UNIVERSIDAD POLITÉCNICA DE MADRID

ESCUELA TÉCNICA SUPERIOR
DE INGENIEROS DE TELECOMUNICACIÓN


GRADO EN INGENIERÍA DE TECNOLOGÍAS Y SERVICIOS DE TELECOMUNICACIÓN

TRABAJO FIN DE GRADO

# IMPLEMENTACIÓN DE UN SISTEMA DE EVALUACIÓN DE CAPACIDADES DE GRANDES MODELOS DE LENGUAJE A TRAVÉS DE BUCLES DE RESUMEN Y EXTENSIÓN DE TEXTOS

JAVIER GONZÁLEZ PÉREZ

2025

## GRADO EN TECNOLOGÍAS Y SERVICIOS DE TELECOMUNICACIÓN

## TRABAJO FIN DE GRADO

**Título:** Implementación de un sistema de evaluación de capacidades de Grandes Modelos de Lenguaje a través de bucles de resumen y extensión de textos.

**Autor:** D. Javier González Pérez

**Tutor:** D. Javier Conde Díaz

**Supervisión académica:** D. ……

**Departamento:** ……

## MIEMBROS DEL TRIBUNAL

**Presidente:** D. ……

**Vocal:** D. ……..

**Secretario:** D. ……..

**Suplente:** D. ……..

Los miembros del tribunal arriba nombrados acuerdan otorgar la calificación de: ……

Madrid, a de de 20…

# UNIVERSIDAD POLITÉCNICA DE MADRID

## ESCUELA TÉCNICA SUPERIOR
## DE INGENIEROS DE TELECOMUNICACIÓN



## GRADO EN INGENIERÍA DE TECNOLOGÍAS Y SERVICIOS DE TELECOMUNICACIÓN

## TRABAJO FIN DE GRADO

# IMPLEMENTACIÓN DE UN SISTEMA DE EVALUACIÓN DE CAPACIDADES DE GRANDES MODELOS DE LENGUAJE A TRAVÉS DE BUCLES DE RESUMEN Y EXTENSIÓN DE TEXTOS


## JAVIER GONZÁLEZ PÉREZ


## 2025

# RESUMEN

Este Trabajo de Fin de Grado propone un sistema experimental para evaluar la estabilidad, fidelidad semántica y capacidad de retención de los Grandes Modelos de Lenguaje (LLMs) mediante ciclos iterativos de resumen y expansión textual. Frente a los métodos tradicionales basados en benchmarks estáticos, se presenta un enfoque más dinámico y novedoso que permite observar cómo se transforma un texto al ser reformulado repetidamente por un modelo. El sistema implementado permite evaluar si, tras varias iteraciones, el modelo conserva la información original o si introduce degradaciones, invenciones o pérdidas semánticas.

Para ello, se desarrolló una arquitectura modular en Python que interactúa con modelos accesibles vía API, como GPT-4 Mini (OpenAI) y Gemini (Google). El sistema toma como entrada textos del dataset CNN/DailyMail, ampliamente utilizado en tareas de resumen, y aplica un bucle de resumen y expansión durante diez iteraciones. En cada iteración, el modelo genera un resumen del texto anterior y lo expande de nuevo a una longitud similar al original, permitiendo así evaluar la evolución del contenido de forma acumulativa.

La evaluación de los resultados se realizó mediante dos enfoques complementarios. Por un lado, se utilizaron métricas automáticas de similitud como ROUGE y BERTScore, que permiten cuantificar la pérdida de información y la coherencia semántica entre los textos generados y los originales. Por otro lado, se diseñó un sistema de evaluación basado en preguntas tipo test generadas automáticamente a partir del texto original. Estas preguntas se aplicaron posteriormente a los textos generados en diferentes iteraciones, evaluando así la retención semántica del contenido a través de la capacidad del modelo para responder correctamente.

Los resultados obtenidos muestran una degradación significativa de la información en las primeras iteraciones, que tiende a estabilizarse en ciclos posteriores. Las métricas ROUGE y BERTScore reflejan una caída inicial pronunciada en precisión, seguida de una meseta que indica un nuevo equilibrio semántico. Esta tendencia se confirma con los resultados de las pruebas tipo test, donde se observa una disminución progresiva en la tasa de respuestas correctas a medida que el texto se aleja del original.

Este trabajo aporta una metodología reproducible y extensible para analizar el comportamiento de los LLMs en escenarios iterativos de generación, complementando los enfoques tradicionales con herramientas de evaluación más centradas en la semántica y la comprensión lectora. La estructura modular del sistema, así como los recursos y scripts disponibles públicamente en GitHub y Zenodo, permiten su reutilización y ampliación futura en distintos contextos de investigación o desarrollo industrial.

# SUMMARY

This Bachelor's Thesis presents an experimental system designed to evaluate the stability, semantic fidelity, and retention capacity of Large Language Models (LLMs) through iterative cycles of summarization and text expansion. In contrast to traditional evaluation methods based on static benchmarks, this work proposes a more dynamic and innovative approach that examines how a text is transformed as it is repeatedly reformulated by a model. The system makes it possible to assess whether the model maintains the original information or introduces semantic degradation, hallucinations, or information loss over time.

To this end, a modular Python-based architecture was developed that interacts with API-accessible models such as GPT-4 Mini (OpenAI) and Gemini (Google). The system uses input texts from the CNN/DailyMail dataset widely used in summarization tasks and applies a loop of summarization and expansion over ten iterations. In each iteration, the model generates a summary of the previous text and then expands it back to the original length, allowing for a cumulative analysis of the content's evolution.

Two complementary evaluation approaches were used to assess the quