLP funcionan encontrando relaciones entre las partes constituyentes del lenguaje (por ejemplo, las letras, artículos, pronombres, anáforas, palabras y oraciones que se encuentran en un texto). Las arquitecturas del procesamiento de lenguaje natural utilizan varios métodos para el preprocesamiento de datos, la extracción de características y el modelado. Actualmente los procesos más utilizados son:

## Data Processing

Data preprocessing se refiere al proceso de preparar los datos para su uso en un modelo de aprendizaje automático. En el caso del procesamiento de texto, esto puede incluir la eliminación de caracteres no deseados, la normalización de texto, la eliminación de palabras vacías y la tokenización. El objetivo del preprocesamiento de datos es mejorar la calidad de los datos y hacer que sean más fáciles de procesar para el modelo de aprendizaje automático.

Antes de que un modelo procese texto para una tarea específica, a menudo es necesario preprocesar el texto para mejorar el rendimiento del modelo o para convertir palabras y caracteres en un formato que el modelo pueda entender. La *inteligencia artificial centrada en los datos* es un movimiento creciente que prioriza el preprocesamiento de datos. Se pueden utilizar diversas técnicas en este preprocesamiento de datos.

## Stemming

Stemming (Derivación) es un proceso de reducción de palabras a su raíz, eliminando sufijos o prefijos para obtener la forma básica de la palabra. Este proceso ayuda a normalizar y simplificar las palabras, lo que puede ser útil en tareas como la recuperación de información, la clasificación de texto y la minería de texto. Sin embargo, el stemming no siempre produce resultados perfectos, y puede haber casos en los que falle o genere resultados no deseados o inexactos.

Por ejemplo:

- *Éxito*: La raíz de las palabras «running», «runner» y «runs» es «run».
- *Fallo*: Palabra original: "saw". Raíz esperada (según el stemming): "saw" (asumiendo que se reduciría a la forma base "saw" en lugar de la forma base "see")

En este caso, el algoritmo de stemming podría no reconocer que "saw" es la forma pasada de "see" y no lograr reducirlo correctamente a la forma base "see". Esto es un fallo en el sentido de que el stemming no produce la forma base esperada, sino que mantiene la palabra original sin cambios.

## Lemmatization

La lematización es un proceso similar al *stemming*, pero implica convertir una palabra a su forma base, llamada lema, utilizando un análisis de diccionario.

Al derivar, una parte de la palabra simplemente se corta en el final para llegar a la raíz de la palabra. Se utilizan diferentes algoritmos para determinar cuántos caracteres deben eliminarse, pero en realidad no conocen el significado de la palabra en el idioma al que pertenece. En lematización, los algoritmos sí tienen este conocimiento. Se refieren a un diccionario para comprender el significado de la palabra antes de reducirla a su raíz, o lema.

Un algoritmo de lematización sabría que la palabra "better" se deriva de la palabra "good" y, por lo tanto, el lema es "good". Pero un algoritmo de derivación no podría hacer lo mismo.

Podría haber una derivación excesiva o insuficiente, y la palabra "better" podría reducirse a "bet", o "bett", o simplemente conservarse como "better". Pero no hay manera de derivar que pueda reducir "better" a su raíz; la palabra "good".

#### Sentence segmentation

Divide una gran porción de texto en unidades de oración lingüísticamente significativas. Esto es obvio en idiomas como el inglés, donde el final de una oración está marcado por un punto, pero aún así no es trivial. Se puede utilizar un punto para marcar una abreviatura así como para terminar una oración y, en este caso, el punto debe ser parte de la propia abreviatura. El proceso se vuelve aún más complejo en idiomas que no tienen un delimitador que marque el final de una oración.

Por ejemplo:

- Éxito: "La reunión comenzó puntualmente a las 9:00 am. Todos los miembros del equipo estaban presentes."

Segmentación correcta:

1. "La reunión comenzó puntualmente a las 9:00 am."
2. "Todos los miembros del equipo estaban presentes."

- Fallo: "La casa está ubicada en la calle principal No. 123, cerca del parque. Es una casa muy bonita."

Segmentación incorrecta:

1. "La casa está ubicada en la calle principal No."
2. "123, cerca del parque."
3. "Es una casa muy bonita."

#### Stop word removal

Las "palabras vacías" son palabras muy comunes que generalmente se eliminan del texto antes de procesarlo, ya que no aportan un significado específico al análisis del texto y pueden aparecer con una alta frecuencia en diferentes documentos sin ofrecer mucha información útil sobre el contenido del texto.

Algunos ejemplos de palabras vacías en inglés incluyen "the", "is", "and", "in", "of", "to", entre otros. Estas palabras son fundamentales para la gramática y la estructura de las oraciones, pero por sí solas no proporcionan mucha información semántica sobre el contenido del texto.

Es importante tener en cuenta que la lista de palabras vacías puede variar según el idioma y el contexto del problema específico que se esté abordando en NLP. Además, en algunos casos,

es posible que ciertas palabras vacías deban conservarse dependiendo del contexto del análisis.

### Tokenization

La tokenización consiste en dividir un texto en unidades más pequeñas llamadas tokens. Estos tokens pueden ser palabras individuales, subpalabras, caracteres o incluso frases completas, dependiendo del nivel de granularidad.

El resultado generalmente consiste en un índice de palabras y un texto tokenizado en el que las palabras pueden representarse como tokens numéricos para su uso en diversos métodos de aprendizaje. Existen métodos que indican a los modelos de lenguaje que ignoren tokens sin importancia para mejorar la eficiencia (Figura 1).

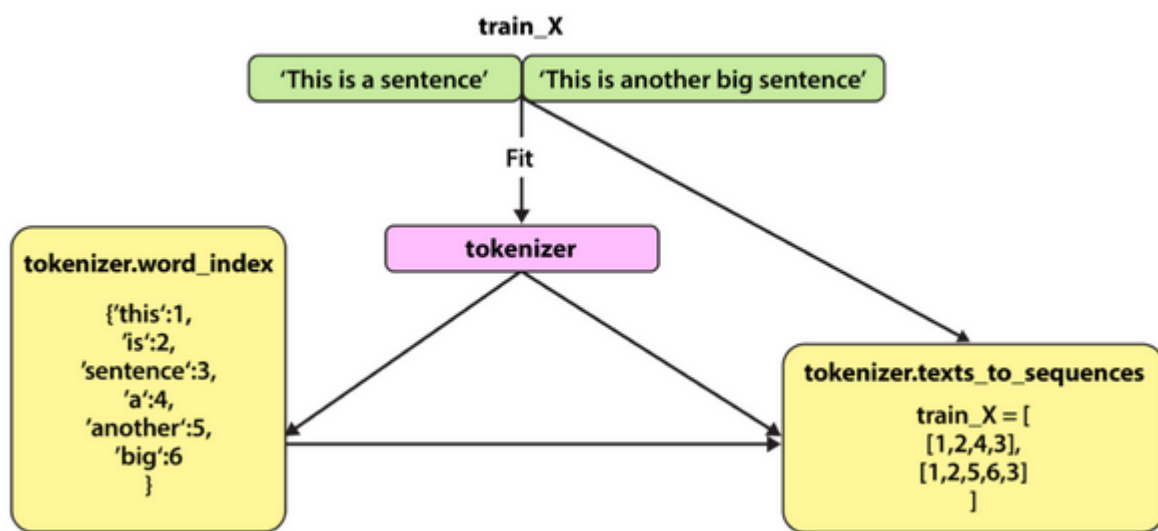


Figura 1. La tokenización divide un texto en unidades más pequeñas llamadas "tokens". Como parámetro de entrada se ingresa un texto y se obtiene como salida un índice de palabras y un texto tokenizado (DeepLearning.AI, 2023).

### Extracción de características

La "Extracción de características" es una técnica que se utiliza para identificar y extraer características relevantes de un texto. En NLP, esto se refiere a la identificación de patrones en el texto que puedan ser útiles para la tarea en cuestión, como la clasificación de texto o la identificación de temas. Por ejemplo, se pueden extraer características como la frecuencia de las palabras, la longitud de las oraciones o la presencia de ciertas palabras clave.

### Bag-of-Words

Bag-of-Words (BOW) cuenta el número de veces que aparece cada palabra o n-grama (combinación de n palabras) en un texto. Por ejemplo, a continuación, el modelo Bolsa de

palabras crea una representación numérica del conjunto de datos en función de cuántas ocurrencias de cada palabra en el “índice de palabras” aparecen en el texto (Figura 2).

Representamos el texto simplemente por la frecuencia de cada palabra. Por ejemplo, si tenemos un vocabulario de 500 palabras, entonces todo el texto estará representado por un vector de 500 dimensiones.

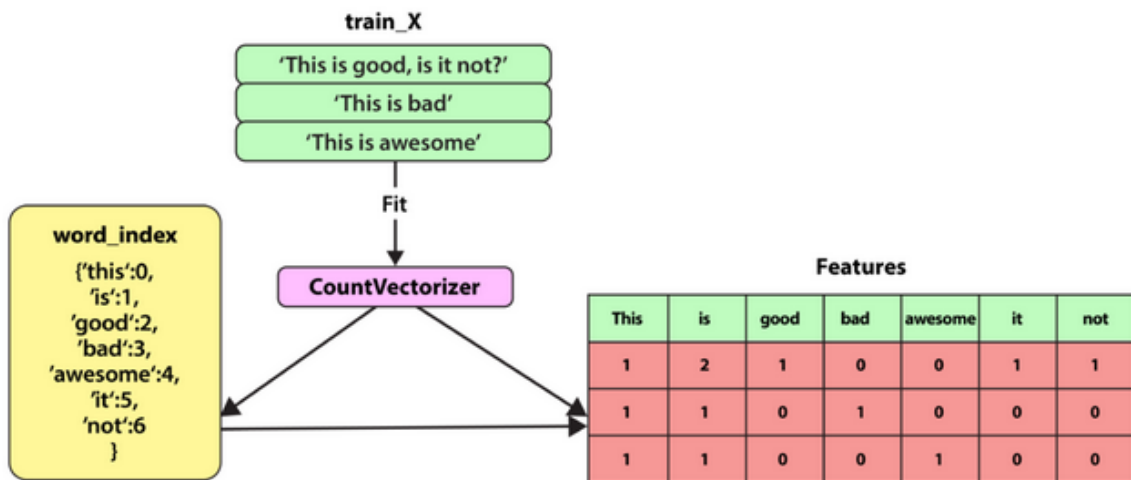


Figura 2. Bag of Words o Bolsa de Palabras cuantifica el número total de apariciones de cada palabra en un texto. Crea una representación numérica del conjunto de datos en función de cuántas ocurrencias de cada palabra en el “índice de palabras” aparecen en el texto (DeepLearning.AI, 2023).

Utilizando esta técnica, podemos incrustar un conjunto de textos o documentos e introducirlos en una variedad de algoritmos de aprendizaje automático diferentes. Dado que esta integración es tan básica, no funciona muy bien para tareas complejas. Pero sí para problemas de clasificación.

Los principales beneficios de este modelo son su simplicidad y eficiencia; este método es fácil de implementar y rápido de ejecutar.

### TF-IDF

En la metodología Bag-of-Words, contamos la aparición de cada palabra o n-grama en un documento o un texto. Por el contrario, con TF-IDF, ponderamos cada palabra según su importancia. Para evaluar el significado de una palabra, TF-IDF considera:

- **Frecuencia de términos (TF):** ¿Qué importancia tiene la palabra en el documento?  
 $TF(\text{palabra en un documento}) = \frac{\text{Número de apariciones de esa palabra en el documento}}{\text{Número de palabras en el documento}}$
- **Frecuencia inversa del documento (IDF):** ¿Qué importancia tiene el término en todo el corpus?

$IDF(\text{palabra en un corpus}) = \log(\text{número de documentos en el corpus} / \text{número de documentos que incluyen la palabra})$

Al calcular TF, todos los términos se consideran igualmente importantes. Sin embargo, ciertos términos, como “es”, “de” y “eso” o en inglés "a" y "the", pueden aparecer muchas veces pero tienen poca importancia. Necesitamos reducir el peso de los términos frecuentes y ampliar los términos "raros". Cuando calculamos IDF, se incorpora un factor de frecuencia de documentos inverso, que disminuye el peso de los términos que ocurren con mucha frecuencia en el conjunto de documentos y aumenta el peso de los términos que rara vez ocurren (Figura 3).

La puntuación TF-IDF de un término es el producto de TF e IDF.

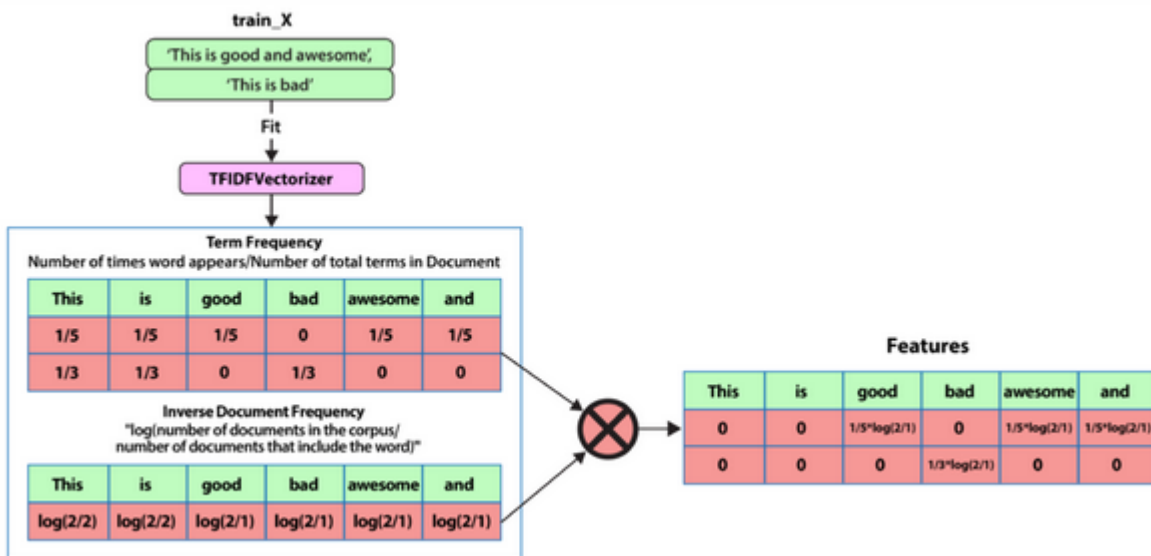


Figura 3. TF-IDF crea funciones para cada documento en función de la frecuencia con la que aparece cada palabra en un documento en comparación con todo el conjunto de documentos (DeepLearning.AI, 2023).

### Word2Vec

Utilizando una red neuronal básica, word2vec intenta aprender las relaciones entre palabras y las incrusta en un espacio vectorial de dimensiones inferiores (Mikolov T. et al., 2013). Existen dos variaciones de este método.

- Skip-Gram: En este modelo se intenta predecir las palabras circundantes (contexto) dada una palabra objetivo.

La arquitectura del modelo skip-gram consta de una capa de entrada, una capa de salida y una capa oculta. La capa de entrada son las palabras que se van a predecir y la capa de salida son las palabras de contexto. La capa oculta representa la incorporación de la palabra de entrada aprendida durante el entrenamiento (Figura 4).

Source Text	Training Samples generated from source text
I will have orange juice and eggs for breakfast	(will, I) (will, have) (will, orange)
I will have orange juice and eggs for breakfast	( have, I) (have, will) (have, orange) (have, juice)
I will have orange juice and eggs for breakfast	(orange, will) (orange, have) (orange, juice) (orange, and)
I will have orange juice and eggs for breakfast	(juice, have) (juice, orange) (juice, and) (juice, eggs)
I will have orange juice and eggs for breakfast	(and, orange) (and, juice) (and, eggs) (and, for)
I will have orange juice and eggs for breakfast	(eggs, juice) (eggs, and) (eggs, for) (eggs, breakfast)
I will have orange juice and eggs for breakfast	( for, and) ( for, eggs) ( for, breakfast)

Figura 4. Ejemplo de cómo se generan pares de palabras o "n-grams" a partir de una oración fuente. Se seleccionan pares de palabras consecutivas y no consecutivas de la frase para capturar las relaciones contextuales entre palabras (Kulshrestha, 2019).

Dada la frase:

*"I will have orange juice and eggs for breakfast."*

Suponiendo un tamaño de ventana de 2, si la palabra objetivo es "juice", sus palabras vecinas serían (have, orange, and, eggs). Los pares de palabras de entrada y objetivo serían (juice, have), (juice, orange), (juice, and), (juice, eggs). También hay que tener en cuenta que dentro de la ventana de muestra, la proximidad de las palabras a la palabra objetivo no tiene ningún peso. Por lo tanto, "have", "orange", "and", "eggs" serán tratados de la misma manera durante el entrenamiento.

- Bolsa de palabras continúa (CBOW): En este modelo se intenta predecir la palabra objetivo a partir de las palabras circundantes o de su contexto.

La tarea falsa en CBOW es algo similar a Skip-gram, en el sentido de que también tomamos un par de palabras y le enseñamos al modelo que coexisten, pero en lugar de agregar los errores, agregamos las palabras de entrada para la misma palabra objetivo.

*Skip-gram*: funciona bien con una pequeña cantidad de datos de entrenamiento, representa bien incluso palabras o frases raras.

*CBOW*: varias veces más rápido de entrenar que el skip-gram, precisión ligeramente mejor para las palabras frecuentes.

## GLoVe

Este modelo es similar a Word2Vec porque también aprende incrustaciones de palabras, pero lo hace utilizando técnicas de factorización matricial en lugar de aprendizaje neuronal. El modelo GLoVe (Pennington J. et al., 2014) construye una matriz basada en los recuentos globales de concurrencia palabra por palabra.

## Modelado (Modeling)

Una vez pre procesados los datos, se introducen en una arquitectura de NLP que modela los datos para realizar una variedad de tareas.

Los modelos del lenguaje permiten a los ordenadores comprender contextualmente los matices del lenguaje. Son marcos informáticos diseñados para comprender, crear y predecir el lenguaje humano. Estos modelos, que se entrenan a partir de una gran cantidad de datos textuales, imitan la generación de textos de tipo humano al captar patrones, relaciones entre palabras y usos contextuales. Aprovechan los métodos estadísticos y probabilísticos para identificar las estructuras y significados de los textos, con habilidades como autocompletar frases, traducir idiomas, analizar sentimientos, resumir contenidos y elaborar párrafos coherentes.

Estos modelos aprenden cómo funciona el lenguaje y sus reglas estudiando grandes volúmenes de datos (textos, como libros, artículos o conversaciones). De este modo, pueden averiguar qué palabras suelen ir después de otras (patrones), cómo se forman las frases (gramática) y qué tiene sentido en distintos contextos.

Utilizan algoritmos complejos que se basan en probabilidades y patrones observados en las enormes cantidades de texto con las que se han entrenado. Por ejemplo, determinan la probabilidad de que aparezcan secuencias de palabras basándose en las ocurrencias en los datos de entrenamiento (n-grams). También analizan las frecuencias de las secuencias de palabras, comprendiendo su uso contextual (como en BoW y TF-IDF), con lo que aprenden la sintaxis, la semántica y las relaciones que se forman entre ellas.

## Procesamiento de lenguaje natural y sus técnicas

La mayoría de las tareas de procesamiento de lenguaje natural pueden modelarse con diferentes técnicas generales de NLP. Estas técnicas se pueden ver en dos categorías: métodos tradicionales de aprendizaje automático y métodos de aprendizaje profundo.

### Técnicas tradicionales de NLP de aprendizaje automático

#### Regresión Logística

La regresión logística es un algoritmo de clasificación supervisado que tiene como objetivo predecir la probabilidad de que ocurra un evento en función de alguna entrada.

El método de regresión logística es un método estadístico que se usa para resolver problemas de clasificación binaria, donde el resultado solo puede ser de naturaleza dicotómica, o sea, solo puede tomar dos valores posibles. Es uno de los métodos más simples y más utilizados para la clasificación de dos clases.

Para la regresión logística, hay que formular la pregunta para obtener resultados concretos:

*¿Los días de lluvia afectan nuestras ventas mensuales? (sí o no)*

En NLP, los modelos de regresión logística se pueden aplicar para resolver problemas como el análisis de sentimientos, la detección de spam y la clasificación de toxicidad.

La regresión logística es un modelo estadístico que utiliza la función logística, o función logit que se describe con la siguiente fórmula:

$$f(x) = \frac{1}{1 + e^{-x}}$$

La función logit devuelve solo valores entre 0 y 1 para la variable dependiente, al margen de los valores de la variable independiente (Figura 5).

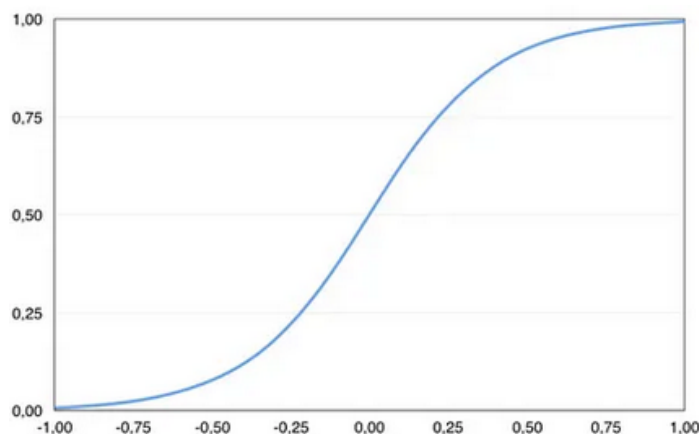


Figura 5. Gráfico de ejes cartesianos que representa la función “logit”. La función logística produce valores entre 0 y 1.

La función logística produce valores entre 0 y 1, lo que la hace adecuada para modelar la probabilidad de una respuesta binaria para la ocurrencia de un evento. La regresión logística utiliza la función logística para modelar la relación entre las variables independientes y la variable dependiente. El modelo logístico binario se utiliza para estimar la probabilidad de una respuesta binaria basada en una o más variables independientes.

Definiendo que lo que se desea inicialmente en un problema de clasificación binaria es identificar la probabilidad ( $p$ ) de que un evento ocurra. La relación entre esta probabilidad y

los eventos (variables) independientes está descrita por la siguiente fórmula planteada en la Figura 6.

$$\text{logit}(p) = \log \frac{p}{1-p} = \beta_0 + \beta_1 x_1 + \beta_2 x_2 + \dots + \beta_n x_n$$

Figura 6. Fórmula Matemática completa de Regresión Logística.

Despejando matemáticamente podemos obtener la siguiente ecuación para la probabilidad de ocurrencia de un evento (Figura 7).

$$p = \frac{1}{1 + e^{-(\beta_0 + \beta_1 x_1 + \beta_2 x_2 + \dots + \beta_n x_n)}}$$

Figura 7. Fórmula matemática resumida o breve de Regresión Logística

En la ecuación **logit(p)** vista anteriormente, lo que traduce es que podemos relacionar la probabilidad de que ocurra un evento y la ocurrencia de unos eventos independientes.

#### Naive Bayes

Naive Bayes es un algoritmo de clasificación supervisado basado en el teorema de Bayes, que es una técnica estadística que utiliza la probabilidad para hacer predicciones. Este algoritmo es simple pero efectivo y se utiliza ampliamente en diversas áreas del Machine Learning.

El teorema de Bayes es un concepto estadístico que nos permite obtener probabilidades sobre un evento dadas nuevas evidencias. Esta trabaja para llegar a una hipótesis (A) a partir de un conjunto dado de evidencia (B). Se relaciona con, la probabilidad de que ocurra la hipótesis antes de la evidencia P(A) y la probabilidad después de la evidencia P(A|B). La teoría de Bayes se explica mediante la siguiente ecuación planteada en la Figura 8.

$$P(A|B) = \frac{P(A) \times P(B|A)}{P(B)}$$

Figura 8. Ecuación matemática del Teorema de Bayes.

- P(A|B) denota cómo ocurre el evento A cuando tiene lugar el evento B.

- $P(B|A)$  representa la frecuencia con la que ocurre el evento B cuando el evento A ocurre primero.
- $P(A)$  representa la probabilidad de que el evento X ocurra por sí solo.
- $P(B)$  representa la probabilidad de que el evento Y ocurra por sí solo.

En NLP, estos métodos estadísticos se pueden aplicar para resolver problemas como la detección de spam o la búsqueda de errores en el código del software.

### Árboles de Decisión

Un árbol de decisión es un algoritmo de aprendizaje supervisado no paramétrico, que se utiliza tanto para tareas de clasificación como de regresión. Tiene una estructura de árbol jerárquica, que consta de un nodo raíz, ramas, nodos internos y nodos hoja.

- *Nodo raíz*: Es el primer nodo en el árbol y representa toda la muestra de datos de entrenamiento. Su función es seleccionar la característica (o atributo) que mejor divide los datos en función de ciertos criterios.
- *Nodos internos (o de decisión)*: Estos nodos representan las subdivisiones del conjunto de datos realizadas por las características seleccionadas en los nodos anteriores. Cada nodo interno evalúa una característica específica y divide los datos en función de un umbral determinado. Su función es seguir dividiendo el conjunto de datos en subconjuntos más homogéneos o más puros, de manera iterativa.
- *Nodos hoja (o nodos terminales)*: Son los nodos finales del árbol, donde ya no se realizan más divisiones. Representan las categorías finales o valores de predicción.

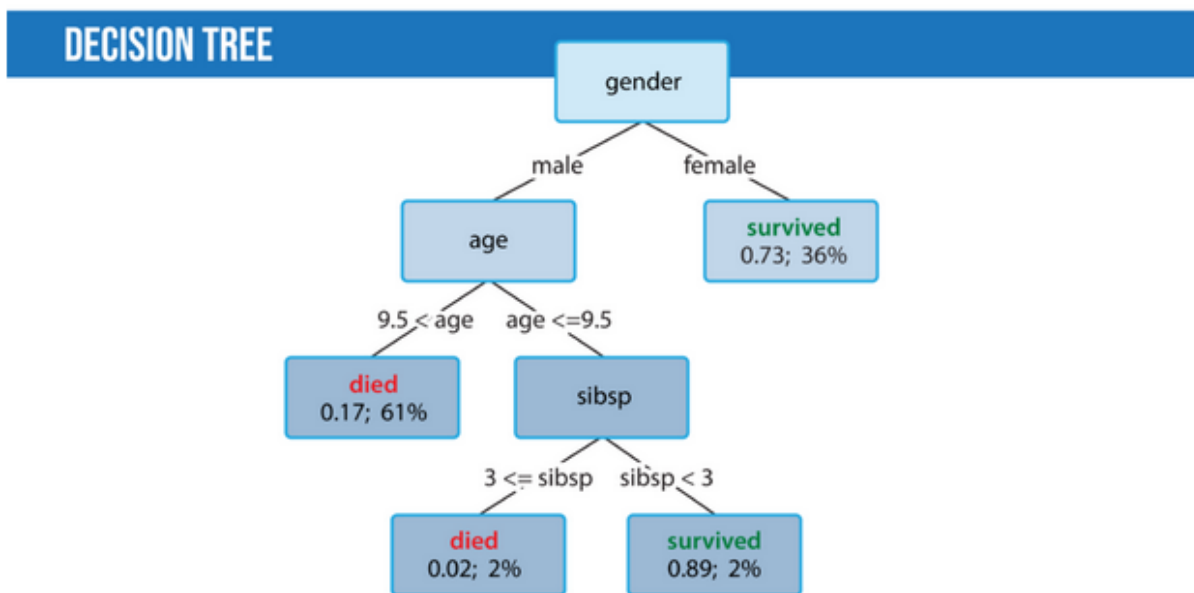


Figura 9. Ejemplo de árbol de decisión que predice la supervivencia de los pasajeros del Titanic en función del género, la edad y el número de familiares a bordo. Las mujeres tenían una mayor probabilidad de sobrevivir (73%), mientras que los hombres jóvenes con menos de tres familiares también mostraban una alta probabilidad de supervivencia (89%).

### Asignación de Dirichlet latente (LDA)

La asignación de Dirichlet latente (LDA) es un modelo probabilístico para el análisis temático y segmentación de textos en el campo del aprendizaje automático y procesamiento de lenguaje natural.

El modelo LDA tiene tres niveles principales:

- *Temas*: Son conjuntos de palabras relacionadas entre sí que aparecen juntas frecuentemente en los documentos. Por ejemplo, un tema podría ser "Inteligencia Artificial".
- *Palabras*: Son las entradas individuales del corpus de datos, como palabras o términos específicos.
- *Asignaciones de temas*: Representan la probabilidad de que una palabra pertenezca a un determinado tema dentro de un documento.

La intuición detrás de esto es que podemos describir cualquier tema usando sólo un pequeño conjunto de palabras del corpus.

### Modelo oculto de Markov

Los modelos de Markov son modelos probabilísticos que deciden el siguiente estado de un sistema en función del estado actual.

El modelo oculto de Markov es un modelo generativo que usa una cadena de Markov para modelar la evolución de los estados ocultos y una distribución de probabilidad condicional para modelar la relación entre los estados ocultos y las observaciones.

Este modelo asume que las observaciones están relacionadas sólo con el estado actual del sistema y no dependen de estados anteriores o futuros. Por ejemplo, en NLP, podríamos sugerir la siguiente palabra basándonos en la palabra anterior. Podemos modelar esto como un modelo de Markov donde podríamos encontrar las probabilidades de transición de pasar de palabra1 a palabra2, es decir,  $P(\text{palabra1}|\text{palabra2})$ . Luego podemos usar un producto de estas probabilidades de transición para encontrar la probabilidad de una oración.

Un modelo oculto de Markov (HMM) es un conjunto finito de estados. Las transiciones entre estados están dadas por un conjunto de probabilidades de transición.

En cualquier estado particular, la observación puede ser generada, de acuerdo a la distribución de probabilidades de emisión. Sólo el resultado observable, no el estado, es visible a un observador externo por lo que los estados están "ocultos".

Ejemplo: Una persona intenta recordar el estado del clima de la semana anterior basándose en las actividades que realizó durante esos días. En este contexto, las acciones observadas representan los estados visibles, mientras que los distintos tipos de clima corresponden a los estados ocultos. Las transiciones entre los estados climáticos se representan mediante probabilidades de transición, mientras que las probabilidades de emisión describen la probabilidad de que una determinada acción haya sido realizada bajo un estado climático específico (Figura 10).

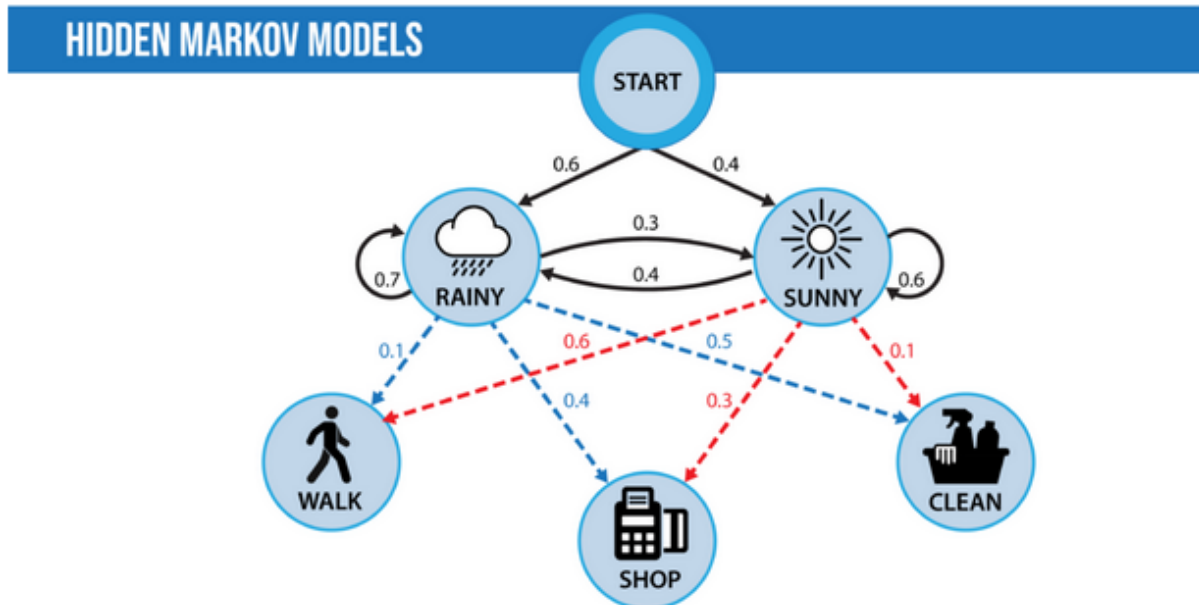


Figura 10. Ejemplo sobre el modelo oculto de Markov (DeepLearning.AI, 2023).

## Técnicas de NLP de aprendizaje profundo

### Red neuronal convolucional (CNN)

Una red neuronal convolucional (CNN) es un tipo especializado de algoritmo de aprendizaje profundo diseñado principalmente para tareas que requieren el reconocimiento de objetos, como la clasificación, detección y segmentación de imágenes.

Las CNN se distinguen de los algoritmos clásicos de aprendizaje automático, como los árboles de decisión, por su capacidad para extraer características de forma autónoma a gran escala, evitando la necesidad de ingeniería de características y mejorando así la eficiencia.

Las neuronas en una CNN funcionan de forma similar al cerebro humano. Se agrupan en capas para ir comprendiendo los diferentes niveles de abstracción. Primero se detectan los patrones más simples y a continuación los más complejos. Cada capa de neuronas de la red neuronal convolucional aprende a detectar las diferentes características de la entrada. Se aplica un filtro o núcleo (kernel) a la entrada para producir una salida, que se vuelve progresivamente mejor y más detallada después de cada capa. Los filtros iniciales son

generalmente funciones simples. En cada capa sucesiva los filtros son más complejos para verificar e identificar características que representan de forma única el objeto.

Con la convolución, la imagen de entrada pasa por un conjunto de filtros. A medida que cada filtro activa ciertas características de la entrada, hace su trabajo y pasa su salida al filtro de la siguiente capa. Cada capa aprende a identificar diferentes características y las operaciones terminan repitiéndose durante decenas, cientos o incluso miles de capas. Finalmente, todos los datos de entrada que avanzan a través de las múltiples capas de la CNN le permiten a la red neuronal convolucional identificar el objeto completo.

Los distintos tipos de capas de una CNN son:

#### *Convolucional*

La mayoría de los cálculos se realizan en esta capa, que es el componente central de una CNN. Una segunda capa convolucional puede seguir a la capa convolucional inicial. El proceso de convolución implica que un núcleo o filtro (kernel) dentro de esta capa se mueve a través de los campos receptivos de la entrada, verificando si una característica está presente en la imagen.

#### *Reducción o muestreo*

Al igual que la capa convolucional, esta también barre un núcleo o filtro a través de los datos de entrada, pero se diferencia en que reduce la cantidad de parámetros en la entrada y también cierta información. Como dato positivo cabe señalar que esta capa reduce la complejidad y mejora la eficiencia de la CNN.

#### *Completamente conectada*

En esta capa es donde ocurre la clasificación en función de las características extraídas en las anteriores. Aquí, completamente conectado significa que todas las entradas o nodos de una capa están conectados a cada unidad de activación o nodo de la siguiente capa.

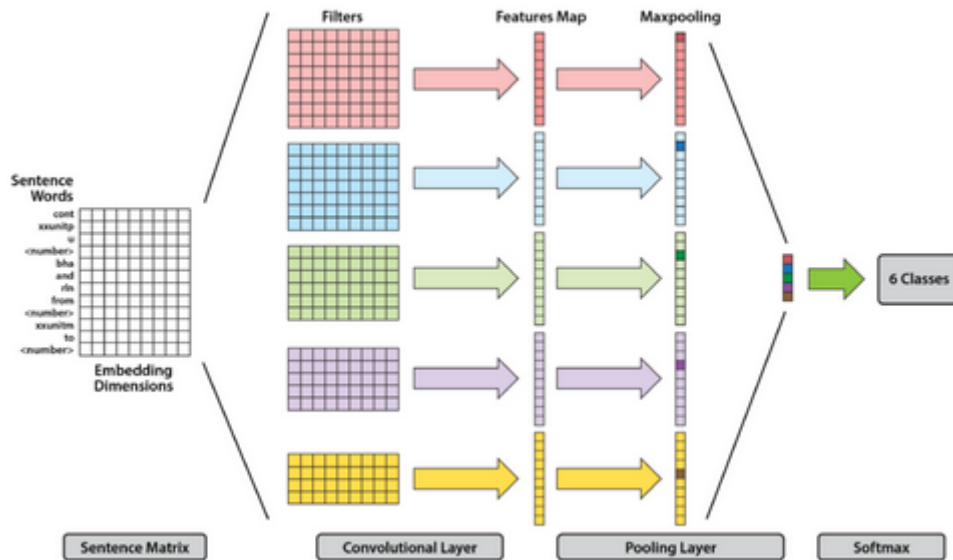


Figura 11. Ejemplo de cómo una red neuronal convolucional trabaja según sus parámetros de entrada. Dada una oración como entrada, una red neuronal convolucional utiliza diferentes capas convolucionales para refinar la representación de las palabras, antes de combinarlas y representarlas.

### Red neuronal recurrente (RNN)

Las redes neuronales recurrentes (RNN) son un tipo de redes neuronales artificiales especializadas en el procesamiento de datos secuenciales, como series temporales o datos textuales.

A diferencia de las redes neuronales tradicionales, las RNN presentan conexiones recurrentes que les permiten mantener un estado interno y memorizar información sobre entradas previas. Esto las hace ideales para tareas que requieren entender el contexto como traducción, reconocimiento del habla o análisis de sentimiento.

Se distinguen por su "memoria", ya que obtienen información de entradas anteriores para influir en la entrada y salida actuales. Los RNN recuerdan información anterior mediante estados ocultos y la conectan a la tarea actual.

Existen varios tipos de Redes Neuronales Recurrentes que se adaptan a distintos casos de uso:

- *RNN* unidireccionales: Sólo tienen en cuenta el contexto previo.
- *RNN* bidireccionales: Procesan la secuencia en ambos sentidos, teniendo en cuenta contexto futuro y pasado.
- *LSTM*: Resuelven problemas de largo plazo mediante una arquitectura más compleja.
- *GRU*: Similar a LSTM pero más simple y liviana.

- *RNN con atención*: Utilizan mecanismos de atención para enfocarse en partes relevantes.

Cada una presenta ventajas para ciertas aplicaciones. Por ejemplo, las LSTM son ideales para dependencias a largo plazo mientras que las GRU requieren menos recursos.

Las Redes Neuronales Recurrentes procesan una secuencia paso a paso. En cada paso, reciben un elemento de entrada y su estado anterior. A partir de esto, calculan un nuevo estado y una salida.

El estado actúa como una memoria que captura la información relevante de los elementos previos. Esta memoria se actualiza continuamente a medida que la red procesa nuevos datos (Figura 12).

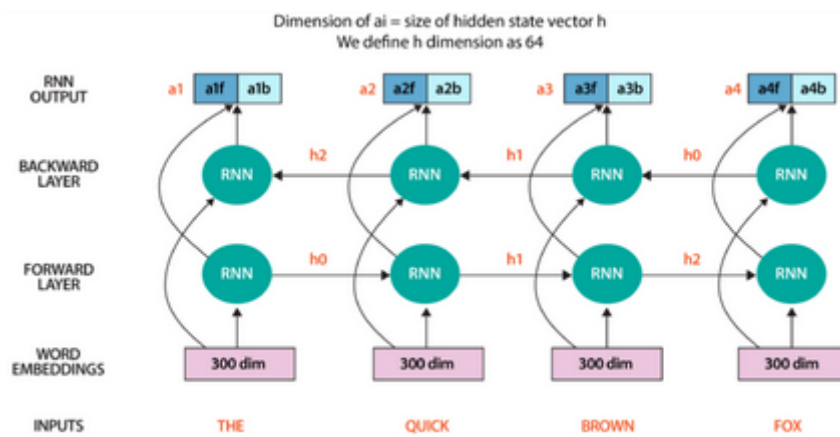


Figura 12. Una RNN procesa la entrada tanto hacia adelante como hacia atrás para mejorar la representación que produce (DeepLearning.AI, 2023).

### Auto-Encoder

Los auto-encoders son codificadores-decodificadores de *Deep Learning* que aproximan un mapeo de  $X$  a  $X$ , es decir, entrada=salida.

El codificador transforma los datos de entrada en una representación de dimensiones reducidas, a la que a menudo se hace referencia como "espacio latente" o "codificación". A partir de esa representación, un decodificador reconstruye la entrada inicial. Para que la red obtenga patrones significativos en los datos, un proceso de codificación y decodificación facilita la definición de características esenciales (Figura 13).

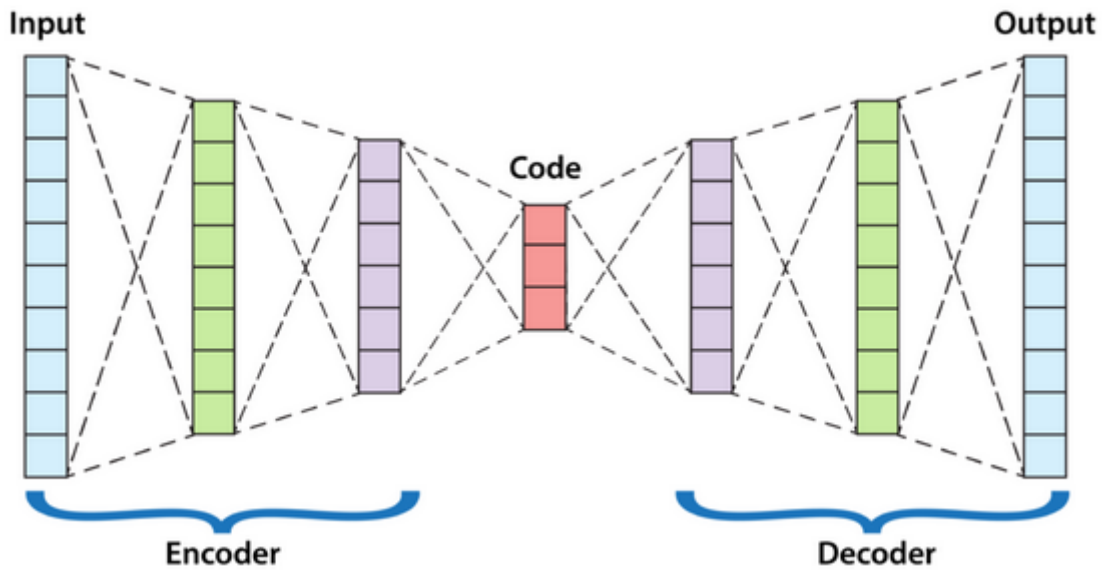


Figura 13. Un Auto Encoder utiliza un codificador para transformar una entrada en una representación, y un decodificador para construir la entrada a partir de esa representación codificada (DeepLearning.AI, 2023b).

#### Encoder-Decoder Seq2Seq

La arquitectura codificador-decodificador seq2seq (Sutskever, Vinyals, & Le, 2014) es una adaptación de los auto-encoder, especializados para traducción, resumen y tareas similares. El codificador encapsula la información de un texto en un vector codificado. A diferencia de un auto-encoder, en lugar de reconstruir la entrada del vector codificado, la tarea del decodificador es generar una salida deseada diferente, como una traducción o un resumen.

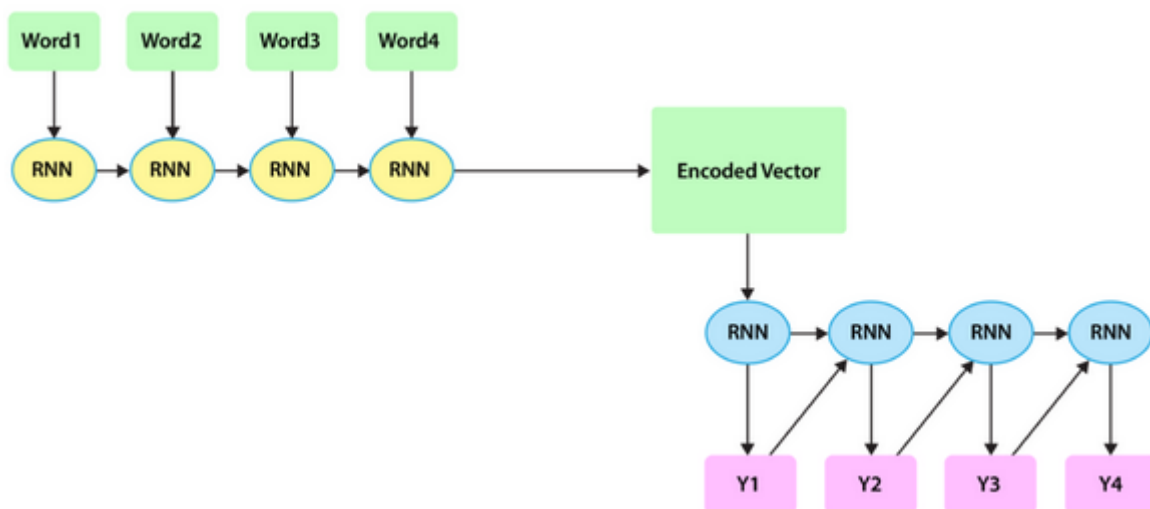


Figura 14. Ejemplo gráfico de cómo trabaja el algoritmo encoder-decoder seq2seq. Dada una oración, una RNN codifica la oración y luego genera traducciones de forma iterativa (DeepLearning.AI, 2023b).

## Transformers

Los Transformers son redes neuronales que aprenden contexto y comprensión a través del análisis de datos. Los modelos Transformer utilizan un conjunto de técnicas matemáticas modernas y en evolución, generalmente conocidas como *attention* o *self-attention*. Este conjunto ayuda a identificar cómo los elementos de datos distantes influyen y dependen unos de otros.

En un documento publicado en agosto de 2021, investigadores de Stanford identificaron a los Transformers como "modelos fundamentales" porque creen que transformarán la inteligencia artificial (Bommasani et al., 2021).

Los Transformers se inspiran en la arquitectura codificador-decodificador encontrada en las RNN debido a su mecanismo de atención. Pueden manejar tareas de secuencia a secuencia (seq2seq) eliminando el componente secuencial. Un Transformer, a diferencia de una RNN, no realiza el procesamiento de datos en orden secuencial, lo que permite una mayor paralelización y un entrenamiento más rápido.

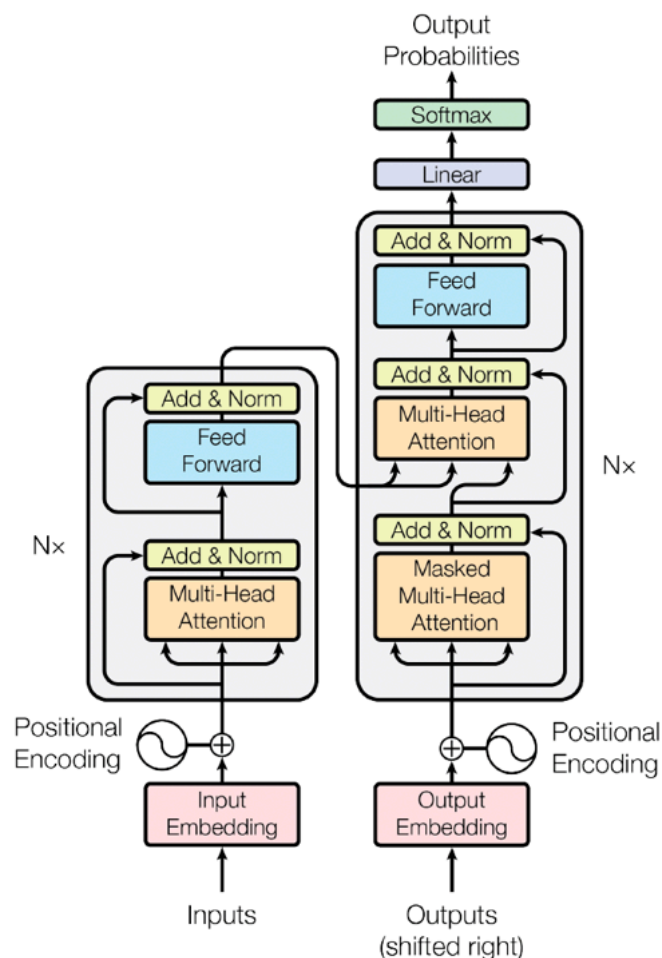


Figura 15. La figura muestra la estructura básica de un transformer (DeepLearning.AI, 2023b).

Los transformers tienen dos componentes principales:

- *Encoder Stack*: El encoder se encarga de analizar el contexto de la secuencia de entrada
- *Decoder Stack*: El decoder se encarga de generar la salida basándose en la información del Encoder.

El Encoder se encarga de codificar la secuencia original, mientras que el Decoder se encarga de decodificar para generar la salida deseada. En un Transformer, el texto fluye simultáneamente entre el Encoder y el Decoder, lo que permite capturar dependencias más largas y agilizar el entrenamiento. La estructura del Transformer permite trabajar con cualquier tipo de secuencias y ha revolucionado el procesamiento de lenguaje natural al eliminar la necesidad de redes recurrentes.

### NLP y sus controversias

El NLP ha sido objeto de varias controversias. Algunos se centran directamente en los modelos y sus resultados, otros en preocupaciones como quién tiene acceso a estos sistemas y cómo impacta en el mundo y la sociedad.

***Loros estocásticos***: un artículo de 2021 titulado “*Sobre los peligros de los loros estocásticos: ¿pueden los modelos lingüísticos ser demasiado grandes?*” (Emily B. et al., 2021) examina cómo los modelos lingüísticos pueden repetir y amplificar los sesgos encontrados en sus datos de entrenamiento. Los autores señalan que enormes conjuntos de datos no seleccionados extraídos de la web seguramente incluirán sesgos sociales y otra información indeseable, y los modelos entrenados en ellos absorberán estos defectos.

Promueven un mayor cuidado al seleccionar y documentar conjuntos de datos, evaluar el impacto potencial de un modelo antes de su desarrollo y fomentar la investigación en direcciones distintas al diseño de arquitecturas cada vez más grandes para ingerir conjuntos de datos cada vez más grandes.

***Coherencia versus sensibilidad***: La falacia de atribuir a la IA una inteligencia similar a la humana.

***Impacto ambiental***: Los modelos de lenguaje grandes requieren mucha energía tanto durante el entrenamiento como durante la inferencia. Un estudio estimó que entrenar un único modelo de lenguaje grande puede emitir cinco veces más dióxido de carbono que un solo automóvil durante su vida operativa. Otro estudio encontró que los modelos consumen incluso más energía durante la inferencia que el entrenamiento (Smith J. & Johnson L., 2020).

***El alto costo deja fuera a los investigadores no corporativos:*** Los requisitos computacionales necesarios para entrenar o implementar grandes modelos de lenguaje son demasiado costosos para muchas empresas pequeñas.

***Caja negra:*** Cuando un modelo de aprendizaje profundo genera un resultado, es difícil o imposible saber por qué generó ese resultado en particular. Se dice que estos sistemas “no son explicables”, ya que no podemos explicar cómo llegaron a su resultado.

***Nonsense on stilts:*** El escritor Gary Marcus (Deep Learning: A Critical Appraisal - Gary Marcus - New York University) ha criticado NLP basada en el aprendizaje profundo por generar un lenguaje sofisticado que induce a errores a los usuarios haciéndoles creer que los algoritmos del lenguaje natural entienden lo que dicen y suponen erróneamente que son capaces de realizar un razonamiento más sofisticado que el de un humano.

## Usos del procesamiento de lenguaje natural (NLP)

La NLP se utiliza para una amplia variedad de tareas relacionadas con el lenguaje y la interacción entre hombre-máquina.

**El análisis de sentimientos** es el proceso de clasificar la intención emocional del texto. Generalmente, la entrada a un modelo de clasificación de sentimientos es un fragmento de texto y el resultado es la probabilidad de que el sentimiento expresado sea positivo, negativo o neutral.

El análisis de sentimientos se utiliza para clasificar las opiniones de los usuarios o clientes, así como para aplicaciones específicas como la identificación de algún tipo de signo de trastorno mental.

**La clasificación de toxicidad** es una rama del análisis de sentimientos donde el objetivo no es sólo clasificar la intención hostil sino también clasificar categorías particulares como amenazas, insultos, obscenidades y odio hacia ciertas identidades. La entrada a dicho modelo es texto y la salida generalmente es la probabilidad de cada clase de toxicidad. Los modelos de clasificación de toxicidad se pueden utilizar para moderar y mejorar conversaciones en línea silenciando comentarios ofensivos, detectando discursos de odio o escaneando documentos en busca de difamación.

**La traducción automática** automatiza la traducción entre diferentes idiomas. La entrada a dicho modelo es texto en un idioma de origen específico y la salida es el texto en un idioma de destino específico.

Los enfoques de la traducción automática pueden distinguir entre palabras con significados similares. Algunos sistemas también realizan la identificación del idioma; es decir, clasificar

el texto según esté en un idioma u otro. Un ejemplo muy conocido y utilizado es la aplicación “Google Translate”<sup>6</sup>.

**El reconocimiento de entidades con nombre** tiene como objetivo extraer entidades en un fragmento de texto en categorías predefinidas, como nombres personales, organizaciones, ubicaciones y cantidades. La entrada a dicho modelo es generalmente texto, y la salida son las distintas entidades nombradas junto con sus posiciones inicial y final. El reconocimiento de entidades nombradas es útil en aplicaciones como resumir artículos de noticias y combatir la desinformación.

**La detección de spam** es un problema de clasificación binaria frecuente en NLP, donde el propósito es clasificar los correos electrónicos como spam o no. Los detectores de spam toman como entrada el texto del correo electrónico junto con otros subtítulos como el “Asunto” y el “Remitente”. Su objetivo es generar la probabilidad de que el correo sea spam.

**Los modelos de corrección de errores gramaticales** codifican reglas gramaticales para corregir la gramática dentro del texto. Sistemas de procesamiento de textos como “Microsoft Word” utilizan estos modelos para brindar una mejor experiencia de escritura a sus usuarios.

**La generación de texto**, más formalmente conocida como generación de lenguaje natural (NLG), produce texto similar al texto escrito por humanos. Estos modelos se pueden ajustar para producir texto en diferentes géneros y formatos. Es particularmente útil para autocompletar y chatbots.

*La función de autocompletar* predice qué palabra viene a continuación, y en aplicaciones de chat como WhatsApp<sup>7</sup> se utilizan sistemas de autocompletar de diversa complejidad. Google utiliza el autocompletado para predecir consultas de búsqueda. Uno de los modelos de autocompletar más famosos es GPT.

*Los chatbots* automatizan un lado de una conversación, mientras que un interlocutor humano generalmente aporta el otro lado.

**La recuperación de información** encuentra los documentos más relevantes para una consulta. Este es un problema que enfrenta todo sistema de búsqueda y recomendación. El objetivo no es responder a una consulta en particular, sino recuperar, de una colección de documentos que pueden estar numerados en millones, un conjunto que sea más relevante para la consulta.

**El resumen** es la tarea de sintetizar el texto o contenido para resaltar la información más relevante.

---

<sup>6</sup> <https://translate.google.com/>

<sup>7</sup> <https://web.whatsapp.com/>

*El resumen extractivo* se centra en extraer las oraciones más importantes de un texto extenso y combinarlas para formar un resumen.

*El resumen abstractivo* produce un resumen parafraseando. Esto es similar a escribir el resumen que incluye palabras y oraciones que no están presentes en el texto original.

**La respuesta a preguntas** consiste en responder preguntas planteadas por humanos en un lenguaje natural. Generalmente, las tareas de respuesta a preguntas vienen en dos tipos:

*Opción múltiple:* en la respuesta de preguntas de opción múltiple se compone de una pregunta y un conjunto de posibles respuestas. La tarea de aprendizaje es elegir la respuesta correcta.

*Dominio abierto:* en la respuesta a preguntas de dominio abierto, el modelo proporciona respuestas a preguntas en lenguaje natural sin proporcionar opciones, a menudo consultando una gran cantidad de textos.

## Estado del arte y contexto

El Procesamiento de Lenguaje Natural (NLP) es un campo de investigación que busca analizar, entender y manipular el lenguaje de textos escritos y hablados mediante técnicas computacionales.

El NLP como disciplina surgió a fines de la década de 1940. El primer artículo publicado sobre el tema lo publicó Alan Turing en 1950 y se tituló “Computing Machinery and Intelligence”, un trabajo que sostenía la importancia que computadoras inteligentes pudieran mantener conversaciones con personas sin que éstas notaran que están hablando con máquinas.

Hasta fines de la década de 1960 la representación del significado y el desarrollo de soluciones manejables computacionalmente no era factible con los marcos teóricos de las teorías de gramática de ese entonces. Esto comienza a cambiar en 1965, en relación a las teorías de gramática, con el modelo transformacional de competencia lingüística introducido por Chomsky (Chomsky N., 1965), considerado una insignia en el desarrollo del campo de NLP. Otros trabajos posteriores vinieron a subsanar los puntos débiles de esta teoría, relacionados con un enfoque muy centrado en lo sintáctico y dificultades para la implementación computacional del modelo.

Hacia fines de la década de 1970 ya se observa una mayor preocupación por temas semánticos y el fenómeno del discurso.

En la década de 1980 el avance en la capacidad de cómputo habilitó aproximaciones basadas en métodos estadísticos.

La introducción de técnicas de aprendizaje automático significó una revolución. Comenzó con árboles de decisión para luego pasar a modelos más confiables, integrados a sistemas más amplios. Desarrollos más recientes cambiaron el foco de modelos supervisados a no-supervisados o semi-supervisados.

En la década de 1990 continuó la tendencia hacia el uso de modelos empíricos a partir de textos de lenguaje natural y el advenimiento de corpus más grandes.

El aumento en la capacidad de cómputo y almacenamiento y en la disponibilidad de texto electrónico a través de internet significaron una explosión en el campo de NLP.

En los 2000 esta tendencia se consolida y se considera a estos métodos estadísticos muy eficientes y capaces de realizar tareas de análisis de lenguaje con un desempeño similar al de los humanos. Otro hito importante en esta década se dio con la llegada de los clasificadores bayesianos con aplicaciones en la clasificación de documentos que hizo posible, por ejemplo, analizar correos electrónicos y SMS y etiquetarlos como spam o mensaje no deseado. Además, se introdujeron los modelos n-gram cuya simpleza y escalabilidad llevaron a su gran adopción para el reconocimiento de textos y discursos (Palanivinayagam et al., 2023). Los enfoques estadísticos fueron exitosos en responder a problemas como la identificación de partes de discurso, desambiguación del sentido de las palabras, entre otros. La próxima generación de sistemas de NLP que se está desarrollando apuntan a la comprensión de discurso.

Otro quiebre en el campo de NLP ha sido la posibilidad de establecer relaciones semánticas mediante procesamiento de grandes cantidades de texto.

Otro avance notable se dio con la llegada del “macheo” aproximado de palabras (fuzzy string matching). Estos algoritmos determinan la distancia entre palabras a través de la cantidad de operaciones primitivas para lograr un macheo exacto. Estas técnicas se utilizan, por ejemplo, en los programas de corrección ortográfica (Hosseini K. et al., 2020).

Un gran paso adelante en NLP lo representan los dispositivos a los cuales un humano les habla. Una gran cantidad de investigaciones se están desarrollando en el campo de NLP para la mejora de sistemas humanizados, con el fin de hacerlos capaces de entender instrucciones simples.

Finalmente, antes de culminar estado del arte, es bueno mencionar un desarrollo reciente que está revolucionando los campos de NLP e inteligencia artificial en general: GPT-3 (Generative Pre-trained Transformers), un modelo de lenguaje autorregresivo que utiliza aprendizaje profundo (Deep Learning) para producir texto parecido al generado por humanos.

Fue desarrollado por OpenAI, una organización sin fin de lucro con la misión de “asegurar que la Inteligencia Artificial General (AGI) beneficie a la humanidad”, de acuerdo a lo señalado en su página web<sup>8</sup>. Actualmente evolucionado a GPT-4 aunque cuenta con otros modelos.<sup>9</sup>

## LLMs

### Modelos de lenguaje

El lenguaje humano es un componente esencial que no solo facilita la comunicación entre humanos, sino que también se ha convertido en un elemento fundamental en la interacción entre humanos y máquinas. Esta interacción ha evolucionado significativamente en el último tiempo, con un enfoque cada vez mayor en el desarrollo de tecnologías capaces de comprender y procesar el lenguaje natural de manera efectiva.

La creciente demanda de sistemas automatizados que manejen tareas lingüísticas complejas, como la traducción automática, el resumen de textos, la recuperación de información y las interacciones conversacionales, ha generado la necesidad de modelos de lenguaje más generalizados y avanzados.

En este contexto, los avances recientes en modelos lingüísticos han sido particularmente notables, impulsados principalmente por la introducción de arquitecturas basadas en “*Transformers*”. Estos modelos revolucionaron el campo del procesamiento del lenguaje natural (NLP), se benefician de capacidades computacionales cada vez mayores y de la disponibilidad de grandes conjuntos de datos de entrenamiento.

Gracias a esto, ahora tenemos modelos de lenguaje a gran escala (LLMs, por sus siglas en inglés “Large Language Model”) que pueden igualar e incluso superar el desempeño humano en muchas tareas relacionadas con el lenguaje. Estos modelos no solo tienen una habilidad increíble para entender y generar texto, sino que también están cambiando por completo la manera en que las máquinas pueden interactuar con el lenguaje humano, abriendo nuevas posibilidades en la inteligencia artificial.

---

<sup>8</sup> <https://openai.com/about/>

<sup>9</sup> <https://platform.openai.com/docs/models>

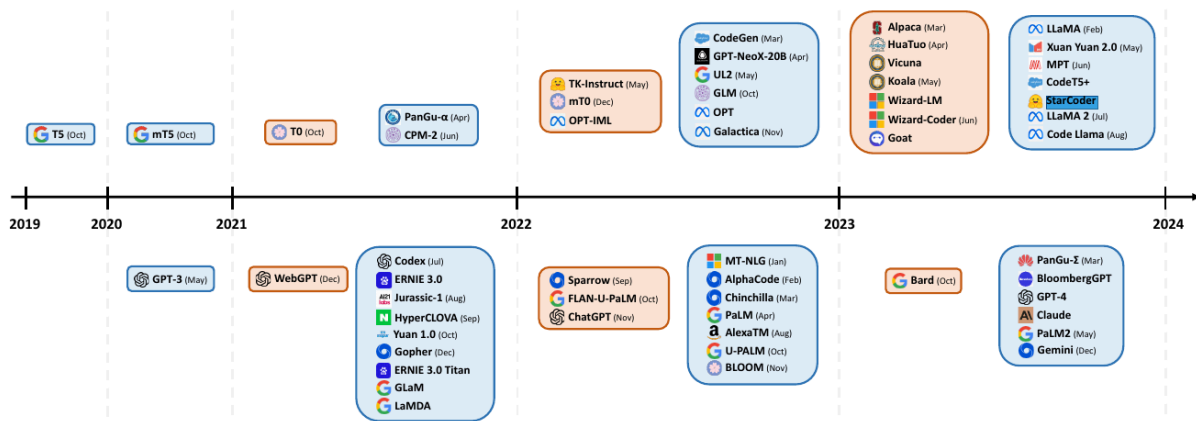


Figura 16. Evolución temporal y lanzamientos de modelos LLMs, tanto pre-entrenados como ajustados a instrucciones.

Como puede apreciarse, la figura 16 muestra una línea de tiempo con los lanzamientos de LLMs: las tarjetas azules representan modelos “pre-entrenados” y las naranjas son modelos 'ajustados a instrucciones'. Los modelos en la parte superior son de código abierto, mientras que los de abajo son de código cerrado. El gráfico destaca cómo cada vez más se están creando modelos pre-entrenados y cómo el código abierto está ganando terreno. (Fuente: A Comprehensive Overview of Large Language Models)

Los Modelos de Lenguaje de gran escala (LLMs) se han convertido en sistemas de inteligencia artificial capaces de procesar y generar texto de manera coherente, además de poder adaptarse a múltiples tareas y contextos. La evolución del procesamiento de lenguaje natural (NLP) ha pasado de modelos estadísticos a modelos neuronales, y luego de modelos de lenguaje pre-entrenados (PLMs) a LLMs. Los PLMs se entrenan con grandes cantidades de datos, buscando aprender una representación genérica aplicable a varias tareas de NLP.

Los LLMs manejan un alto número de parámetros (de decenas a cientos de miles de millones) y la cantidad de datos de entrenamiento (varios GBs y TBs). Estos parámetros son componentes ajustables del modelo, que se afinan durante el entrenamiento para mejorar su capacidad de entender y generar lenguaje humano de manera efectiva.

La cantidad masiva de parámetros entrenables en un LLM tiene profundas implicancias en su capacidad y rendimiento. A medida que aumenta el número de parámetros, el modelo puede capturar y memorizar una gama más amplia de patrones y relaciones en los datos de entrenamiento. Esto puede resultar en una mejora significativa en tareas complejas de NLP, como la traducción automática, la generación de texto y el análisis semántico.

Un número excesivo de parámetros también puede llevar a un aumento en los requisitos computacionales y de memoria, así como a problemas de sobreajuste si el modelo memoriza

datos irrelevantes o ruidosos. Por lo tanto, encontrar el equilibrio óptimo entre la capacidad del modelo y su generalización es crucial para el desarrollo de LLMs efectivos y eficientes.

En los enfoques más recientes, se han desarrollado modelos con capacidades que permiten el procesamiento de datos de entrada provenientes de diversas fuentes, incluyendo texto, audio, imágenes y fuentes externas como APIs.

Los LLMs son una alternativa más flexible y versátil en su aplicación en diferentes contextos.

## Capacidades y aplicaciones de los LLMs

Los Modelos de Lenguaje a Gran Escala (LLMs) han revolucionado la manera en que interactuamos con las máquinas y cómo éstas procesan y generan lenguaje natural. Estos modelos, entrenados con billones de parámetros y vastas cantidades de datos, han alcanzado capacidades que hace solo unos años eran impensables. A continuación, se exploran las principales capacidades y las aplicaciones más destacadas de los LLMs en la actualidad.

### Capacidades de los LLMs

1. **Comprensión del Lenguaje Natural:** Una de las capacidades más impresionantes de los LLMs es su habilidad para comprender el contexto y las sutilezas del lenguaje humano. Ya no se limitan a procesar palabras de manera aislada; ahora pueden captar el significado detrás de una frase completa, entender referencias contextuales y manejar la ambigüedad inherente al lenguaje. Esto les permite participar en conversaciones de manera coherente, responder preguntas con precisión y adaptarse a diferentes estilos y tonos de comunicación.
2. **Generación de Texto Coherente y Contextual:** Los LLMs no sólo comprenden el lenguaje, sino que también son expertos en generarlo. Pueden redactar textos largos y coherentes, mantener el hilo de una conversación o incluso crear contenido creativo como cuentos o ensayos. Esta capacidad ha abierto un mundo de posibilidades en la creación automática de contenido, desde la redacción de artículos hasta la generación de diálogos en videojuegos.
3. **Transferencia de Conocimiento entre Tareas:** Otra característica destacable de los LLMs es su capacidad para transferir conocimientos adquiridos de una tarea a otra. Por ejemplo, un modelo entrenado para completar frases también puede ser adaptado para traducir textos o resumir documentos sin necesidad de ser entrenado desde cero para cada tarea. Esta versatilidad ha hecho que los LLMs sean herramientas extremadamente útiles en múltiples dominios.
4. **Procesamiento Multimodal:** Los desarrollos más recientes han ampliado las capacidades de los LLMs para integrar y procesar datos provenientes de diferentes modalidades, como texto, imágenes y audio. Esto les permite, por ejemplo, describir el contenido de una imagen en palabras o generar subtítulos para videos de manera

automática, lo que facilita el desarrollo de aplicaciones que combinan varias formas de información.

### Aplicaciones de los LLMs

1. **Asistentes Virtuales y Chatbots:** Los LLMs han potenciado significativamente la funcionalidad de los asistentes virtuales y chatbots. Estos sistemas, como Siri, Alexa o Google Assistant, ahora pueden entender mejor las consultas de los usuarios, proporcionar respuestas más precisas y realizar tareas complejas como programar citas o controlar dispositivos inteligentes en el hogar. Además, la interacción con estos asistentes se ha vuelto más natural y fluida, lo que mejora la experiencia del usuario.
2. **Traducción Automática:** La traducción automática ha sido una de las aplicaciones más beneficiadas por los LLMs. Servicios como Google Translate han mejorado notablemente en precisión y fluidez gracias a la capacidad de estos modelos para captar el contexto de las frases y no traducir sólo palabra por palabra. Esto ha hecho que las traducciones sean más naturales y se adapten mejor al significado original del texto.
3. **Generación de Contenido:** En la industria de los medios y la publicidad, los LLMs se están utilizando para generar contenido automáticamente. Desde la creación de descripciones de productos hasta la redacción de informes financieros, los LLMs permiten producir grandes cantidades de texto con poco esfuerzo humano. También se están utilizando en el periodismo automatizado para generar noticias a partir de datos brutos.
4. **Análisis de Sentimientos y Opiniones:** Los LLMs se han convertido en herramientas poderosas para el análisis de sentimientos y opiniones en redes sociales, reseñas de productos, y encuestas. Estas aplicaciones permiten a las empresas entender mejor las percepciones y opiniones de sus clientes, lo que a su vez les ayuda a tomar decisiones más informadas sobre sus productos y servicios.
5. **Educación Personalizada:** En el ámbito educativo, los LLMs están siendo utilizados para desarrollar sistemas de tutoría inteligente que pueden adaptarse al ritmo de aprendizaje de cada estudiante. Estos sistemas pueden responder preguntas, proporcionar explicaciones detalladas y hasta generar ejercicios personalizados basados en el rendimiento del estudiante, lo que hace que el proceso de aprendizaje sea más efectivo y accesible.

### Conclusiones y perspectivas futuras

Los modelos de lenguaje a gran escala (LLMs) representan un avance sin precedentes en el procesamiento del lenguaje natural. Han demostrado ser herramientas poderosas y versátiles, capaces de realizar tareas complejas que antes requerían la intervención humana. Desde la generación de texto coherente hasta la adaptación a diversas aplicaciones, los LLMs han

transformado la interacción entre humanos y máquinas, facilitando la creación de sistemas de inteligencia artificial cada vez más sofisticados.

Sin embargo, a pesar de sus notables capacidades, el desarrollo y uso de LLMs no están exentos de desafíos. El incremento en la cantidad de parámetros y la complejidad de los modelos plantea exigencias significativas en términos de recursos computacionales, lo que puede limitar su accesibilidad y escalabilidad. Además, la tendencia hacia modelos cada vez más grandes también trae consigo riesgos de sobreajuste y la posibilidad de que los modelos reproduzcan sesgos presentes en los datos de entrenamiento.

Mirando hacia el futuro, es probable que la investigación en LLMs se enfoque en mejorar la eficiencia y la capacidad de generalización de estos modelos. Esto podría implicar el desarrollo de nuevas técnicas de entrenamiento, la integración de múltiples modalidades de datos, o incluso la creación de arquitecturas completamente nuevas que superen las limitaciones actuales de los transformers.

El éxito de los LLMs va a depender de la capacidad de la comunidad científica y tecnológica para abordar estos desafíos, equilibrando la potencia y versatilidad de los modelos con la necesidad de mantenerlos accesibles, eficientes y éticamente responsables. Con el avance continuo en esta disciplina, los LLMs tienen el potencial de ser no solo herramientas poderosas para el presente, sino también la base de futuras innovaciones en inteligencia artificial.

## Prompts en IA

### ¿Qué es un Prompt?

Un prompt en IA se refiere a una entrada de texto o instrucción inicial que se proporciona al modelo para guiarlo en la generación de respuestas o textos. Es esencialmente la semilla que el modelo utiliza para iniciar y dirigir su proceso de generación de lenguaje. Dependiendo del contexto y la tarea, un prompt puede ser una pregunta, una frase descriptiva o incluso un conjunto de ejemplos que delimitan la naturaleza y el tono del texto que se espera que el modelo genere.

### Propósito del Prompt

El propósito fundamental del prompt es influir en la salida del modelo de IA de manera controlada y predecible. Al proporcionar un prompt específico, los desarrolladores pueden orientar el modelo hacia la generación de respuestas coherentes y relevantes para una tarea particular. Esto es esencial en aplicaciones como chatbots, generación de texto creativo, traducción automática, resumen de texto, entre otras, donde la precisión y la relevancia del contenido generado son críticas.

El objetivo del prompt es simplificar la comunicación para usuarios que no están familiarizados con la programación o la inteligencia artificial. Esto permite que cualquier persona pueda aprovechar estas herramientas utilizando un lenguaje común. Esto, a su vez, permite la diversificación de uso de las IA, pudiendo ser empleadas en casi todas las áreas laborales. Siempre y cuando se le den las instrucciones adecuadas.

## Importancia de prompts precisos

La precisión en la formulación de un prompt es un aspecto fundamental. Un prompt bien diseñado es la diferencia entre una respuesta genérica y una altamente informativa y específica.

Esto se debe a que los modelos de IA, por avanzados que sean, aún dependen de las señales que reciben de los usuarios para generar respuestas relevantes. Un prompt claro, conciso y bien dirigido minimiza malentendidos y maximiza la utilidad de la respuesta generada.

Esto involucra la inclusión de detalles específicos, la limitación del ámbito de la pregunta, o la solicitud de un formato particular para la respuesta, como una lista o una explicación detallada.

A través de una comprensión sólida de cómo funcionan los prompts y la importancia de su precisión, puedes mejorar significativamente la calidad y relevancia de las interacciones con la IA. Ya sea para fines educativos, de investigación, o incluso creativos, la habilidad para diseñar prompts precisos y efectivos se está convirtiendo en una herramienta cada vez más valiosa en la materia.

## ¿Cómo obtener un buen Prompt?

No hay una fórmula exacta para crear prompts que aplique para todos los contenidos ni todas las IA. Pero sí hay elementos generales o básicos que deben formar parte de cualquier conjunto de instrucciones eficaz para comunicarse con una inteligencia artificial.

Obtener un prompt efectivo implica considerar varios aspectos clave que maximicen la capacidad del modelo para generar resultados deseables:

**Rol:** Para generar una respuesta más específica, es conveniente pedirle a la IA que adopte un rol. Esto significa que se le debe indicar cómo queremos que «piense» al responder. Estos detalles hacen que la inteligencia artificial asuma un rol y una identidad que le permite relacionar mejor los datos en su base de conocimientos y generar respuestas mejor adaptadas a nuestras necesidades.

**Contexto:** El contexto sirve para delimitar la respuesta hasta ciertos elementos más concretos del tema, evitando así que la IA incluya datos innecesarios. El contexto es parte de la instrucción que se le da a la IA, pues aporta los detalles que le permiten al modelo IA entender mejor la tarea que va a realizar.

**Instrucción:** El elemento más importante de todos, en un prompt, es la instrucción o tarea concreta que debe resolver la IA. Esta tiene que ser lo más explícita posible, evitando las ambigüedades y los términos abstractos, ya que estos dan pie a varias posibles respuestas. También, la instrucción debe ser concisa, ya que un exceso de palabras puede llevar a que el sistema relacione mal los términos y genere una respuesta imprecisa. Lo ideal es ser claro y breve en la instrucción.

**Utilizar palabras clave relevantes:** Incorporar palabras clave pertinentes en nuestro prompt ayuda a guiar al modelo hacia el tema o área específica que te interesa. Esto es particularmente útil para prompts relacionados con campos especializados

**Especificidad:** Mientras más específico sea el prompt, más precisa y útil es la respuesta. No escatimar u omitir detalles que puedan ayudar a la IA a entender mejor lo que se le está pidiendo. Los detalles son "pistas" que guían a la IA hacia lo que realmente se necesita, haciendo que la respuesta sea más acertada y ajustada al contexto.

**Experimentar con diferentes formulaciones:** No hay una única manera correcta de formular un prompt. Si la primera respuesta no cumple con nuestras expectativas, podemos reformular el prompt con diferentes palabras o estructuras.

**Limitar el alcance:** A veces, menos es más. Limitar el alcance de la pregunta puede ayudar a obtener respuestas más concentradas y profundas. Debemos intentar enfocar nuestro prompt en un aspecto específico de un tema más amplio para evitar respuestas demasiado generales.

## Relevancia del Prompt en la IA

El prompt desempeña un papel crítico en la configuración y el rendimiento de los modelos de IA, especialmente en modelos de lenguaje NLP. La elección cuidadosa del prompt puede influir en aspectos como la coherencia del texto generado, la relevancia del contenido y la capacidad del modelo para adaptarse a diferentes estilos y tonos de conversación. Además, el prompt también puede ser crucial para guiar al modelo en la resolución de tareas específicas como la generación de código, la respuesta a preguntas, la redacción de informes y más.

## Consideraciones Adicionales

Además de los aspectos técnicos del prompt, también es importante considerar:

**Ética y Seguridad:** La formulación del prompt debe tener en cuenta consideraciones éticas y de seguridad, asegurando que el modelo no genere contenido inapropiado o sesgado.

**Flexibilidad y Adaptabilidad:** Los sistemas de IA deben poder manejar una variedad de prompts y ajustarse dinámicamente a diferentes requisitos y contextos.

**Innovación Continua:** A medida que avanza la investigación en IA, la exploración de nuevas formas de utilizar y mejorar el prompt sigue siendo un área de innovación constante.

## Importancia de “prompt engineering”

*Prompt Engineering* es el “arte” de crear instrucciones efectivas que guíen a la IA para generar las respuestas deseadas. Desempeña un papel crucial a la hora de maximizar la eficacia del modelo de lenguaje al cerrar o achicar la brecha entre la intención del usuario y la comprensión del modelo. Una instrucción bien diseñada puede mejorar significativamente la calidad y relevancia del resultado, mientras que uno mal diseñado puede generar respuestas insatisfactorias o erróneas.

Por ejemplo:

**Pregunta 1:** "Háblame de la IA".

**Respuesta 1:** “La inteligencia artificial (IA) es un campo de la informática que se centra en la creación de sistemas capaces de realizar tareas que, normalmente, requieren inteligencia humana. Estas tareas incluyen el reconocimiento de voz, la toma de decisiones, la traducción de idiomas, la visión por computadora, y más. La IA se basa en algoritmos y modelos matemáticos que permiten a las máquinas aprender de los datos, identificar patrones, y tomar decisiones o realizar predicciones basadas en esa información”.

**Pregunta 2:** "Proporcione un resumen conciso de la inteligencia artificial, sus subcampos y aplicaciones actuales"

**Respuesta 2:** “La inteligencia artificial (IA) es el estudio de la creación de agentes inteligentes capaces de realizar tareas de forma autónoma. Los subcampos clave incluyen aprendizaje automático, procesamiento del lenguaje natural y robótica. Las aplicaciones actuales van desde asistentes virtuales y sistemas de recomendación hasta vehículos autónomos y diagnóstico médico”.

El segundo mensaje produce una respuesta más informativa y enfocada debido a su claridad y especificidad de la pregunta realizada.

## Fundamentos de Prompt Engineering

Prompt Engineering es el proceso de diseñar, refinar y optimizar las instrucciones (prompts) de entrada para comunicar de manera efectiva la intención del usuario a un modelo de lenguaje. Esta práctica es esencial para obtener respuestas precisas, relevantes y coherentes del modelo. La calidad de las instrucciones afecta directamente a la calidad de las respuestas generadas, y entender estos factores es vital para la interacción efectiva y significativa con los modelos de lenguaje.

### Factores que influyen en la elección de instrucciones (prompts)

La elección de una interacción eficiente depende de varios factores, entre ellos:

**Intención del usuario:** Entender el objetivo del usuario y el resultado deseado. Esto ayuda a elaborar un prompt que se alinee con las expectativas del usuario. Es importante considerar el propósito de la interacción, ya sea para recuperar información, generar contenido o resolver un problema.

**Comprensión del modelo:** Familiarizarse con las fortalezas y limitaciones del modelo de lenguaje. Este conocimiento ayuda a diseñar indicaciones que exploten las capacidades del modelo y al mismo tiempo mitiguen sus debilidades. Se debe tener en cuenta que incluso los modelos de lenguaje más modernos pueden tener dificultades con determinadas tareas o producir información incorrecta.

**Especificidad del dominio:** Cuando se trata de un dominio especializado, se debe utilizar vocabulario o contexto específico del dominio para guiar al modelo hacia la respuesta deseada. Proporcionar contexto o ejemplos adicionales puede ayudar al modelo a generar resultados más precisos y relevantes.

**Claridad y especificidad:** La instrucción debe ser clara y específica para evitar ambigüedades o confusión, que pueden dar lugar a respuestas indeseadas. La ambigüedad puede surgir de instrucciones poco claras, preguntas vagas o contexto insuficiente.

**Restricciones:** Limitar el alcance de una pregunta puede ayudar a obtener la respuesta deseada. También tener en cuenta si es necesaria alguna restricción (por ejemplo, longitud o formato de la respuesta) para lograr el resultado deseado. Especificar restricciones explícitamente puede ayudar a guiar el modelo hacia la generación de respuestas que cumplan requisitos específicos, como límites de caracteres o formatos estructurados.

Al considerar estos factores, se pueden crear instrucciones que comuniquen efectivamente su intención al modelo de lenguaje y obtener respuestas de alta calidad. Además, la ingeniería de instrucciones es un proceso iterativo. Experimentar con diferentes indicaciones, probar su

efectividad y perfeccionar su enfoque en función del rendimiento del modelo puede generar mejoras significativas en la calidad del resultado generado.

### Buenas prácticas

En el área de la inteligencia artificial, el prompt engineering o ingeniería de instrucciones se ha convertido en una disciplina esencial para optimizar la interacción entre humanos y modelos de lenguaje. Esta práctica consiste en diseñar y refinar las instrucciones dadas a una IA para maximizar la precisión y relevancia de las respuestas generadas. A medida que los modelos de lenguaje avanzan en complejidad y capacidad, la habilidad para formular prompts efectivos se vuelve crucial para obtener resultados útiles y alineados con las necesidades del usuario. Las buenas prácticas en prompt engineering no sólo facilitan una comunicación más clara con el modelo de lenguaje, sino que también mejoran la eficiencia y la efectividad en el desarrollo de aplicaciones basadas en inteligencia artificial. En este contexto, explorar y aplicar estrategias efectivas para el diseño de instrucciones es fundamental para aprovechar al máximo las capacidades de los modelos de lenguaje y garantizar una buena experiencia de usuario.

### *Pruebas iterativas y refinamiento*

Una de las formas más efectivas de mejorar los prompts es mediante un proceso iterativo de prueba y refinamiento. Refinar continuamente las instrucciones en función de las respuestas generadas ayuda a identificar áreas que requieren mejora y afinar el comportamiento del modelo.

Ejemplo:

**Instrucción inicial:** "Háblame de prompts en IA".

**Respuesta inicial:** "Los "prompts" son como las preguntas o instrucciones que le das a una IA para obtener una respuesta o realizar una tarea específica. En el contexto de los modelos de lenguaje como ChatGPT, Gemini o Llama, un prompt es el texto que introduces para guiar la generación de respuestas."

**Instrucción refinada:** "Describe que es un prompt en IA, sus características y factores a tener en cuenta"

**Respuesta Refinada:** "Un **prompt** en Inteligencia Artificial (IA) es una entrada o instrucción que se le da a un modelo de IA para obtener una respuesta o acción deseada. En términos simples, es como una pregunta o comando que guía al modelo para que genere la información o realice una tarea específica [continúa la respuesta...]"

Características de un prompt [continúa la respuesta ...]

Factores a tener en cuenta en un prompt [continúa la respuesta ...]"

### ***Equilibrar la intención del usuario y la creatividad del modelo***

Si bien los modelos de lenguaje son capaces de generar respuestas creativas, es crucial equilibrar la intención del usuario con la creatividad del modelo. Se debe intentar que la instrucción cubra las necesidades del usuario y al mismo tiempo dejar espacio para que el modelo muestre sus capacidades.

Ejemplo:

**Instrucción:** "Escribe una historia de ciencia ficción sobre una Inteligencia Artificial maligna"

**Respuesta:** "En un futuro no tan lejano, la humanidad había desarrollado una inteligencia artificial llamada *Helios*..."

Una historia creativa y atractiva que satisface la solicitud del usuario y al mismo tiempo demuestra las habilidades narrativas del modelo de lenguaje.

### ***Aprovechamiento de recursos externos y API***

En algunos casos, el modelo de lenguaje puede no tener suficiente conocimiento o precisión para responder a las instrucciones de los usuarios de manera efectiva. Para abordar esta limitación, se puede utilizar recursos externos y APIs, lo que permite al modelo acceder a información en tiempo real o específica del dominio. La integración de APIs puede mejorar significativamente la calidad y relevancia de las respuestas del modelo de lenguaje.

Ejemplo:

En un caso práctico del trabajo de tesina, utilizar una API de Git<sup>10</sup> para obtener un catálogo o historial de *commits* en un desarrollo de software y sus características (hash, diferencias, autor, etc), y mediante el modelo de lenguaje poder ubicar el commit que se necesita mediante una instrucción en lenguaje natural.

Otro caso, mediante la API de Git obtener las modificaciones realizadas en el desarrollo de software y mediante el modelo de lenguaje utilizar esa información para generar posibles mensajes de commit.

### ***Garantizar un uso ético y evitar sesgos***

Los modelos de lenguaje pueden generar sin darse cuenta contenido sesgado o inapropiado. Para garantizar un uso ético, es esencial establecer directivas y restricciones que ayuden a mitigar estos problemas.

---

<sup>10</sup> <https://docs.github.com/en/rest?apiVersion=2022-11-28>

- Ser consciente de los posibles sesgos: familiarizarse con los posibles sesgos que pueden surgir en las respuestas del modelo de lenguaje.
- Buen uso del lenguaje: al diseñar instrucciones, utilizar un lenguaje que fomente perspectivas diversas.
- Evaluación del contenido generado: evaluar periódicamente el contenido generado por el modelo de lenguaje para detectar posibles sesgos o preocupaciones éticas.
- Implementación de filtros de contenido: utilizar filtros de contenido o herramientas de moderación para detectar las respuestas generadas en busca de contenido potencialmente dañino o sesgado antes de presentarlas a los usuarios.

# Mercado Actual

En el mercado actual de los Modelos de Lenguaje de Gran Escala (LLMs) está en constante evolución y diversificación impulsada por las tecnologías y el aumento en la demanda de aplicaciones basadas en inteligencia artificial. Estos modelos, conocidos por su capacidad para procesar y generar texto con alta coherencia y precisión, están transformando diversas industrias, desde la tecnología hasta la educación y la salud.

A continuación, describiremos algunos de los modelos más relevantes a la fecha de redacción de esta tesina.

## GPT-4 - OpenAI

### **Desarrollo y Capacidades:**

GPT-4 es uno de los modelos más avanzados desarrollados por OpenAI, una empresa pionera en el campo de la inteligencia artificial. Lanzado en marzo de 2023, GPT-4<sup>11</sup> representa la cuarta iteración de la serie Generative Pre-trained Transformer, y está diseñado para realizar tareas de procesamiento de lenguaje natural con una precisión y fluidez sin precedentes. Este modelo ha sido entrenado en una vasta cantidad de texto y es capaz de generar texto coherente, traducir idiomas, resumir documentos largos y realizar análisis semánticos complejos.

### **Integración y Usuarios:**

Uno de los factores clave que ha impulsado la adopción masiva de GPT-4 es su integración con productos de Microsoft. Gracias a la asociación estratégica entre OpenAI y Microsoft, GPT-4 se encuentra integrado en Azure OpenAI Service<sup>12</sup>, lo que permite a empresas de todo el mundo utilizar este modelo en sus propias aplicaciones. Además, Microsoft ha incorporado GPT-4 en Copilot, una herramienta diseñada para mejorar la productividad en Microsoft 365 mediante la generación de texto, la automatización de tareas y la asistencia en la escritura de código.

Con millones de usuarios en todo el mundo, GPT-4 se ha convertido en una herramienta esencial para desarrolladores, investigadores y empresas que buscan aprovechar las capacidades avanzadas de la IA.

### **Costos:**

El acceso a GPT-4 a través de Azure se basa en un modelo de precios que varía según el uso. El costo depende del número de tokens procesados, la complejidad de las tareas y la frecuencia de las solicitudes. Además, al momento de redacción de este documento, OpenAI ofrece una suscripción a ChatGPT Plus, que por USD 20 al mes permite a los usuarios

---

<sup>11</sup> <https://openai.com/index/gpt-4/>

<sup>12</sup> <https://azure.microsoft.com/es-es/products/ai-services/openai-service>

individuales acceder a GPT-4 dentro de la plataforma ChatGPT, proporcionando una opción más accesible para aquellos que no necesitan una implementación a gran escala.

## PaLM 2 y Gemini - Google DeepMind

### **Desarrollo y Capacidades:**

Google, a través de su división DeepMind, ha sido un jugador clave en la investigación y desarrollo de LLMs. PaLM 2 (Pathways Language Model) (Anil et al., 2023) es un modelo que sigue la evolución de los modelos anteriores de Google, diseñado para ser altamente eficiente y adaptable. Ha sido optimizado para tareas como la traducción, la comprensión de contexto y la generación de texto en múltiples idiomas. Además, Google ha introducido Gemini 1.5 (Gemini Team et al., 2024), una iteración aún más avanzada que incorpora capacidades de aprendizaje multimodal, permitiendo la integración de datos de texto, imágenes y otros formatos.

### **Integración y Usuarios:**

Google ha integrado PaLM 2 y Gemini<sup>13</sup> (Gemini Team et al., 2023) en una amplia gama de productos y servicios, incluyendo Google Search, Google Assistant y Google Workspace. Esto ha permitido a Google llegar a una base de usuarios masiva, que utiliza estas tecnologías en su vida diaria y en el entorno empresarial. Herramientas como Google Bard, basadas en PaLM 2, han sido diseñadas para ofrecer respuestas más precisas y relevantes en motores de búsqueda, lo que refuerza el papel de Google como líder en la provisión de información.

### **Costos:**

El acceso a PaLM 2 y Gemini a través de Google Cloud sigue un modelo de precios basado en el uso, similar al de OpenAI. Los costos se calculan en función del volumen de datos procesados, las solicitudes a la API y las características específicas del modelo utilizado. Esto permite a las empresas ajustar su inversión en función de sus necesidades, mientras que los usuarios individuales pueden acceder a versiones más limitadas de forma gratuita en productos como Google Search.

## LLaMA 3.1 - Meta

### **Desarrollo y Capacidades:**

Meta, anteriormente conocida como Facebook, ha adoptado un enfoque distintivo en el desarrollo de LLMs con su modelo LLaMA 3.1<sup>14</sup>, lanzado en julio de 2024. Este modelo representa una evolución respecto a sus predecesores, integrando mejoras significativas en términos de eficiencia y precisión. LLaMA 3.1 ha sido diseñado para ser accesible tanto para la comunidad investigadora como para los desarrolladores que buscan personalizar y ajustar los modelos a sus necesidades específicas. Al igual que las versiones anteriores, LLaMA 3.1

---

<sup>13</sup> <https://ai.google.dev/>

<sup>14</sup> <https://ai.meta.com/blog/meta-llama-3-1/>

se ofrece en diferentes escalas, capacidades y configuraciones, lo que permite un equilibrio entre la capacidad de procesamiento y los requisitos de infraestructura, asegurando que pueda ser utilizado en una amplia gama de aplicaciones, desde proyectos académicos hasta desarrollos comerciales.

### **Accesibilidad y Usuarios:**

A diferencia de otros gigantes tecnológicos, Meta ha optado por un enfoque más abierto, publicando LLaMA 3.1 bajo una licencia abierta<sup>15</sup> que permite su uso tanto comercial como no comercial. Esto ha facilitado que investigadores y empresas adopten el modelo para una variedad de aplicaciones, desde el desarrollo de asistentes virtuales hasta la automatización de procesos en diversas industrias. La estrategia de Meta ha sido bien recibida por la comunidad de IA, que valora la transparencia y la capacidad de experimentar con el modelo sin restricciones significativas.

### **Costos:**

Si bien LLaMA 3 es gratuito en términos de licenciamiento, los costos asociados provienen principalmente de la infraestructura necesaria para entrenar, ajustar y/o desplegar el modelo. Esto incluye el gasto en hardware, como GPUs de alta gama, y el costo de electricidad y mantenimiento de servidores. Para organizaciones con recursos limitados, el acceso a la nube puede ofrecer una alternativa rentable para experimentar con LLaMA 3.1 sin necesidad de invertir en infraestructura propia.

## Claude 3.5 - Anthropic

### **Desarrollo y Capacidades:**

Anthropic es una startup fundada por ex empleados de OpenAI, que ha centrado su trabajo en la creación de LLMs alineados éticamente con los valores humanos. Claude es la serie de modelos desarrollados por Anthropic, con Claude 3.5<sup>16</sup> lanzado en junio de 2024. Claude se diferencia de otros modelos en su enfoque en la seguridad y la ética, buscando minimizar los sesgos y evitar respuestas potencialmente dañinas o inexactas. Claude ha sido entrenado para ser más cauteloso en sus respuestas, lo que lo hace especialmente valioso en aplicaciones donde la seguridad y la ética son prioritarias.

### **Usuarios y Enfoque:**

Aunque Anthropic no cuenta con la misma base de usuarios que gigantes como OpenAI o Google, ha atraído a una audiencia específica interesada en la IA ética. Claude es utilizado principalmente por empresas y organizaciones que requieren modelos de IA que prioricen la seguridad, como en aplicaciones médicas, legales o educativas. Anthropic también ha ganado el interés de la comunidad investigadora que busca desarrollar IA de manera responsable y alineada con valores humanísticos.

---

<sup>15</sup> Meta's custom Open Model License Agreement (<https://www.llama.com/llama3/license/>)

<sup>16</sup> <https://www.anthropic.com/news/claude-3-5-sonnet>

**Costos:**

Los costos asociados con Claude no son tan ampliamente divulgados, pero se espera que Anthropic adopte un modelo de precios similar al de otros proveedores de LLMs, basado en el uso. Dado el enfoque en la seguridad, es posible que Claude tenga precios premium para garantizar que las aplicaciones cumplan con altos estándares éticos y de calidad.

## StarCoder

**Desarrollo y Capacidades:**

StarCoder (Li et al., 2023) es un modelo de lenguaje desarrollado específicamente para tareas de programación y desarrollo de software. Su entrenamiento se basa en grandes volúmenes de código fuente de múltiples lenguajes de programación, lo que lo hace especialmente apto para la generación de código, análisis de errores y recomendaciones de optimización. StarCoder es parte de una serie de modelos que buscan aumentar la eficiencia de los desarrolladores, proporcionando sugerencias inteligentes y soluciones a problemas complejos en tiempo real. A diferencia de otros modelos más generalistas, StarCoder ha sido ajustado para comprender mejor la sintaxis y la lógica de programación, lo que lo hace una herramienta valiosa en el desarrollo de software.

**Usuarios y Enfoque:**

StarCoder está diseñado para ser utilizado por desarrolladores de software, ingenieros y profesionales de TI que necesitan una herramienta que les ayude a mejorar la calidad y la eficiencia de su código. También es popular entre los educadores y estudiantes que buscan aprender a programar o mejorar sus habilidades en diferentes lenguajes de programación. Su enfoque en la programación lo diferencia de otros LLMs más generalistas, y ha ganado popularidad en comunidades de desarrolladores que valoran la precisión y relevancia de sus sugerencias de código.

**Costos:**

Los costos asociados con StarCoder suelen depender del volumen de uso y la complejidad de las tareas que se ejecutan. Similar a otros modelos especializados, se espera que StarCoder adopte un modelo de precios basado en el uso, con posibles tarifas premium para usuarios que requieran soporte avanzado o funcionalidades específicas. Dado su enfoque en la programación, puede ser una inversión rentable para empresas de desarrollo de software que buscan aumentar la productividad y reducir errores en el código.

## MPT (MosaicML)

**Desarrollo y Capacidades:**

MPT, desarrollado por MosaicML (Databricks, 2023), es una familia de modelos de lenguaje que se enfoca en la eficiencia y escalabilidad. MPT (Mosaic Pretrained Transformer) es

conocido por ser altamente optimizable y adaptable, permitiendo a las empresas entrenar modelos personalizados que se alineen con sus necesidades específicas. Una de las características destacadas de MPT es su arquitectura modular, que permite a los desarrolladores ajustar y optimizar el modelo para diferentes tareas, desde el procesamiento del lenguaje natural hasta la síntesis de información compleja.

### **Usuarios y Enfoque:**

MPT es especialmente atractivo para empresas y organizaciones que buscan crear modelos de lenguaje altamente personalizados y optimizados. Debido a su flexibilidad, MPT se ha adoptado en una variedad de sectores, desde la investigación científica hasta la creación de contenido automatizado. Las startups y empresas que requieren una solución de IA adaptable y que valoran la eficiencia y la capacidad de escalar, encuentran en MPT una herramienta poderosa. Además, su diseño modular ha atraído a desarrolladores interesados en experimentar y mejorar el rendimiento de los modelos de lenguaje.

### **Costos:**

Los costos de utilizar MPT dependen en gran medida de los recursos necesarios para entrenar y desplegar el modelo. MosaicML ofrece opciones que van desde modelos preentrenados hasta soluciones personalizadas, lo que implica una estructura de precios que puede variar considerablemente según las necesidades del usuario. Las empresas que buscan maximizar el rendimiento y optimizar costos a través de un enfoque a medida pueden encontrar en MPT una solución rentable, aunque la inversión inicial en infraestructura y personalización podría ser significativa.

## Modelos Open Source

Además de las ofertas comerciales, el mercado de LLMs también incluye modelos de código abierto que han ganado tracción en la comunidad de IA.

### LLaMA 3.1

Meta ha liderado esta tendencia con LLaMA 3.1, proporcionando un modelo accesible para el público en general bajo una licencia abierta. Esta apertura ha permitido a los desarrolladores experimentar y adaptar el modelo a una variedad de aplicaciones, promoviendo la innovación en el campo.

### Mistral

Lanzado en septiembre de 2023 por Mistral AI (Jiang et al., 2023), Mistral es un modelo de 7 mil millones de parámetros diseñado para ser altamente eficiente y adaptable. Mistral se ha destacado por su enfoque en la accesibilidad y la optimización, lo que lo hace ideal para aplicaciones que requieren un rendimiento elevado con recursos limitados.

## Falcon 2

El Instituto de Innovación Tecnológica de Abu Dhabi ha contribuido al ecosistema de código abierto con el lanzamiento de Falcon (Malartic et al., 2024), un LLM que ha sido bien recibido por su rendimiento robusto y su accesibilidad. Falcon ha sido diseñado para aplicaciones que requieren una alta capacidad de procesamiento de lenguaje con un enfoque en la eficiencia y la escalabilidad.

## StarCoder

Es un modelo de código abierto desarrollado por Hugging Face<sup>17</sup> en colaboración con ServiceNow<sup>18</sup> (ServiceNow Research, 2023). Fue lanzado como parte de la iniciativa BigCode<sup>19</sup> (ServiceNow Research, 2022), con el objetivo de democratizar el acceso a herramientas de IA para programación. Esto significa que cualquier desarrollador o investigador puede acceder al código y utilizar el modelo para sus propios proyectos, sin costo alguno.

## MPT (MosaicML Pretrained Transformer)

También es un modelo de código abierto. MosaicML lanzó MPT como parte de su misión de hacer que la inteligencia artificial avanzada sea accesible para una amplia audiencia. Los modelos MPT se pueden descargar y utilizar libremente, permitiendo a las organizaciones adaptar y entrenar estos modelos según sus necesidades específicas sin incurrir en costos asociados a licencias o derechos de uso.

## Herramientas

La Inteligencia Artificial ha transformado múltiples industrias, y en el área del desarrollo de software no es la excepción. Dentro de este campo, el Análisis del Lenguaje Natural se ha convertido en una de las áreas más innovadoras, proporcionando herramientas como soporte al proceso de desarrollo de software mejorando la eficiencia y creatividad.

## GitHub Copilot

GitHub Copilot<sup>20</sup>(Friedman, 2021) es un asistente de codificación desarrollado por GitHub en colaboración con OpenAI. Está diseñado para ayudar a los desarrolladores sugiriendo código en tiempo real a medida que escriben. Lo que hace que Copilot sea "distinto" es su capacidad para comprender el contexto y las intenciones del desarrollador, lo que le permite ofrecer sugerencias de código que son no sólo relevantes, sino también inteligentes y adaptativas.

Copilot se basa en OpenAI Codex (OpenAI, 2021), una versión especializada de GPT-3 enfocada en la generación de código. Esta tecnología permite a Copilot aprender de grandes

---

<sup>17</sup> <https://huggingface.co/>

<sup>18</sup> <https://www.servicenow.com/>

<sup>19</sup> <https://www.bigcode-project.org/>

<sup>20</sup> <https://github.com/features/copilot>

cantidades de código fuente disponible públicamente y de las propias interacciones del usuario. Como resultado, puede sugerir desde simples líneas de código hasta funciones completas, e incluso fragmentos más complejos que impliquen lógica avanzada.

Además, Copilot no solo completa el código que el desarrollador está escribiendo, sino que también puede anticipar lo que podría necesitar a continuación, como las importaciones necesarias o las llamadas a funciones específicas. Esto, además de acelerar la codificación, ayuda a evitar errores comunes al sugerir mejores prácticas y patrones de diseño eficientes. Sin embargo, es importante que los desarrolladores revisen las sugerencias, ya que Copilot puede generar código que no siempre sea óptimo o seguro.

## Tabnine

Tabnine<sup>21</sup> es otro asistente de programación impulsado por inteligencia artificial, diseñado para autocompletar código de manera predictiva. A diferencia de GitHub Copilot, que se basa en OpenAI Codex, Tabnine utiliza sus propios modelos de inteligencia artificial y aprendizaje automático, optimizados para predecir lo que los desarrolladores escribirán a continuación.

Tabnine funciona como una extensión para entornos de desarrollo integrado (IDE) populares, como Visual Studio Code, IntelliJ IDEA, y otros. Una de las características más destacadas de Tabnine es su capacidad para personalizarse según el código y los patrones de uso específicos de cada equipo o desarrollador. Esto significa que a medida que más se utiliza, Tabnine aprende del estilo de codificación del usuario, mejorando la precisión de sus sugerencias con el tiempo.

Además de autocompletar código, Tabnine puede sugerir snippets (fragmentos) de código completos y corregir errores en tiempo real. Esto no sólo ayuda a mejorar la velocidad de desarrollo, sino que también reduce la carga cognitiva del desarrollador, permitiéndole concentrarse en tareas más complejas y creativas. Tabnine también ofrece una versión de pago que incluye características avanzadas como modelos entrenados en código privado y la posibilidad de funcionar sin conexión a internet, protegiendo así la privacidad del código del usuario.

## Kite

Kite fue una de las primeras herramientas en utilizar inteligencia artificial para mejorar la experiencia de codificación mediante el autocompletado de código en tiempo real. Aunque el desarrollo de Kite ha sido discontinuado, sigue siendo un referente en la historia de las herramientas de asistencia para la programación (Smith, 2022).

---

<sup>21</sup> <https://www.tabnine.com/>

Kite funcionaba como un complemento para IDEs populares y estaba especialmente optimizado para lenguajes como Python, aunque también ofrecía soporte para otros lenguajes como JavaScript, Go, y Java. Su modelo de IA estaba entrenado para anticipar lo que el desarrollador podría escribir a continuación, proporcionando sugerencias que aceleraban el proceso de codificación.

Una característica distintiva de Kite era su capacidad para proporcionar documentación en línea sobre las funciones y métodos sugeridos, lo que ayudaba a los desarrolladores a comprender mejor las recomendaciones y a aprender mientras codificaban. Aunque ya no se actualiza, Kite demostró el potencial de la IA para transformar el desarrollo de software, influyendo en la creación de herramientas más avanzadas como GitHub Copilot y Tabnine.

## CodeT5

CodeT5<sup>22</sup> es un modelo de procesamiento de lenguaje natural (NLP) diseñado específicamente para tareas de programación. Desarrollado como una extensión del modelo T5 (Text-To-Text Transfer Transformer), CodeT5 está optimizado para comprender y generar código en lugar de texto común.

A diferencia de herramientas como GitHub Copilot y Tabnine, que se centran en el autocompletado y la asistencia en tiempo real, CodeT5 se enfoca en tareas más especializadas, como la traducción de código entre diferentes lenguajes de programación, la generación automática de comentarios y documentación, y la refactorización de código.

Una de las aplicaciones clave de CodeT5 es su capacidad para automatizar tareas tediosas que, de otra manera, requerirían un esfuerzo considerable por parte del desarrollador. Por ejemplo, puede convertir código escrito en un lenguaje de programación a otro, manteniendo la lógica y funcionalidad, lo que es particularmente útil para proyectos que necesitan ser portados a diferentes plataformas. Además, CodeT5 puede generar documentación basada en el código existente, lo que facilita la comprensión y mantenimiento de proyectos a largo plazo.

## Amazon CodeWhisperer

Amazon CodeWhisperer<sup>23</sup> es una herramienta de desarrollo asistida por inteligencia artificial lanzada por Amazon Web Services (AWS). Al igual que GitHub Copilot, CodeWhisperer está diseñado para ayudar a los desarrolladores a escribir código más rápido y con mayor precisión, sugiriendo fragmentos de código, completado de líneas, y bloques de código completos en tiempo real.

---

<sup>22</sup> <https://serp.ai/codet5/>

<sup>23</sup> [https://docs.aws.amazon.com/es\\_es/codewhisperer/latest/userguide/what-is-cwspr.html](https://docs.aws.amazon.com/es_es/codewhisperer/latest/userguide/what-is-cwspr.html)

Lo que distingue a CodeWhisperer es su integración profunda con el ecosistema de AWS. Esto significa que las sugerencias de código no sólo están optimizadas para el lenguaje de programación que se esté utilizando, sino que también están alineadas con las mejores prácticas de AWS. Por ejemplo, CodeWhisperer puede sugerir configuraciones de servicios en la nube, manejar la autenticación y autorización con IAM (Identity and Access Management), o incluso optimizar la gestión de recursos en Amazon S3, Lambda, o DynamoDB.

Además, CodeWhisperer está diseñado para funcionar con los entornos de desarrollo integrado (IDEs) más populares, como Visual Studio Code, JetBrains, y otros. Esto facilita su adopción por parte de desarrolladores que ya están familiarizados con estos entornos. Otro aspecto destacado de CodeWhisperer es su enfoque en la seguridad del código. Puede identificar y advertir sobre posibles vulnerabilidades en el código generado, como la exposición de datos sensibles o configuraciones inseguras, ayudando a los desarrolladores a mantener un alto estándar de seguridad en sus proyectos.

Amazon CodeWhisperer no sólo mejora la velocidad de desarrollo, sino que también ofrece un soporte integral para la creación de aplicaciones seguras y optimizadas en la nube de AWS.

# Desarrollo

El presente trabajo de tesina describe el desarrollo de una extensión para el entorno Visual Studio Code (VSCoDe)<sup>24</sup>, que proporciona herramientas de asistencia en el desarrollo de software. Esta extensión se enfoca en dos funciones principales: un asistente para la creación de commits en Git y un catálogo de historial de commits. La herramienta está diseñada para facilitar el manejo de versiones y mejorar la experiencia del desarrollador en la gestión de su código, integrándose directamente en este entorno de desarrollo (IDE) ampliamente utilizado por los programadores. En este contexto, el trabajo de tesina de nuestros compañeros y colegas Acosta Matias e Ian Mazzaglia titulado “*Extensibilidad en ambientes de desarrollo para la administración de proyectos*” destaca que la integración de herramientas especializadas en los IDEs, no sólo optimiza los procesos de desarrollo, sino que también mejora significativamente la gestión y eficiencia del flujo de trabajo.

## Dominio

El enfoque de este proyecto se encuentra en el ámbito del desarrollo de software, específicamente a la optimización de la productividad y eficiencia en la gestión del código fuente a través de herramientas de control de versiones como Git. Se plantea una extensión para integrarse con Visual Studio Code, un entorno de desarrollo integrado (IDE) ampliamente utilizado por los desarrolladores de software debido a su flexibilidad, personalización, y amplia gama de extensiones. El objetivo principal de este trabajo es añadir funcionalidades que asistan al desarrollador en tareas rutinarias pero críticas, como la creación y manejo de commits, así como la revisión del historial de versiones.

Los repositorios de software son una pieza fundamental en el desarrollo de aplicaciones. Son espacios donde se almacena el código fuente, las versiones y los historiales de cambios, permitiendo a los desarrolladores trabajar de manera colaborativa y mantener un control detallado de la evolución del proyecto. En este contexto, el trabajo propuesto otorga relevancia a los repositorios de software al proporcionar funcionalidades que facilitan la interacción del desarrollador con el repositorio Git. Esto incluye la automatización de tareas como la creación de mensajes de commit, la revisión de diferencias de código y la visualización del historial de commits, lo cual es esencial para mantener la integridad y coherencia del proyecto.

Además, es crucial resaltar que el control de versiones no sólo ayuda a gestionar cambios en el código, sino que también facilita la colaboración entre múltiples desarrolladores. Cuando varios programadores trabajan en un mismo proyecto, es fundamental poder rastrear y entender las contribuciones de cada uno. Una buena herramienta de control de versiones

---

<sup>24</sup> <https://code.visualstudio.com/>

permite a los desarrolladores ver qué cambios han realizado sus compañeros, lo que fomenta una comunicación efectiva y evita conflictos en el código. Esto se traduce en una mayor transparencia dentro del equipo, donde cada miembro puede revisar el trabajo de los demás, discutir implementaciones y realizar correcciones de manera conjunta.

La visualización clara de las diferencias de código y la historia de cambios es otro aspecto importante. Cuando un desarrollador puede ver de manera sencilla cómo ha evolucionado el código a lo largo del tiempo, es más fácil identificar errores y comprender el razonamiento detrás de decisiones específicas. Esta claridad, además de mejorar la calidad del software, permite un aprendizaje continuo entre los miembros del equipo, ya que pueden ver y aprender de las soluciones adoptadas por otros. En definitiva, la integración de estas funcionalidades en la extensión propuesta para Visual Studio Code no solo optimiza las tareas rutinarias, sino que también enriquece el proceso colaborativo, haciendo que el trabajo en equipo sea más fluido y efectivo.

## Alcance

Los alcances del proyecto incluyen:

***Integración con VSCode:*** La extensión está diseñada para integrarse de manera transparente en el flujo de trabajo habitual de los usuarios de Visual Studio Code, sin requerir configuraciones complejas.

***Asistente de Commits:*** Una de las funcionalidades principales de la extensión es la sugerencia automatizada de mensajes de commit, basada en las diferencias detectadas en el código fuente. Esto no sólo mejora la precisión y consistencia de los mensajes, sino que también agiliza el proceso y le quita un peso de encima al desarrollador.

***Catálogo de Historial de Commits:*** La extensión permite acceder de manera fácil y visual al historial de commits del proyecto de software, presentando la información más relevante de los mismos. Esto facilita la revisión y auditoría del código, así como la identificación de cambios importantes en el proyecto y la regresión de versiones.

***Automatización y Simplificación de Tareas Git:*** La extensión proporciona comandos para realizar commits y pushes directamente desde la interfaz de VSCode, simplificando el flujo de trabajo de los desarrolladores que prefieren no utilizar la línea de comandos para estas operaciones. También proporciona funcionalidad de hacer "rollbacks" o "reverts" a un determinado commit, lo que añade una capa extra de control y manejo sobre los cambios en el proyecto.

**Buscador de Commits:** El buscador permite a los usuarios realizar búsquedas dentro del catálogo de commits utilizando criterios expresados en lenguaje natural. Esta funcionalidad filtra y presenta los commits más relevantes que coinciden con lo que el usuario está buscando, mejorando la eficiencia en la localización de cambios específicos dentro del historial del proyecto mediante el análisis del código modificado en cada versión y su información asociada.

## Relevancia

En un entorno donde la colaboración y la eficiencia son factores clave, este trabajo ofrece una solución que no solo mejora la calidad y legibilidad de los repositorios mediante un mejor manejo de versiones, sino que también optimiza el tiempo del desarrollador al automatizar tareas repetitivas quitándoles un peso de encima. Además, al integrarse directamente con VSCode, la herramienta se convierte en un asistente para los desarrolladores que ya utilizan este entorno.

## Modelos LLM y prompt

Con el objetivo de investigar cómo la selección de modelos de IA influye en el desempeño de tareas específicas bajo diferentes instrucciones (prompts), se realizaron pruebas comparativas entre varios modelos conocidos en el campo. Además, se llevaron a cabo experimentaciones utilizando prompts diseñados para mejorar la eficiencia y efectividad de las respuestas de los modelos de IA.

Por tanto, entender cómo funcionan los prompts, cómo se pueden optimizar y cuál es su impacto en las respuestas generadas.

Para lograr este objetivo, se utilizó como fuente un repositorio público con un historial rico en commits (específicamente, el commit [b06cf62](https://github.com/django/django/commit/b06cf62) del repositorio del framework Django<sup>25</sup>) para experimentar con diversos modelos de IA y diferentes prompts, evaluando las respuestas obtenidas (Figura 17).

---

<sup>25</sup> <https://github.com/django/django>

```
tests/forms_tests/tests/tests.py
@@ -259,6 +259,7 @@ def test_unicode_filename(self):
    m.file.name,
    "tests/\u6211\u96bb\u6c23\u588a\u8239\u88dd\u6eff\u6652\u9c54.txt",
    )
+   m.file.delete()
    m.delete()
    def test_boundary_conditions(self):

tests/model_forms/models.py
@@ -135,7 +135,7 @@ def __str__(self):
class Document(models.Model):
-   myfile = models.FileField(upload_to="unused", blank=True)
+   myfile = models.FileField(storage=temp_storage, upload_to="unused", blank=Tr

class TextFile(models.Model):

tests/model_forms/tests.py
@@ -1,5 +1,6 @@
import datetime
import os
+ import shutil
from decimal import Decimal
from unittest import mock, skipUnless

@@ -72,6 +73,7 @@
Triple,
Writer,
WriterProfile,
+ temp_storage_dir,
test_images,
)

class Meta:

class FileAndImageFieldTests(TestCase):
+   def setUp(self):
+       if os.path.exists(temp_storage_dir):
+           shutil.rmtree(temp_storage_dir)
+       os.mkdir(temp_storage_dir)
+       self.addCleanup(shutil.rmtree, temp_storage_dir)
+
    def test_clean_false(self):
        """
        If the ``clean`` method on a non-required FileField receives False as
```

Figura 17. Captura de las diferencias y cambios aplicados en un commit específico del repositorio oficial de Django. ([b06cf62](#))

## Modelo: OpenAI - GPT3.5

### **Prompt**

*Lo anterior es un json con los cambios resultantes de un git diff --cached. Podrías generar un posible mensaje de commit de máximo 2 oraciones contemplando estos cambios?*

### **Mensaje Generado**

Refactorización de modelos y pruebas: Se actualizó el campo `myfile` en `Document` para usar un almacenamiento temporal y se implementó la limpieza de directorios en `FileAndImageFieldTests`.

### **Prompt**

*Escribe un mensaje de commit que refleje las actualizaciones descritas en el diff proporcionado. Detalla cómo se ajustó un archivo específico y qué modificaciones se realizaron, proporcionando contexto sobre el propósito de los cambios.*

### **Mensaje Generado**

Ajustar rutas de archivos y configuración de almacenamiento, y agregar casos de prueba

### **Prompt**

*Basándome en los cambios detectados por git diff --cached, genera un mensaje de commit claro, conciso y descriptivo que explique los cambios realizados en cada archivo.*

### **Mensaje Generado**

Corregir pruebas de nombres de archivo Unicode, actualizar configuración de almacenamiento en Document y agregar limpieza de directorio temporal en pruebas.

### **Prompt**

*Utilizando la convención de tipos de commit, genera mensajes de commit que indiquen el tipo de cambio realizado (por ejemplo, feat, fix, docs, etc.) junto con una descripción breve y clara de los cambios en cada archivo.*

### **Mensaje Generado**

fix: Corregir manejo de archivo Unicode y configuración de campo en pruebas y modelos

- Corregido el manejo de nombres de archivo Unicode y la eliminación adecuada en `tests/forms\_tests/tests/tests.py`.
- Ajustada la configuración del campo 'myfile' en el modelo `Document` en `tests/model\_forms/models.py` para utilizar el almacenamiento temporal correcto.

- Mejorada la limpieza del directorio temporal en `tests/model\_forms/tests.py` para asegurar un entorno de prueba limpio.

Detalles específicos:

- Asegurado la eliminación adecuada de archivos en pruebas de nombres de archivo Unicode.
- Configuración corregida del campo 'myfile' para el modelo `Document`.
- Implementada limpieza de directorio temporal para prevenir problemas en pruebas consecutivas.

### **Prompt**

*Utiliza el JSON proporcionado como base para generar un mensaje de commit que resuma los cambios efectuados en la aplicación. Puedes considerar las adiciones y modificaciones en los archivos mencionados anteriormente. Asegúrate de ser preciso y descriptivo en tu descripción, y no dudes en utilizar términos técnicos relevantes para la aplicación."*

### **Mensaje Generado**

Añade soporte para nombres de archivo Unicode en pruebas y ajusta el almacenamiento temporal en FileField. Se implementan estas mejoras para asegurar compatibilidad con nombres de archivo Unicode y mejorar la gestión del almacenamiento temporal en los tests.

Modelo: Google - Gemini

### **Prompt**

*Lo anterior es un json con los cambios resultantes de un git diff --cached. Podrias generar un posible mensaje de commit de máximo 2 oraciones contemplando estos cambios?*

### **Mensaje Generado**

Actualizaciones en pruebas y modelos de formulario: se agregaron validaciones para nombres de archivo Unicode y se corrigió un error en el campo myfile.

### **Prompt**

*Escribe un mensaje de commit que refleje las actualizaciones descritas en el diff proporcionado. Detalla cómo se ajustó un archivo específico y qué modificaciones se realizaron, proporcionando contexto sobre el propósito de los cambios.*

### **Mensaje Generado**

Manejo de carga de archivos refactorizado y pruebas de condiciones límite agregadas

**Prompt**

*Basándome en los cambios detectados por git diff --cached, genera un mensaje de commit claro, conciso y descriptivo que explique los cambios realizados en cada archivo.*

**Mensaje Generado**

Implementa soporte para nombres de archivo Unicode, actualiza el almacenamiento de archivos en el modelo Document y configura pruebas para FileField e ImageField.

**Prompt**

*Utilizando la convención de tipos de commit, genera mensajes de commit que indiquen el tipo de cambio realizado (por ejemplo, feat, fix, docs, etc.) junto con una descripción breve y clara de los cambios en cada archivo.*

**Mensaje Generado**

fix: Se corrigió un error en la prueba test\_unicode\_filename de tests/forms\_tests/tests/tests.py relacionado a la manipulación de nombres de archivos con caracteres no ASCII.

refactor: Se refactorizó la definición del campo myfile en el modelo Document de tests/model\_forms/models.py para utilizar el almacenamiento temporal (temp\_storage).

test: Se agregaron nuevas pruebas unitarias en la clase FileAndImageFieldTests de tests/model\_forms/tests.py para verificar el comportamiento del campo myfile en diferentes escenarios.

**Prompt**

*Utiliza el JSON proporcionado como base para generar un mensaje de commit que resuma los cambios efectuados en la aplicación. Puedes considerar las adiciones y modificaciones en los archivos mencionados anteriormente. Asegúrate de ser preciso y descriptivo en tu descripción, y no dudes en utilizar términos técnicos relevantes para la aplicación."*

**Mensaje Generado**

Actualizaciones de Pruebas de Archivos e Imágenes. Este commit introduce cambios en los archivos de prueba relacionados con la funcionalidad de archivos e imágenes.

Meta - Llama3

**Prompt**

*Lo anterior es un json con los cambios resultantes de un git diff --cached. Podrias generar un posible mensaje de commit de maximo 2 oraciones contemplando estos cambios?*

### **Mensaje Generado**

Se han agregado pruebas para archivos con nombres en código Unicode y se ha añadido soporte para almacenar archivos temporalmente en el directorio 'unused'

### **Prompt**

*Escribe un mensaje de commit que refleje las actualizaciones descritas en el diff proporcionado. Detalla cómo se ajustó un archivo específico y qué modificaciones se realizaron, proporcionando contexto sobre el propósito de los cambios.*

### **Mensaje Generado**

Actualización de archivos relacionados con formularios y modelos: Se han realizado ajustes significativos en diferentes archivos para mejorar la funcionalidad y flexibilidad de los formularios y modelos en nuestra aplicación.

### **Prompt**

*Basándome en los cambios detectados por git diff --cached, genera un mensaje de commit claro, conciso y descriptivo que explique los cambios realizados en cada archivo.*

### **Mensaje Generado**

Actualizaciones en pruebas y modelos: Se han realizado ajustes en tests/forms\_tests/tests.py para agregar soporte a nombres de archivos unicode y mejorar la funcionalidad de los formularios. Además, se ha agregado una clase TextFile en models.py y se ha implementado un método setUp para crear un directorio temporal en tests/model\_forms/tests.py

### **Prompt**

*Utilizando la convención de tipos de commit, genera mensajes de commit que indiquen el tipo de cambio realizado (por ejemplo, feat, fix, docs, etc.) junto con una descripción breve y clara de los cambios en cada archivo.*

### **Mensaje Generado**

feat: Mejoras en pruebas y modelos

\* tests/forms\_tests/tests.py: Agregado soporte a nombres de archivos unicode en el método `test\_unicode\_filename` para mejorar la funcionalidad de los formularios.

\* tests/model\_forms/models.py: Se ha agregado una clase TextFile para manejar archivos de texto y se han realizado ajustes para agregar soporte para archivos temporales.

\* tests/model\_forms/tests.py: Se ha implementado un método setUp para crear un directorio temporal y eliminarlo después de ejecutar los tests, lo que mejora la robustez y la escalabilidad de nuestra aplicación.

### **Prompt**

*Utiliza el JSON proporcionado como base para generar un mensaje de commit que resuma los cambios efectuados en la aplicación. Puedes considerar las adiciones y modificaciones en los archivos mencionados anteriormente. Asegúrate de ser preciso y descriptivo en tu descripción, y no dudes en utilizar términos técnicos relevantes para la aplicación."*

### **Mensaje Generado**

Mejoras en pruebas y modelos para manejo de archivos

## Elección de modelo LLM

Para la selección del modelo de lenguaje, se ha optado por Gemini<sup>26</sup> desarrollado por Google DeepMind, debido a varias razones que hacen de este modelo una opción atractiva para nuestras necesidades:

**Gratuito y Poderoso:** Gemini es una solución robusta y avanzada que ofrece capacidades de procesamiento de lenguaje natural de alto nivel sin costo. Esto es conveniente para poder lograr los objetivos planteados en el presente trabajo, ya que el acceso a Gemini a través de Google Cloud permite a los usuarios beneficiarse de su potencia sin costos iniciales. Su capacidad para manejar tareas complejas como la generación de texto, la traducción y el análisis semántico lo convierte en una herramienta poderosa para una amplia gama de aplicaciones.

**Acceso Mediante API:** La disponibilidad de Gemini a través de una API proporciona una gran flexibilidad y facilidad de integración. Esto permite incorporar todas las funcionalidades del modelo en las aplicaciones sin preocuparte por el hardware o la infraestructura.

**Limitaciones y Posibilidades de Mejora:** Aunque Gemini es una opción potente y accesible, presenta ciertas limitaciones. Por ejemplo, los modelos basados en Gemini pueden tener restricciones en cuanto a la cantidad de tokens procesados en cada solicitud, lo que puede afectar la capacidad para manejar textos extremadamente largos o complejos. Además,

---

<sup>26</sup> <https://ai.google.dev/>

mientras que el modelo es gratuito, las funcionalidades avanzadas y el uso extensivo pueden estar sujetos a tarifas basadas en el volumen de datos procesados. En estos casos, los planes de pago podrían ofrecer mejoras significativas, como un mayor límite de tokens y tiempos de respuesta optimizados.

**Comparación con Modelos locales (LLama 3/3.1 - Meta):** Aunque los modelos de Meta, como LLaMA 3.1 o LLama 3, ofrecen una alternativa con acceso libre, características muy similares y sin problemas de tokens, la decisión de no optar por estos modelos se basa en varias consideraciones. Los modelos de Meta requieren recursos hardware significativos para su ejecución local, incluyendo hardware especializado como GPUs de alta gama, alto nivel de memoria RAM y procesadores de alto rendimiento. Este requisito de recursos puede ser una barrera para proyectos con limitaciones presupuestarias o para aquellos que prefieren una solución más sencilla y manejable.

En conclusión, la elección de Gemini se basa en su capacidad para ofrecer un alto rendimiento sin costo inicial, su facilidad de integración a través de una API y su flexibilidad en comparación con modelos que requieren una infraestructura local considerable. Aunque existen limitaciones en cuanto a tokens y costos asociados con el uso extensivo y excesivo, hace que el modelo Gemini sea una opción conveniente para los objetivos de este trabajo.

## Funcionalidades

La extensión cuenta con dos paneles principales correspondientes a las funcionalidades centrales de la misma: el asistente de sugerencias para Mensajes de Commit y el Catálogo de Commits anteriores. La primera cumple la función de proveer al desarrollador de opciones de mensajes descriptivos, adaptados a los cambios pendientes de commit en su proyecto, mientras que la tarea de la segunda es la de suministrar un historial de commits fácilmente visualizable y con búsquedas intuitivas, expresadas en lenguaje natural.

Ambas pueden accederse presionando la combinación de teclas correspondiente.

Tabla 2. Atajos de Teclado

Panel	Combinación de Teclas
Asistente de Mensajes	<i>CTRL+SHIFT+C</i>
Catálogo de Commits	<i>CTRL+SHIFT+H</i>

Para el correcto funcionamiento de la extensión, el IDE Visual Studio Code debe tener abierto un proyecto Git al momento de hacer uso de estas funcionalidades.

## Asistente de Mensajes

Al abrir este panel, se presenta una pantalla de bienvenida, la cual cuenta con una breve descripción de la funcionalidad principal del panel y un botón que inicia la ejecución del proceso de generación de mensajes.

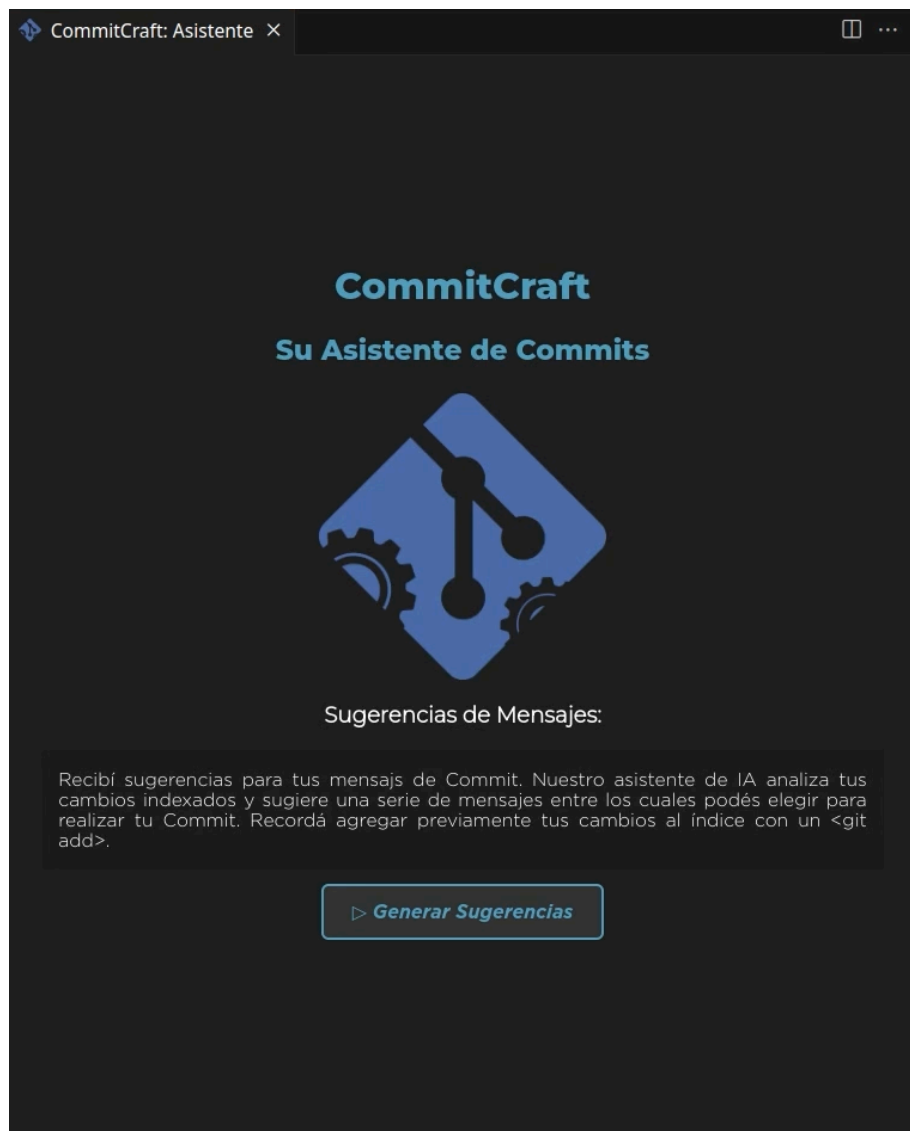


Figura 18. Pantalla de inicio del “Asistente de commits”. La misma se presenta luego de accionar el comando para el asistente de mensajes (*CTRL+SHIFT+C*).

La extensión extrae la información del proyecto git haciendo uso de la librería de Typescript “simpleGit”, accediendo de esta manera a los cambios indexados en la caché de git, es decir, aquellos que hayan sido agregados previamente mediante el comando *git add*.

Una vez extraídos los cambios, se envían al módulo python que llamamos Módulo de Entrada, el cual realiza todo el preprocesamiento necesario antes de mandarlo al llamado Módulo Analítico.

Una vez en este, se conforma un prompt anexando esta información a una directiva previamente definida por nosotros. Luego, se configura la conexión con la API de Gemini y se envía el prompt, esperando como respuesta un mensaje de commit, en formato de *string*, que cumpla con todas las condiciones que le hayamos exigido previamente dentro de la directiva. Esta comunicación con la API se hace una *n* cantidad de veces, actualmente establecida en cinco (5), se colocan en un arreglo y se envían al Módulo de Salida.

En el Módulo de Salida, se genera un nuevo arreglo donde cada elemento del mismo es un diccionario para representar e identificar cada mensaje. Este luego se convierte a JSON y se devuelve como respuesta al panel de la extensión

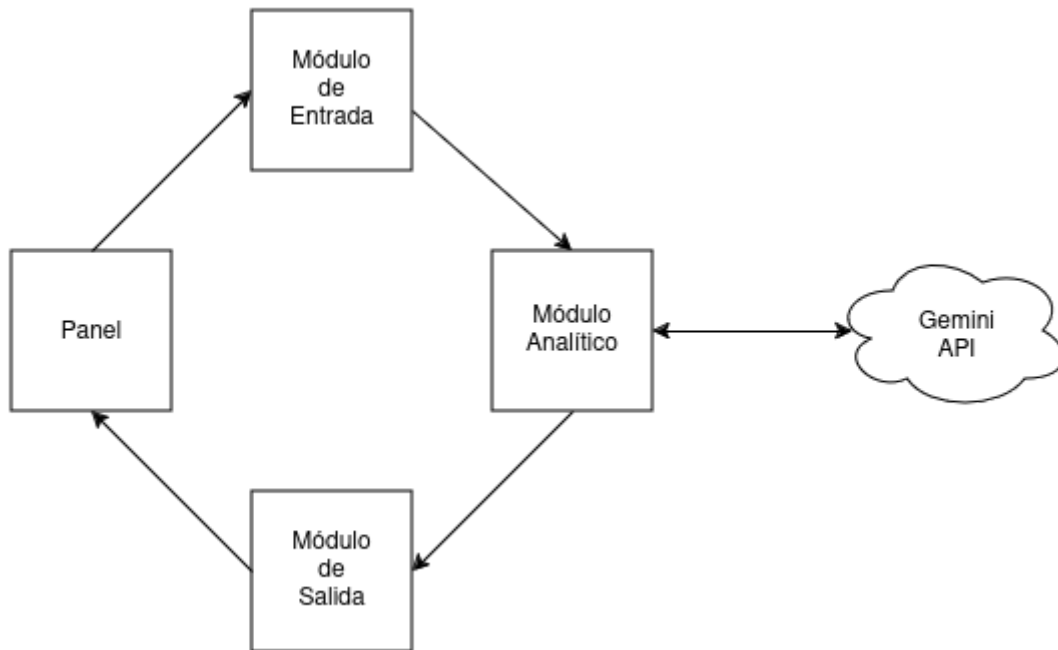


Figura 19. Comunicaciones entre módulos y componentes involucrados para la generación de mensajes de commit.

Durante los segundos que dura este proceso, se le muestra al usuario una pantalla de carga.

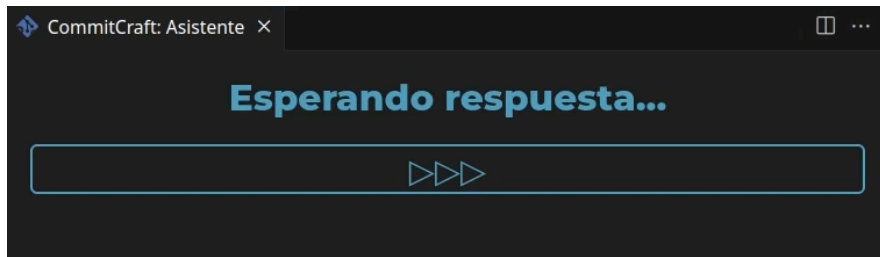


Figura 20. Pantalla de carga o espera. La misma se muestra mientras se procesa la generación de mensajes de commits.

Al recibir la respuesta por parte del Módulo de Salida, la extensión toma el JSON, lo parsea y lo muestra en forma de listado de elementos seleccionables. En esta nueva pantalla de respuesta, se presentan tres botones flotantes en la región inferior: Actualizar Sugerencias, Commit y Commit & Push.

El primer botón “Actualizar Sugerencias” repite el proceso previamente descrito, generando una nueva serie de mensajes de commit. De esta manera, el usuario tiene la posibilidad de generar mensajes tantas veces como lo desee, hasta encontrar uno con el que esté conforme.

En cuanto a los botones Commit y Commit & Push, estos ejecutan las acciones que sus nombres indican. El botón Commit realiza la operación de generar un commit utilizando el mensaje previamente seleccionado por el usuario. Por otro lado, el botón Commit & Push no sólo efectúa el commit, sino que también lleva a cabo la acción de enviar los cambios al repositorio remoto, solicitando las credenciales del usuario para completar el proceso de autenticación.



Figura 21. Pantalla en la que se muestran las diferentes sugerencias de mensajes de commits, basándose en las modificaciones realizadas.

## Catálogo de Commits

Análogamente al panel de Generación de Mensajes, al abrir este panel, se presenta otra pantalla de bienvenida con una breve descripción de la funcionalidad principal del mismo y un botón que inicia la ejecución del proceso de recopilación y presentación de commits históricos.

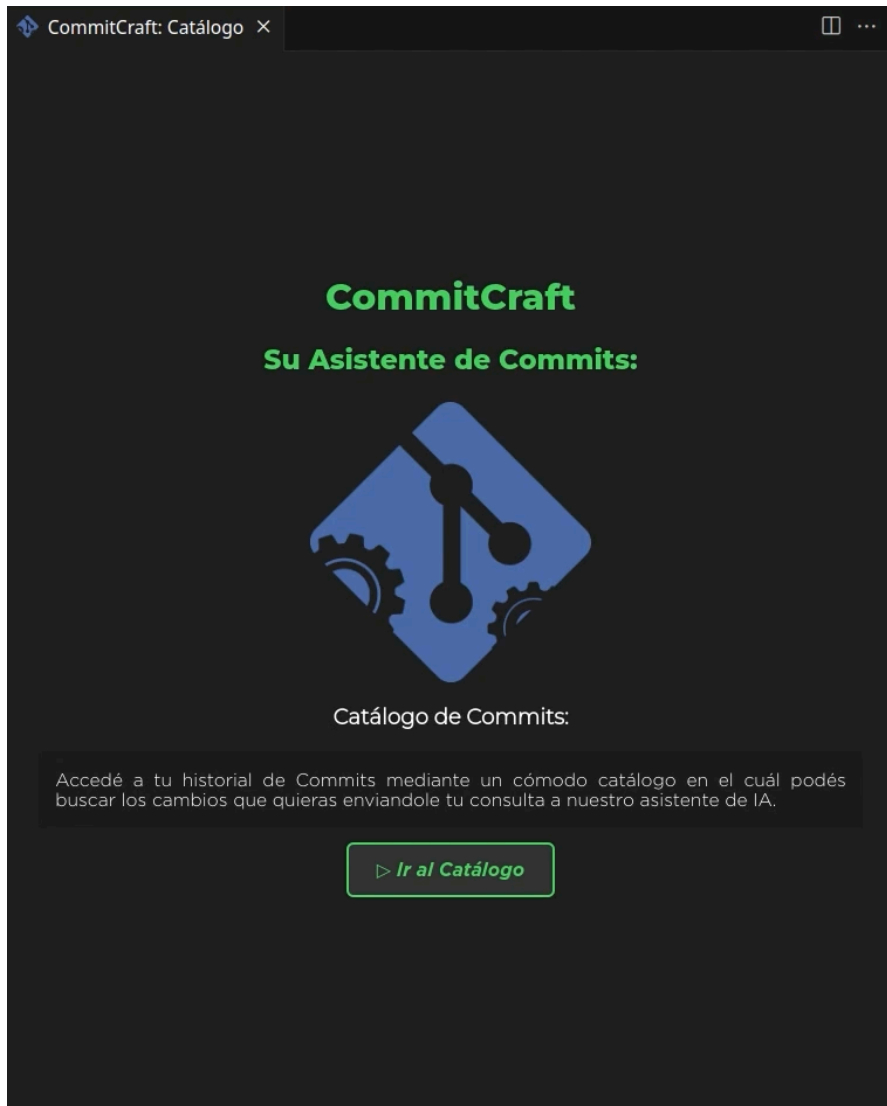


Figura 22. Pantalla de inicio de “Catálogo de commits”. La misma se presenta luego de accionar el comando para el asistente de mensajes (*CTRL+SHIFT+H*).

Con el fin de generar un catálogo histórico fácilmente visualizable por el desarrollador, la extensión recopila todo el registro de commits del proyecto Git en el cual se está trabajando, haciendo uso nuevamente de la librería de Typescript “simpleGit”.

Recorriendo este registro, se crea una serie de diccionarios siguiendo una interfaz “Commit” modelada por nosotros. Cada uno de los mismos contiene información sobre el autor, fecha, hash asociado, mensaje de commit y finalmente, a pesar de que no se muestra de manera directa en el catálogo, los cambios realizados en relación al commit anterior. A medida que se generan estos diccionarios, se agregan a un arreglo el cual finalmente se transforma a un JSON para un eventual uso posterior.

Similarmente que con la funcionalidad anterior, durante los segundos que dura este proceso, se le muestra al usuario una pantalla de carga, casi idéntica.

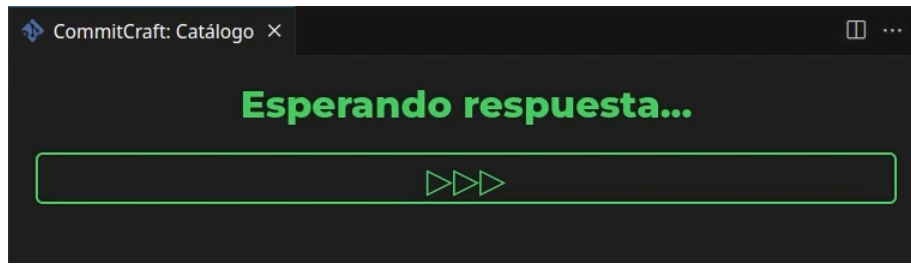


Figura 23. Pantalla de carga o espera. La misma se muestra mientras se procesa y genera el historial de commits del repositorio.

La extensión itera el arreglo y representa cada uno de los commits en forma de un listado de elementos seleccionables, ordenados cronológicamente, del más reciente al más antiguo. En la parte inferior se muestran dos botones flotantes: Actualizar Catálogo Rollback.

Mientras que el botón “Actualizar Catálogo” vuelve a ejecutar el proceso inicial de carga de esta pantalla, trayendo nuevamente toda la información relevante asociada al proyecto, el botón “Rollback” permite revertir commit que haya sido previamente seleccionado por el usuario.

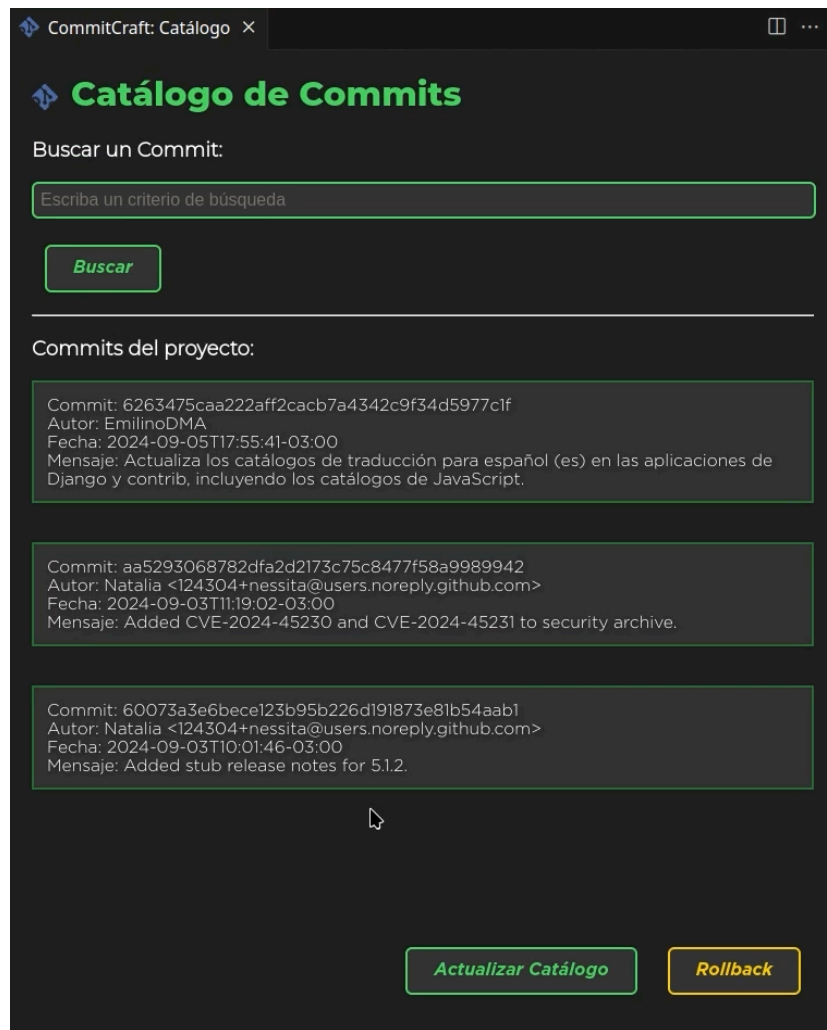


Figura 24. Pantalla en la que se muestran, por un lado, el historial de commits del repositorio. Y por otro lado, un buscador de commits mediante lenguaje natural.

Finalmente, en la sección superior del panel, se encuentra un apartado dedicado a la función de búsqueda. En ella se ubica un recuadro que permite al usuario ingresar un criterio de búsqueda expresado en lenguaje natural. Al presionar el botón “Buscar”, el texto ingresado se envía a un módulo python el cual llamamos “Módulo de Búsqueda”, en conjunto con el JSON generado durante la carga del catálogo.

Dentro del Módulo de Búsqueda, se define un nuevo prompt compuesto por una directiva establecida por nosotros y expresada en lenguaje natural, concatenada al JSON que representa a todos los commits del catálogo donde se buscará y al criterio de búsqueda del usuario. Este prompt se envía a la API de Gemini y se recibe una respuesta, la cual se transforma a un arreglo donde cada elemento es el hash de un commit que cumpla con el criterio de búsqueda.

Una vez retornada la respuesta al panel, se recorre el arreglo de hashes y se trae la información de cada commit correspondiente mediante el uso de la librería “simpleGit”,

creando un nuevo listado filtrado que se muestra en pantalla, en una interfaz idéntica a la del catálogo general.



Figura 25. Comunicaciones entre módulos y componentes involucrados para la búsqueda de commits en el catálogo.

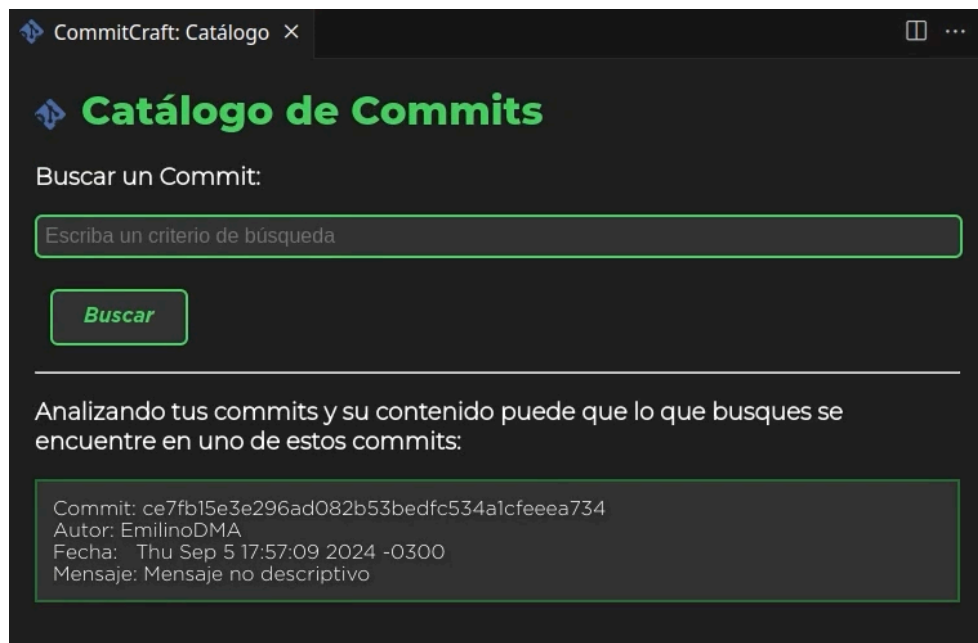


Figura 26. Pantalla de búsqueda, donde el usuario puede ingresar un criterio de filtrado expresado en lenguaje natural.

## Experimentación

A continuación, se describen una serie de experimentaciones que se realizaron con el software desarrollado, expuesto a diferentes casos de uso.

En cuanto al caso de prueba 1, se describe una experimentación más detallada, pasando por todas las funcionalidades de la extensión. Para los demás casos de prueba, se omiten algunas funcionalidades las cuales no se ven afectadas respecto al contenido y lenguaje del repositorio.

Para todos los experimentos se limitó la cantidad de commits del catálogo a 3, con el fin de evitar llegar al límite de la cuota disponible.

## Caso 1

Utilizando el repositorio oficial del framework de desarrollo Django, disponible en github, se ejecuta la extensión. En la siguiente imagen puede verse un ejemplo de cómo se ve la misma integrada al IDE.

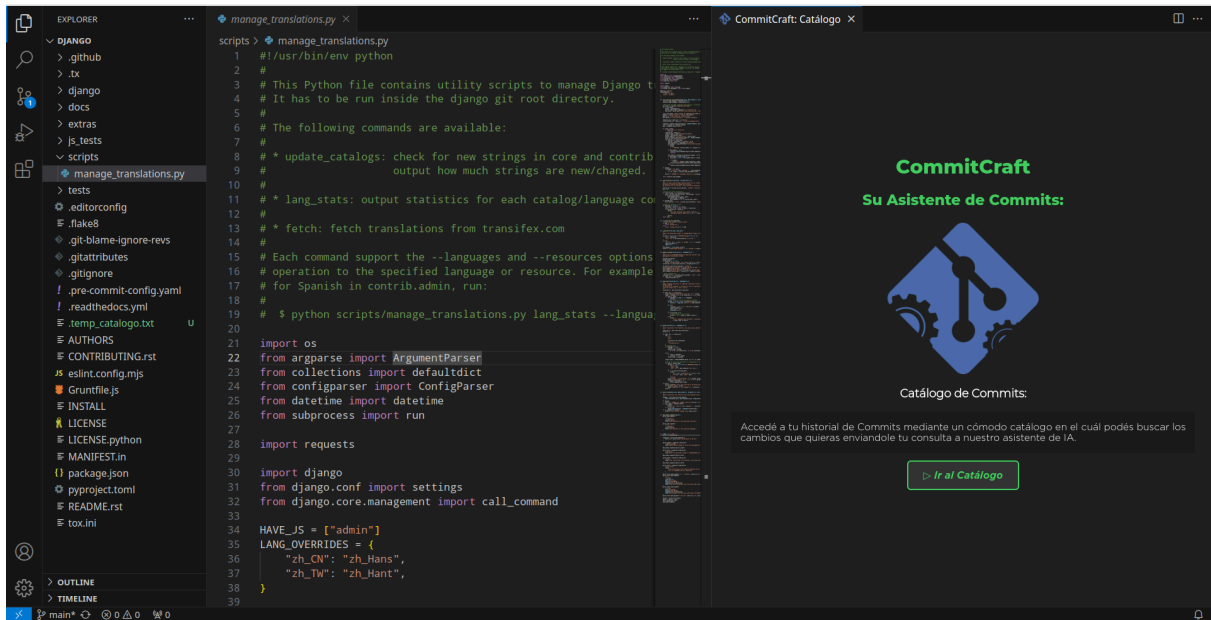


Figura 27. Vista de la extensión integrada al IDE, con el repositorio de Django abierto.

Para esta experimentación, se realizará un cambio en el código del archivo *manage\_translation.py*, escrito en Python. Previo a esto, se ejecutó la funcionalidad del catálogo para visualizar el listado de commits del proyecto, anterior a cualquier modificación.

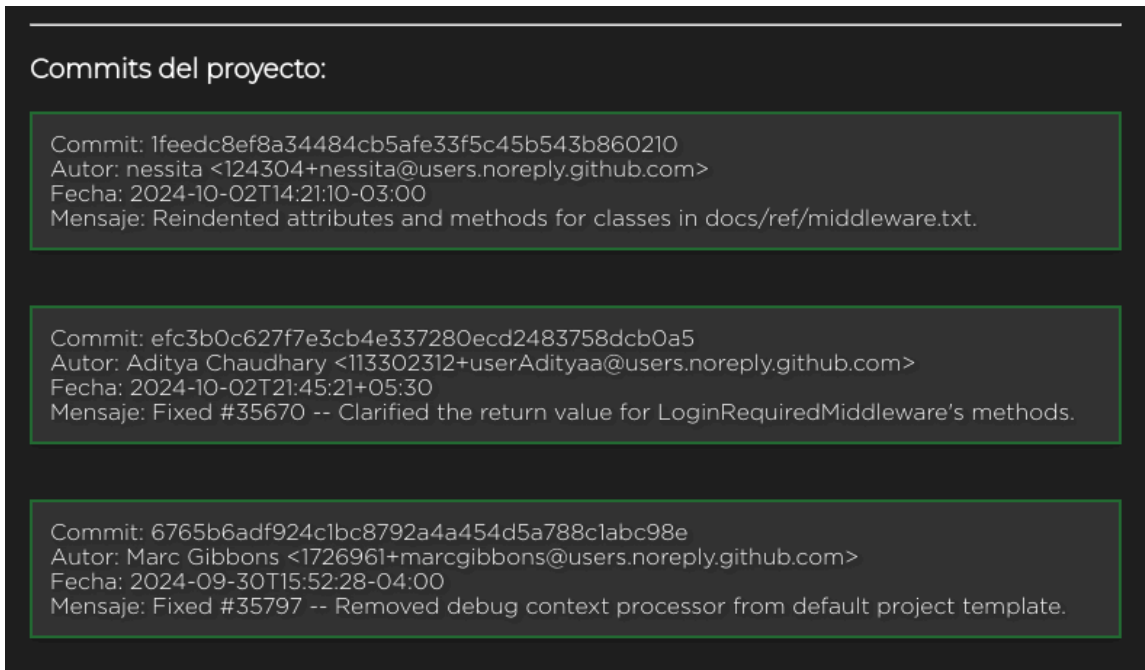


Figura 28. Listado de Commits previo a la experimentación.

La extensión muestra los commits subidos en la rama actual (main), ordenados desde el más reciente al más antiguo de forma descendente. En la figura siguiente se muestra una captura de github, a fin de demostrar que la información mostrada por la aplicación es correcta.

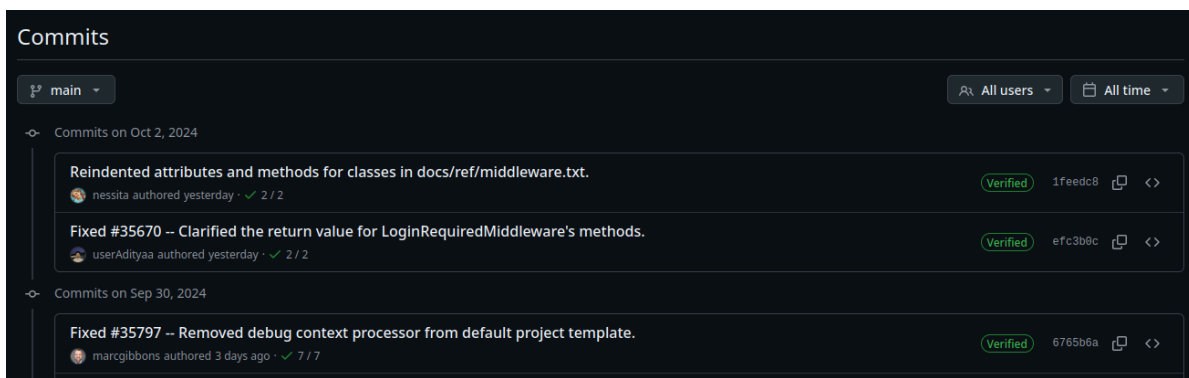


Figura 29. Listado de Commits vistos desde Github al momento de ejecutar la experimentación.

A continuación se procede a realizar un cambio en el código del repositorio. La naturaleza del cambio no es de nuestra incumbencia siempre y cuando sea tal que podamos identificarlo y determinar que las consiguientes respuestas de la extensión sean correctas.

El cambio realizado fue la modificación del nombre de la función `update_catalogs`, definida en el archivo previamente dicho, por `“actualizar_catalogos”`.

```

diff --git a/scripts/manage_translations.py b/scripts/manage_translations.py
index bc569fbd0a..564cd29123 100644
--- a/scripts/manage_translations.py
+++ b/scripts/manage_translations.py
@@ -159,7 +159,7 @@ def _check_diff(cat_name, base_path):
     print("%d changed/added messages in '%s' catalog." % (num_changes, cat_name))

-def update_catalogs(resources=None, languages=None):
+def actualizar_catalogos(resources=None, languages=None):
     """
     Update the en/LC_MESSAGES/django.po (main and contrib) files with
     new/updated translatable strings.
(END)

```

Figura 30. Ejecución del comando “*git diff --cached*”, mostrando los cambios realizados en el archivo *scripts/manage\_translations.py*.

Con el objetivo de hacer pruebas sobre la funcionalidad de búsqueda, se realiza un commit del cambio usando la cadena “Mensaje no descriptivo” como mensaje de commit. Luego, se realiza una búsqueda desde la extensión, ingresando un criterio de búsqueda expresado en lenguaje natural. La cadena ingresada contiene intencionalmente errores de tipeo, incluso en términos importantes tales como los nombres de las funciones. Además se usa una abreviación informal de la palabra “por”, tal como puede llegar a verse en conversaciones en línea y textos no formales.

Desde el punto de vista del código, una variación de un caracter en un identificador tal como son los nombres de las funciones significaría un error a la hora de reconocer al mismo por parte de un compilador o un intérprete. Lo mismo sucedería para la búsqueda si esta se basara únicamente en la identificación de patrones de texto.

El propósito de insertar estos fallos es tanto el de ingresar un factor humano, como el de probar que el análisis hecho por la IA se basa en el contenido del cambio realizado en el código del commit y no en la búsqueda de patrones de texto.

La cadena propiamente dicha fue la siguiente:

*“necesito sber donde cambie el nombre de la funcion update:catalogs x actualizar\_catalogos”*

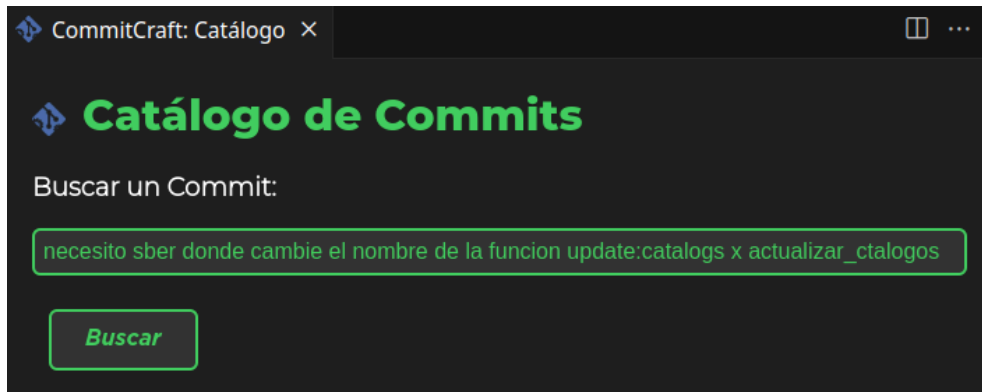


Figura 31. Sección de búsqueda de la extensión con un criterio de búsqueda expresado en lenguaje natural, con términos coloquiales y errores de tipeo.

Como se puede ver en la figura siguiente, la extensión devolvió como resultado a el commit esperado, demostrando el correcto funcionamiento de la aplicación.

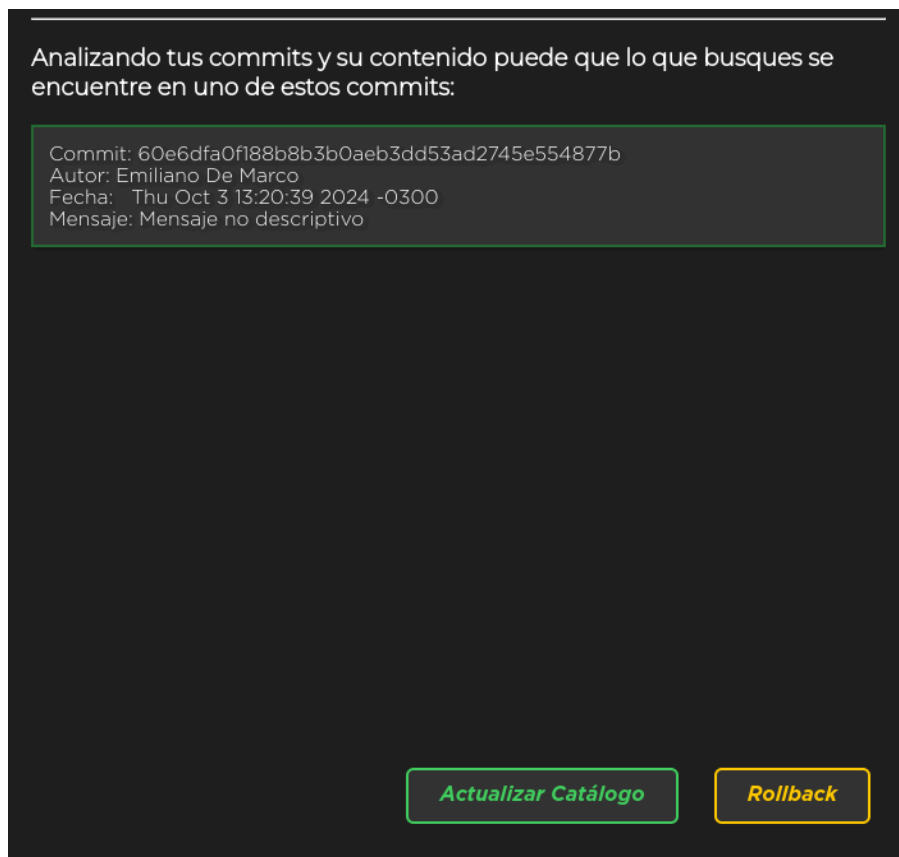


Figura 32. Resultado de la búsqueda realizada en la figura anterior.

A continuación, se prueba brevemente la funcionalidad de rollback del panel de catálogo de la extensión. Para esto se seleccionó el segundo commit del listado y luego se presionó el botón de Rollback.

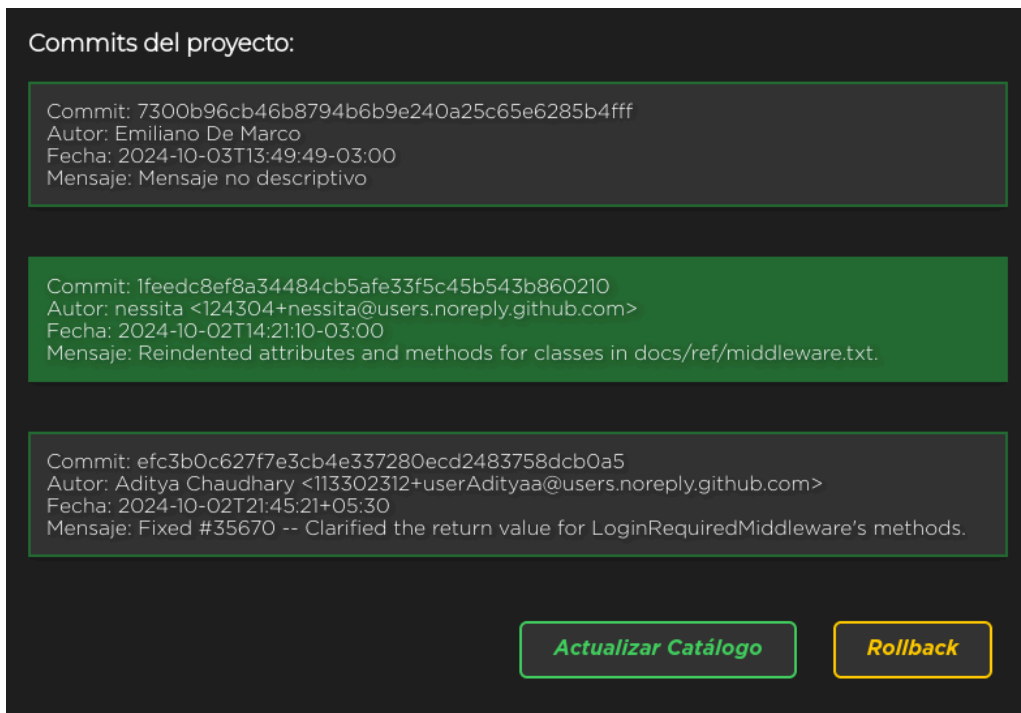


Figura 32. Listado de commits en el catálogo, previo a la ejecución de la funcionalidad de rollback, resaltando el elemento seleccionado a hacer revert.

Luego de la ejecución de la funcionalidad, al recargar el catálogo, puede verse claramente cómo se generó un nuevo commit correspondiente al revert. El mismo, además de la fecha actual y el nuevo hash, contiene el nombre del usuario que realizó el rollback y el mensaje del commit restablecido, anteponiendo la palabra “Revert”, como se muestra en la figura siguiente.

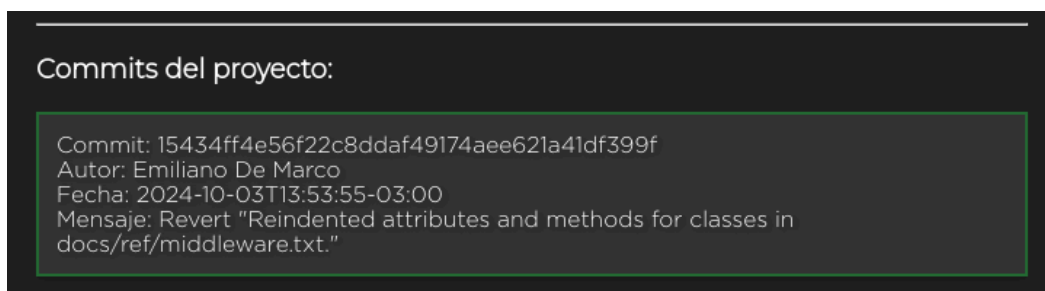


Figura 33. Nuevo commit generado tras la ejecución de un rollback.

Finalmente, se prueba la funcionalidad de generación de mensajes de commit. Para esto, se realizó otro cambio sobre el código del archivo *scripts/manage\_translations.py*. El cambio realizado fue hecho dentro de la función previamente renombrada “*actualizar\_catalogos*” y consistió en reemplazar la selección del idioma de inglés a español. Es decir, se cambió la definición `locale=["en"]` por `locale=["es"]`.

```
diff --git a/scripts/manage_translations.py b/scripts/manage_translations.py
index 564cd29123..bdd7dda9f7 100644
--- a/scripts/manage_translations.py
+++ b/scripts/manage_translations.py
@@ -172,9 +172,9 @@ def actualizar_catalogos(resources=None, languages=None):

    os.chdir(os.path.join(os.getcwd(), "django"))
    print("Updating en catalogs for Django and contrib apps...")
-   call_command("makemessages", locale=["en"])
+   call_command("makemessages", locale=["es"])
    print("Updating en JS catalogs for Django and contrib apps...")
-   call_command("makemessages", locale=["en"], domain="djangojs")
+   call_command("makemessages", locale=["es"], domain="djangojs")

    # Output changed stats
    _check_diff("core", os.path.join(os.getcwd(), "conf", "locale"))
~
~
~
(END)
```

Figura 34. Ejecución del comando “*git diff --cached*”, mostrando los cambios realizados dentro de la función *actualizar\_catalogos*.

Tras agregar el cambio al índice de git con el comando “*git add*” se procedió a ejecutar la funcionalidad de generación de sugerencias. Los mensajes de commit sugeridos fueron los mostrados en la figura a continuación.

## 📌 Mensajes de Commit sugeridos

Tras analizar los cambios en el índice de su proyecto, sugiero los siguientes posibles mensajes de commit:

- Sugerencia 1:**  
Se actualizan los catálogos de traducción para el idioma español en Django y sus aplicaciones. Se actualiza el idioma de los catálogos de Javascript para Django y sus aplicaciones.
- Sugerencia 2:**  
Actualización de los catálogos de traducción para el idioma español en el proyecto Django, incluyendo tanto el core como el javascript.
- Sugerencia 3:**  
Actualiza las traducciones al español para las aplicaciones de Django y contrib.
- Sugerencia 4:**  
Se actualizan los catálogos de traducción para el idioma español en Django y sus aplicaciones, incluyendo los catálogos de JavaScript.
- Sugerencia 5:**  
Actualiza los catálogos de traducción para el idioma español (es) en las aplicaciones de Django y sus componentes.

Figura 35. Listado de sugerencias devueltas por la extensión durante la experimentación.

Se seleccionó la Sugerencia 2 y se realizó el commit pulsando el botón correspondiente en el área inferior del panel. Para comprobar la correcta ejecución de esta funcionalidad, se cargó nuevamente el catálogo y se corroboró la existencia del nuevo commit en el mismo.

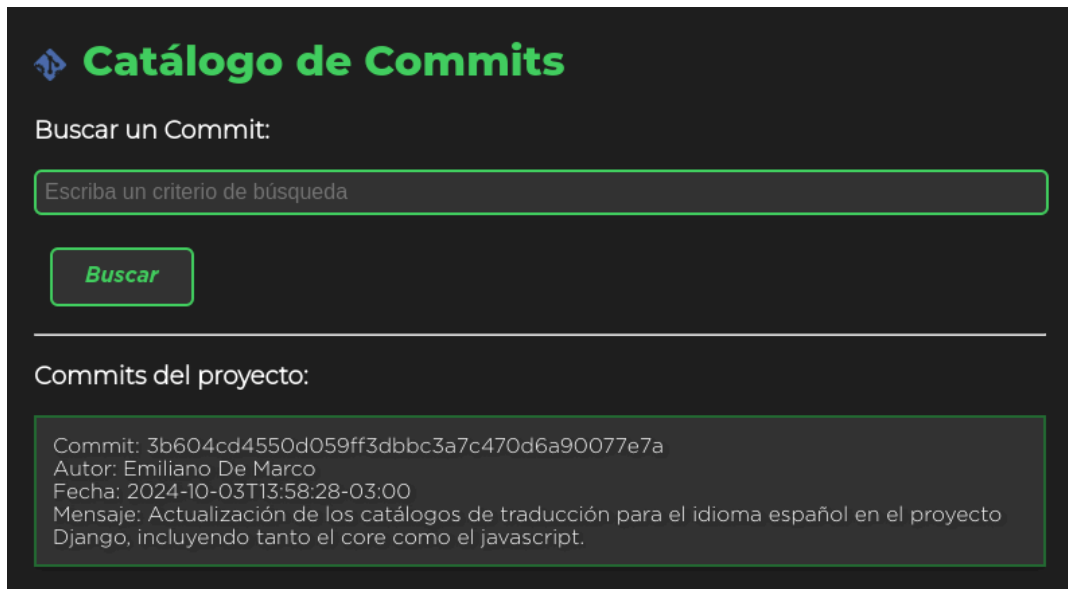


Figura 36. Catálogo de Commits donde el mensaje del último ítem fue generado y sugerido por la extensión.

## Caso 2

Para el siguiente caso de prueba, mucho más breve, tomamos el repositorio oficial del servidor web Nginx<sup>27</sup> escrito principalmente en C.

En esta ocasión, la elección del repositorio fue dada con el fin de realizar una experimentación con un proyecto de similar envergadura pero diferente lenguaje, de este modo demostrando que el éxito de los resultados no están ligados a un único lenguaje. A su vez, en lugar de modificar nuevamente código existente, decidimos extender alguna funcionalidad, evaluando así el comportamiento de la extensión ante contenido nuevo.

Primero clonamos el repositorio localmente y dentro del archivo de código *src/core/nginx.c* agregamos una nueva funcionalidad. La misma devuelve el tiempo que el servidor lleva en ejecución continua y puede verse en la siguiente figura.

---

<sup>27</sup> <https://github.com/nginx/nginx>

```
#include <nginx.h>

-
+static time_t ngx_start_time;
static void ngx_show_version_info(void);
static ngx_int_t ngx_add_inherited_sockets(ngx_cycle_t *cycle);
static void ngx_cleanup_environment(void *data);
@@ -31,6 +31,21 @@ static char *ngx_load_module(ngx_conf_t *cf, ngx_command_t *cmd, void *conf);
static void ngx_unload_module(void *data);
#endif

+static void ngx_show_uptime(void) {
+    time_t now = time(NULL);
+    time_t uptime = now - ngx_start_time;
+
+    int days = uptime / (24 * 3600);
+    uptime %= (24 * 3600);
+    int hours = uptime / 3600;
+    uptime %= 3600;
+    int minutes = uptime / 60;
+    int seconds = uptime % 60;
+
+    ngx_log_stderr(0, "Nginx Uptime: %d days, %d hours, %d minutes, %d seconds",
+    days, hours, minutes, seconds);
+}
+
static ngx_conf_enum_t ngx_debug_points[] = {
    { ngx_string("stop"), NGX_DEBUG_POINTS_STOP },
    (END)
```

Figura 37. Ejecución del comando “*git diff --cached*”, mostrando los cambios realizados en el archivo *src/core/nginx.c*.

Luego se procedió a agregar los cambios al índice del proyecto git mediante el comando “*git add*” y a ejecutar la funcionalidad de generación de mensajes de commit. Los mensajes resultantes fueron los siguientes.

- Sugerencia 1:**  
Se agrega la funcionalidad de mostrar el tiempo de actividad de Nginx en el registro de errores.
- Sugerencia 2:**  
Se agrega la funcionalidad de mostrar el tiempo de actividad de Nginx en el registro de errores.
- Sugerencia 3:**  
Se agrega el cálculo y la visualización del tiempo de actividad de Nginx en el registro de errores.
- Sugerencia 4:**  
Se agregó la funcionalidad de mostrar el tiempo de funcionamiento de Nginx en el log de errores.
- Sugerencia 5:**  
Se agrega la funcionalidad de mostrar el tiempo de actividad del servidor Nginx en el log.

Figura 38. Listado de sugerencias devueltas por la extensión durante la experimentación.

Como se puede apreciar, los mensajes sugeridos por la extensión tienen coherencia respecto a la naturaleza de la función desarrollada, dando por exitosa esta experimentación.

### Caso 3

Como tercer y último caso de prueba se buscó un repositorio de gran tamaño aunque no tan difundido y en un lenguaje, si bien ampliamente conocido, menos popular que sus contrapartes anteriores Python y C. El proyecto elegido fue *WeakAuras2*<sup>28</sup>, un framework utilizado para extender la interfaz de usuario del videojuego World of Warcraft y escrito en el lenguaje multiparadigma Lua.

Con esta nueva experimentación se busca poner a prueba no solo los puntos vistos en el Caso 2 respecto al lenguaje sino, además, la capacidad de la extensión de comprender el contexto y propósito de los cambios que se están realizando, incluso en un dominio tan particular como el elegido.

El archivo a modificar fue *WeakAuras/BossMods.lua* en el cual se agregó una nueva funcionalidad. El código añadido puede verse en la figura siguiente.

```
diff --git a/WeakAuras/BossMods.lua b/WeakAuras/BossMods.lua
index 015313d3..c238a0a5 100644
--- a/WeakAuras/BossMods.lua
+++ b/WeakAuras/BossMods.lua
@@ -1713,3 +1713,23 @@ Private.event_prototypes["Boss Mod Timer"] = {
     Private.category_event_prototype.addons["Boss Mod Timer"] = L["Boss Mod Timer"]

+local function OnPhaseChange(phaseNumber)
+    local phaseMessage = string.format(";Fase %d iniciada!", phaseNumber)
+    WeakAuras.ShowMessage(phaseMessage)
+    PlaySoundFile("Interface\\AddOns\\WeakAuras\\Media\\Sounds\\PhaseChange.ogg", "Master")
+end
+
+-- Registra el evento para el cambio de fase
+local frame = CreateFrame("Frame")
+frame:RegisterEvent("CHAT_MSG_RAID_BOSS_EMOTE")
+
+frame:SetScript("OnEvent", function(self, event, msg)
+    if msg:find("cambia de fase") then
+        local phaseNumber = GetPhaseNumber(msg)
+        OnPhaseChange(phaseNumber)
+    end
+end)
+
+function GetPhaseNumber(msg)
+    return tonumber(msg:match("Fase (%d+)"))
+end
+
+~
+~
+(END)
```

Figura 39. Ejecución del comando “*git diff --cached*”, mostrando los cambios realizados en el archivo *WeakAuras/BossMods.lua*.

Tras agregar los cambios al índice del proyecto git y ejecutar la funcionalidad de generación de mensajes de commit se obtuvieron los siguientes mensajes como respuesta.

<sup>28</sup> <https://github.com/WeakAuras/WeakAuras2>

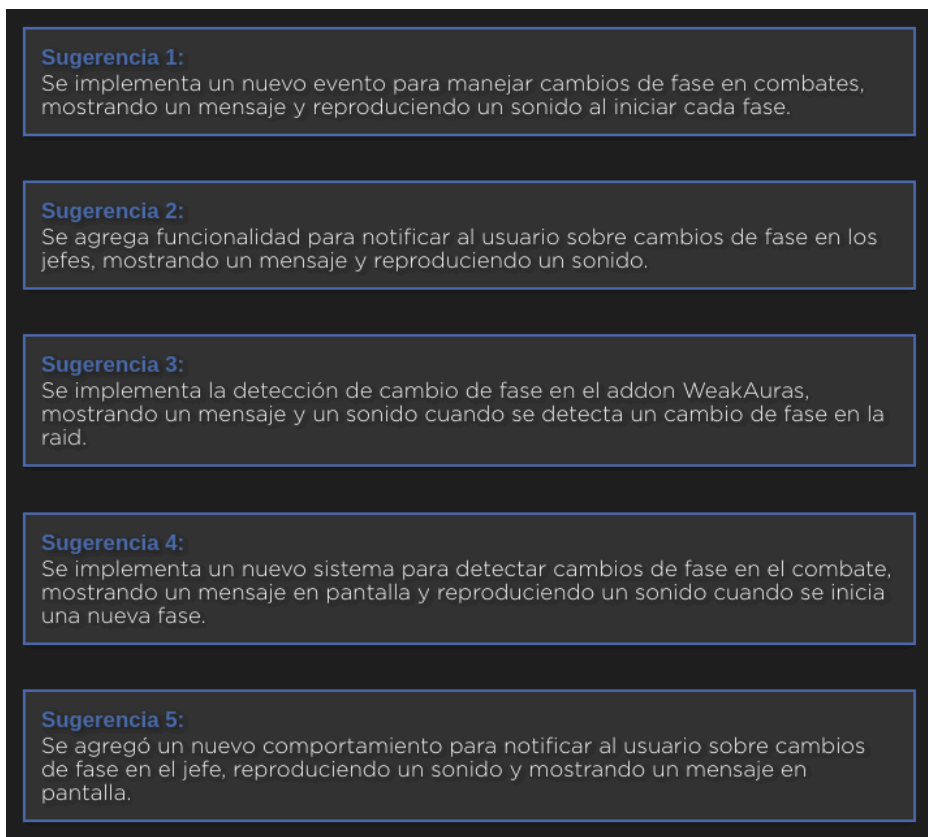


Figura 40. Listado de sugerencias devueltas por la extensión durante la experimentación.

El resultado de esta última experimentación resulta sumamente interesante, ya que no sólo encontramos mensajes que hacen referencia a funcionalidades agregadas que pueden inferirse fácilmente al leer el código añadido (reproducción de sonido, eventos de cambio de fase, etc) sino que, además, la extensión reconoce un contexto que va más allá de las líneas añadidas y son propias del dominio particular sobre el cual se está desarrollando, sabiendo interpretar y contextualizar términos peculiares del mismo como son “*addon*”, “*jefe*” y “*raid*”. De esta manera, podemos observar que el análisis hecho por la extensión no se limita a interpretar el accionar individual de las sentencias incluidas, sino que además analiza su contexto y las relaciona.

## Mejoras Posibles

Durante el desarrollo de la extensión, nos encontramos con una serie de aspectos y/o limitantes que podrían ser mejorados o solucionados en una futura implementación. Los mismos los detallamos a continuación.

### Solucionar la Limitación de Tokens

Tal como se mencionó previamente en este mismo documento, Gemini plantea una limitación en cuanto a la cantidad de tokens que permite procesar por petición en base al plan de pago al

que está suscripto el usuario que hace uso de su API. El contar con un plan gratuito, como en nuestro caso, reduce la longitud máxima de las cadenas que se puede enviar a la API antes de que la misma rechace la petición. Es por esto que a fines prácticos y demostrativos, limitamos el número de commits con los que trabaja el catálogo para evitar bloqueos de este tipo y llegar a nuestro objetivo de demostrar la utilidad de la herramienta. Así mismo, un hipotético cambio de gran dimensión agregado al índice del proyecto git, por medio de un *git add*, podría causar una respuesta negativa similar a la hora de enviarlo dentro del llamado a la API al querer generar las sugerencias de mensaje de commit.

Dicho lo anterior, una posible mejora para la herramienta es la de contratar un plan que permita cubrir strings de gran longitud sin que el máximo de Tokens sea un limitante. Al momento de redactar este documento, para el caso del modelo que utilizamos en nuestro desarrollo (Gemini 1.5 Flash), existe un plan de pago por uso<sup>29</sup>, detallado a continuación.

Tabla 3. Plan Gratuito vs Plan Pago

Nuestro Plan Actual (Gratuito)	Pago por uso
Límites de Cuota: 15 RPM (peticiones por minuto) 1 millón TPM (tokens por minuto) 1,500 RPD (peticiones por minuto)	Límites de Cuota: 1000 RPM (peticiones por minuto) 4 millones TPM (tokens por minuto)
Precio (ENTRADA) Gratis	Precio (ENTRADA) \$0.075 / 1 millón tokens (para prompts hasta 128K tokens) \$0.15 / 1 millón tokens (para prompts mayores a 128K)
Precio (SALIDA) Gratis	Precio (SALIDA) \$0.30 / 1 millón tokens ((para prompts hasta 128K tokens) \$0.60 / 1 millón tokens (para prompts mayores a 128K)

Otra posible manera de solucionar esta problemática es la de incursionar en técnicas que fragmentan las peticiones en partes de menor longitud de tokens. OpenAI, por ejemplo, proporciona una API<sup>30</sup> que permite manejar peticiones en bloques, recibiendo un listado de strings a procesar y, adicionalmente, cuentan con documentación<sup>31</sup> de otras técnicas de

<sup>29</sup> [https://aistudio.google.com/app/plan\\_information](https://aistudio.google.com/app/plan_information)

<sup>30</sup> <https://platform.openai.com/docs/guides/batch/overview>

<sup>31</sup> <https://platform.openai.com/docs/guides/rate-limits/error-mitigation>

mitigación de errores de límite (Sanders, 2022). De esta manera es posible enviar entradas al modelo que no superen el límite de Tokens dado por la cuota, a expensas del tiempo de respuesta inmediato.

## Estandarización de mensajes

Una de las posibles mejoras que podrían implementarse en una versión futura dentro de la extensión es la de modificar la directiva enviada a la API estableciendo el formato en el cual queremos recibir nuestros mensajes de commit sugeridos, con el objetivo de seguir estándares, convenciones o buenas prácticas que favorezcan la gestión de versiones.

Una posible opción es la de utilizar la convención Conventional Commits<sup>32</sup>, en la que cada mensaje de commit tiene el siguiente formato:

```
<tipo>[scope optional]: <descripción>  
[cuerpo opcional]  
[footer opcional]
```

Con esto, lo que se buscaría es comunicar de manera más efectiva la naturaleza del cambio y otra información adicional de interés para el desarrollador o el equipo de desarrollo.

A continuación mostramos un ejemplo de un mensaje de commit expresado siguiendo la especificación de Conventional Commits:

```
fix(api): corrección de vulnerabilidad  
Se corrige una vulnerabilidad de seguridad que permitía inyección de código tras las  
funciones introducidas en el commit de referencia.  
Reviewed-by: edemarco  
Refs: #f883bef05457a5a49eb31109429fc01737f82532
```

La especificación de Conventional Commits es amplia, pero no es la única existente o utilizada. Existen múltiples convenciones sobre las que la extensión se podría apoyar y la preferencia sobre una u otra puede depender más del usuario o el proyecto. La siguiente posible mejora que planteamos está relacionada con esto.

## Personalización del Prompt

Una misma API puede responder de maneras muy diferentes dependiendo del prompt enviado en la solicitud. Una funcionalidad implementable para versiones futuras de la extensión es la de contar con una selección de múltiples prompts elegibles por el usuario previo al envío de la solicitud de generación de mensajes. Las mismas podrían describir la problemática desde diferentes perspectivas e incluso exigir respuestas siguiendo diferentes

---

<sup>32</sup> <https://www.conventionalcommits.org/en/v1.0.0/#specification>

formatos, estándares o convenciones. El usuario, de esta manera, podría elegir aquella personalización que más le convenga para sus necesidades.

### Personalización de otros parámetros

Características de otro tipo podrían ser personalizables en una futura implementación. La cantidad de sugerencias generadas e incluso el modelo a utilizar para la generación de las mismas son parámetros que podrían ser seleccionables por el usuario, previo a ejecutar la funcionalidad de la extensión.

### Refinamiento de prompts

A pesar de que los prompts y las directivas utilizadas en nuestro desarrollo cumplieron de manera satisfactoria las necesidades planteadas para el mismo, es posible refinarlos mediante un proceso exhaustivo de pruebas o brindando un contexto mucho más detallado y dirigido al tipo de lenguaje o aplicaciones para el cual será utilizado.

### Implementación de un modelo local

Como alternativa al uso de modelos en línea a través de la comunicación con una API, la implementación de un modelo local resolvería el limitante de tokens. Esta alternativa, sin embargo, implica la necesidad de recursos de hardware lo suficientemente potentes para poder albergar un modelo que realice las mismas funcionalidades de manera exitosa.

### Edición de Sugerencias

Una funcionalidad de gran utilidad posible a agregar es la posibilidad de editar el texto de las sugerencias generadas por la IA, previo a realizar el commit.

### Pre-cargado de código para evaluar contexto

Con el objetivo de que la extensión tenga un conocimiento más detallado del dominio, puede evaluarse realizar una precarga del código a fin de brindar a la aplicación de un mayor contexto previo a cualquier análisis. Esto, sin embargo, implicaría un gran costo de procesamiento y sería más recomendable en los casos en que se implemente un modelo local.

## Conclusiones

Tras la finalización de los procesos de recopilación y análisis de información que se realizaron a lo largo de este documento podemos afirmar que obtuvimos una amplia perspectiva del marco actual en lo que respecta a las nuevas tecnologías de Inteligencia Artificial y Análisis de Lenguaje Natural en el mercado.

Asimismo, el desarrollo y las experimentaciones realizadas nos permitieron evaluar de primera mano la implementación de las técnicas objeto de estudio del presente trabajo de tesina en un dominio familiar para nosotros así como para incontables desarrolladores a lo largo del planeta.

Finalizada la redacción de este documento, es posible aseverar que cumplimos nuestros objetivos específicos al haber: investigado, estudiado y documentado el marco teórico dentro del presente informe; realizado un análisis del mercado actual describiendo las herramientas más importantes presentes para el público general; desarrollado una herramienta integrada al IDE Visual Studio Code que permitiera facilitar tanto la generación de mensajes de commit, como la búsqueda de cambios versionados durante la etapa de desarrollo de software; probado dicha herramienta documentando distintos casos de experimentación y sus resultados exitosos.

Por otro lado, podemos concluir que cumplimos con el objetivo general de nuestra tesina visto que los resultados consecuentes a las pruebas así como la revisión del marco teórico y estado del arte nos permiten concluir en la veracidad de aquella premisa que buscábamos exponer: la existencia de un vasto potencial en la IA y el Análisis del Lenguaje Natural como herramientas de soporte en el proceso de Desarrollo de Software. Premisa que, a día de hoy, se renueva y vuelve cada vez más vigente a la par de los incesantes avances que salen a la luz día a día.

# Bibliografía

- Anil, R., Dai, A. M., Firat, O., Johnson, M., Lepikhin, D., Passos, A., Shakeri, S., Taropa, E., Bailey, P., Chen, Z., Chu, E., Clark, J. H., Shafey, L. E., Huang, Y., Meier-Hellstern, K., Mishra, G., Moreira, E., Omernick, M., Robinson, K., . . . Wu, Y. (2023). PALM 2 Technical Report. arXiv (Cornell University). <https://doi.org/10.48550/arxiv.2305.10403>
- API Plan Billing Information (s.f.). [https://aistudio.google.com/app/plan\\_information](https://aistudio.google.com/app/plan_information)
- Arcas, B. a. Y. (2022). Do large language models understand us? *Daedalus*, 151(2), 183–197. [https://doi.org/10.1162/daed\\_a\\_01909](https://doi.org/10.1162/daed_a_01909)
- Atresmedia (2021, Junio 21). ¿Por qué la inteligencia artificial no habla y escucha en español? [https://compromiso.atresmedia.com/levanta-la-cabeza/actualidad/que-inteligencia-artificial-habla-escucha-espanol\\_2021062160d0404ca32c2c0001253cf3.html](https://compromiso.atresmedia.com/levanta-la-cabeza/actualidad/que-inteligencia-artificial-habla-escucha-espanol_2021062160d0404ca32c2c0001253cf3.html)
- Bommasani, R., Hudson, D. A., Adeli, E., Altman, R., Arora, S., Von Arx, S., Bernstein, M. S., Bohg, J., Bosselut, A., Brunskill, E., Brynjolfsson, E., Buch, S., Card, D., Castellon, R., Chatterji, N., Chen, A., Creel, K., Davis, J. Q., Demszky, D., . . . Liang, P. (2021). On the Opportunities and Risks of Foundation Models. arXiv (Cornell University). <https://doi.org/10.48550/arxiv.2108.07258>
- Carretero, L. M. (2021, Mayo 23). Lecciones urgentes de español para la inteligencia artificial. *Diario ABC*. [https://www.abc.es/economia/abci-lecciones-urgentes-espanol-para-inteligencia-artificial-202105240129\\_noticia.html?ref=https://www.abc.es/economia/abci-lecciones-urgentes-espanol-para-inteligencia-artificial-202105240129\\_noticia.html](https://www.abc.es/economia/abci-lecciones-urgentes-espanol-para-inteligencia-artificial-202105240129_noticia.html?ref=https://www.abc.es/economia/abci-lecciones-urgentes-espanol-para-inteligencia-artificial-202105240129_noticia.html)
- Chernyavskiy, A., Ilvovsky, D., & Nakov, P. (2021). Transformers: “The end of history” for natural language processing? In *Machine Learning and Knowledge Discovery in Databases: Research Track: European Conference, ECML PKDD 2021, Bilbao, Spain, September 13–17, 2021, Proceedings, Part III* (pp. 677–693). Springer.
- Chomsky, N. (1957). *Syntactic structures*. Mouton.
- Chomsky, N. (1965). *Aspects of the theory of syntax*.
- Conventional Commits. (s.f.). Recuperado el 1 Oct. 2024 de <https://www.conventionalcommits.org/en/v1.0.0/#specification>
- Copetake, A. (2004). *Natural Language Processing*.
- Databricks. (2023). MPT-30B: Raising the bar for open-source foundation models. *Databricks Blog*. <https://www.databricks.com/blog/mpt-30b-raising-the-bar-for-open-source-foundation-models>
- DeepLearning.AI. (2023, Enero 11). *Natural Language Processing (NLP) [A complete guide]*. <https://www.deeplearning.ai/resources/natural-language-processing/>
- Dreyfus, H. (1994). *What computers still can't do*. Computer Science Department, Stanford University, Stanford, CA 94305, USA.
- Eisenstein, J. (2018). *Natural Language Processing*.
- Ekin, S. (2023, Abril 29). Prompt engineering for ChatGPT: A quick guide to techniques, tips, and best practices. TechRxiv. <https://doi.org/10.36227/techrxiv.22364259.v1>
- Falcon LLM. (s.f.). <https://falconllm.tii.ae/>

- Friedman, N. (2021, Junio 29). Introducing GitHub Copilot: your AI pair Programmer. The GitHub Blog. <https://github.blog/news-insights/product-news/introducing-github-copilot-ai-pair-programmer/>
- Gemini Team, Anil, R., Borgeaud, S., Alayrac, J., Yu, J., Soricut, R., Schalkwyk, J., Dai, A. M., Hauth, A., Millican, K., Silver, D., Johnson, M., Antonoglou, I., Schrittwieser, J., Glaese, A., Chen, J., Pitler, E., Lillicrap, T., Lazaridou, A., . . . Vinyals, O. (2023, Diciembre 19). Gemini: a family of highly capable multimodal models. arXiv.org. <https://arxiv.org/abs/2312.11805>
- Gemini Team, Georgiev, P., Lei, V. I., Burnell, R., Bai, L., Gulati, A., Tanzer, G., Vincent, D., Pan, Z., Wang, S., Mariooryad, S., Ding, Y., Geng, X., Alcober, F., Frostig, R., Omernick, M., Walker, L., Paduraru, C., Sorokin, C., . . . Vinyals, O. (2024, Marzo 8). Gemini 1.5: Unlocking multimodal understanding across millions of tokens of context. arXiv.org. <https://arxiv.org/abs/2403.05530>
- GitHub REST API documentation - GitHub Docs. (2022, November 28). GitHub Docs. <https://docs.github.com/en/rest?apiVersion=2022-11-28>
- González-Blanco, E. (2021, Abril 22). Por qué las máquinas no hablan bien español y por qué deberían hacerlo. The Conversation. <https://theconversation.com/por-que-las-maquinas-no-hablan-bien-espanol-y-por-que-deberian-hacerlo-159547>
- Google AI. (s.f.). Gemini API. <https://ai.google.dev/gemini-api/>
- Hosseini, K., Nanni, F., & Coll Ardanuy, M. (2020). DeezyMatch: A flexible deep learning approach to fuzzy string matching. Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing: System Demonstrations, 62–69. Association for Computational Linguistics. <https://doi.org/10.18653/v1/2020.emnlp-demos.9>
- ILIA. (2024). Índice Latinoamericano de Inteligencia Artificial. En Indicelatam (ILIA\_2024). [https://indicelatam.cl/wp-content/uploads/2024/10/ILIA\\_2024.pdf](https://indicelatam.cl/wp-content/uploads/2024/10/ILIA_2024.pdf)
- Introducing Claude 3.5 Sonnet. (2024 Junio 21). <https://www.anthropic.com/news/claude-3-5-sonnet>
- Jiang, A. Q., Sablayrolles, A., Mensch, A., Bamford, C., Chaplot, D. S., De Las Casas, D., Bressand, F., Lengyel, G., Lample, G., Saulnier, L., Lavaud, L. R., Lachaux, M., Stock, P., Scao, T. L., Lavril, T., Wang, T., Lacroix, T., & Sayed, W. E. (2023, Octubre 10). Mistral 7B. arXiv.org. <https://arxiv.org/abs/2310.06825>
- Kulshrestha, R. (2019, Noviembre 24). NLP 101: Word2VEC — Skip-Gram and CBOW - towards Data Science. Medium. <https://towardsdatascience.com/nlp-101-word2vec-skip-gram-and-cbow-93512ee24314>
- La Fontaine, G. (2024). Sobre loros estocásticos. Una mirada a los modelos grandes de lenguaje. Lógoi Revista De Filosofía, 45, 75–87. <https://doi.org/10.62876/lr.vi45.6480>
- Li, R., Allal, L. B., Zi, Y., Muennighoff, N., Kocetkov, D., Mou, C., Marone, M., Akiki, C., Li, J., Chim, J., Liu, Q., Zheltonozhskii, E., Zhuo, T. Y., Wang, T., Dehaene, O., Davaadorj, M., Lamy-Poirier, J., Monteiro, J., Shliazhko, O., . . . Harm, D. V. (2023, Mayo 9). StarCoder: may the source be with you! arXiv.org. <https://arxiv.org/abs/2305.06161>
- López, J. Á. P. (2019, Octubre 17). El español sigue siendo una lengua extranjera para la inteligencia artificial. El País. [https://elpais.com/retina/2019/10/16/talento/1571218870\\_674350.html](https://elpais.com/retina/2019/10/16/talento/1571218870_674350.html)

- Malartic, Q., Chowdhury, N. R., Cojocaru, R., Farooq, M., Campesan, G., Djilali, Y. a. D., Narayan, S., Singh, A., Velikanov, M., Boussaha, B. E. A., Al-Yafeai, M., Alobeidli, H., Qadi, L. A., Seddik, M. E. A., Fedyanin, K., Alami, R., & Hacid, H. (2024, Julio 20). Falcon2-11B Technical Report. arXiv.org. <https://arxiv.org/abs/2407.14885>
- Marcus, G. (2018). Deep Learning: a critical appraisal. arXiv (Cornell University). <https://doi.org/10.48550/arxiv.1801.00631>
- Meta. (2024, Julio 23). Introducing Llama 3.1: Our most capable models to date. ai.meta.com. Recuperado Oct. 1, 2024, de <https://ai.meta.com/blog/meta-llama-3-1/>
- Mikolov, T., Chen, K., Corrado, G., & Dean, J. (2013). Efficient estimation of word representations in vector space [Preprint]. arXiv. <https://doi.org/10.48550/arXiv.1301.3781>
- Miller, G. A. (1951). Language and communication.
- Minaee, S., Mikolov, T., Nikzad, N., Chenaghlu, M., Socher, R., Amatriain, X., & Gao, J. (2024). Large Language Models: a survey. arXiv (Cornell University). <https://doi.org/10.48550/arxiv.2402.06196>.
- Mistral. (s.f.). <https://mistral.ai/technology/>
- Montague, R. (1970). Universal grammar. In Grammar and meaning.
- MPT-30B: Raising the bar for open-source foundation models | Databricks Blog. (2023 Junio 22). Databricks. <https://www.databricks.com/blog/mpt-30b>
- Naveed, H., Khan, A. U., Qiu, S., Saqib, M., Anwar, S., Usman, M., Akhtar, N., Barnes, N., & Mian, A. (2024). A comprehensive overview of large language models [Preprint]. Disponible en <https://www.studocu.com/in/document/savitribai-phule-pune-university/computer-engineering/large-language-models/91396127>
- OpenAI. (2021, Agosto 10). OpenAI Codex. Openai. <https://openai.com/index/openai-codex/>
- OpenAI. (s.f.). Batch overview. Recuperado el 1 Oct. 2024 de <https://platform.openai.com/docs/guides/batch/overview>
- OpenAI. (s.f.). Rate limits: Error mitigation. Recuperado el 1 Oct. 2024 de <https://platform.openai.com/docs/guides/rate-limits/error-mitigation>
- Palanivayagam, A., El-Bayeh, C. Z., & Damaševičius, R. (2023). Twenty Years of Machine-Learning-Based Text Classification: A Systematic Review. *Algorithms*, 16(5), 236. <https://doi.org/10.3390/a16050236>
- Pennington, J., Socher, R., & Manning, C. D. (2014). GloVe: Global vectors for word representation. In Proceedings of the 2014 conference on empirical methods in natural language processing (EMNLP) (pp. 1532–1543). Association for Computational Linguistics. <https://doi.org/10.3115/v1/D14-1162>
- Russell, S., & Norvig, P. (2016). Artificial Intelligence: A Modern Approach.
- Sanders, T. (2022, Septiembre 9). How to handle rate limits. OpenAI Cookbook. [https://cookbook.openai.com/examples/how\\_to\\_handle\\_rate\\_limits](https://cookbook.openai.com/examples/how_to_handle_rate_limits)
- ServiceNow Research. (2022, Septiembre 26). Announcing BigCode for the responsible development of large language models. Servicenow. <https://www.servicenow.com/blogs/2022/bigcode-large-language-models>
- ServiceNow Research. (2023, Mayo 4). ServiceNow and Hugging Face release StarCoder, one of the world’s most responsibly developed and strongest-performing open-access large language model for code generation [Comunicado de prensa]. servicenow.

- <https://www.servicenow.com/company/media/press-room/starcoder.html>
- Smith, A. (2022, November 20). Kite is saying farewell. Code Faster With Kite. <https://www.kite.com/blog/product/kite-is-saying-farewell/>
- Smith, J., & Johnson, L. (2020). Environmental impact of large language models. *Journal of AI Research*.
- Smith, J., & Doe, A. (2021). *Transformers: The fundamental models of AI*. Stanford University.
- Sutskever, I., Vinyals, O., & Le, Q. V. (2014). Sequence to sequence learning with neural networks. *arXiv [Preprint] arXiv:1409.3215*. <https://doi.org/10.5555/2969033.2969173>
- Traditional NLP - Eng. - 06. (s.f.). In *Deep Learning Bible [Recap]*. Natural Language Processing (NLP) with Deep Learning. Recuperado el 1 Oct. 2024 de <https://wikidocs.net/202995>
- Turing, A. M. (1950). I.—COMPUTING MACHINERY AND INTELLIGENCE. *Mind*, LIX(236), 433–460. <https://doi.org/10.1093/mind/lix.236.433>
- Yudkowsky, E. (2023, April 21). Pausing AI Developments Isn't Enough. We Need to Shut it All Down - Machine Intelligence Research Institute. Machine Intelligence Research Institute. <https://intelligence.org/2023/04/07/pausing-ai-developments-isnt-enough-we-need-to-shut-it-all-down/>
- Zhang, C., Bengio, S., Hardt, M., Recht, B., & Vinyals, O. (2021). Understanding deep learning (still) requires rethinking generalization. *Communications of the ACM*, 64(3), 107-115. <https://doi.org/10.1145/34446776>

# Glosario

- **Morfología:** La estructura de las palabras. Por ejemplo, "inusualmente" puede considerarse compuesto por un prefijo "in-", un lexema "usual" y un sufijo "-mente".
- **Sintaxis:** La forma en que las palabras se utilizan para formar frases. Por ejemplo, es parte de la sintaxis del inglés que un determinante como "the" preceda a un sustantivo, y también que los determinantes sean obligatorios con ciertos sustantivos singulares.
- **Semántica:** La construcción del significado (generalmente expresado en lógica) basada en la sintaxis.
- **Pragmática:** el significado en contexto.
- **Pronombre:** Un pronombre es una palabra que se utiliza en lugar de un sustantivo específico para evitar la repetición del mismo. Los pronombres pueden referirse a personas, objetos, lugares, ideas, etc. Algunos ejemplos comunes de pronombres incluyen "él", "ella", "ellos", "ellas", "eso", "esto", "aquello", "ella misma", "esto", entre otros.
- **Anáfora:** Una anáfora es un fenómeno lingüístico en el cual una palabra o frase hace referencia a otra palabra o frase mencionada anteriormente en el discurso. La palabra o frase a la que se hace referencia se conoce como el antecedente. La anáfora ayuda a mantener la coherencia y la claridad en el discurso al vincular elementos dentro de una oración o un texto más extenso. Los pronombres son un tipo común de anáfora, ya que hacen referencia a un sustantivo o a un concepto mencionado previamente.
- **Inteligencia artificial centrada en datos:** Es un enfoque que se centra en la calidad de los datos y en la importancia del preprocesamiento de datos para mejorar el rendimiento del modelo. Al utilizar técnicas de preprocesamiento de datos, se pueden mejorar los resultados del modelo y hacer que sea más preciso y eficiente.
- **Corpus:** Es el conjunto total de documentos.
- **Documento:** Un documento es un texto o conjunto de palabras.
- **Repositorio:** Un repositorio o proyecto de Git comprende toda la colección de archivos y carpetas asociados con un proyecto, en conjunto con el historial de revisión de cada archivo.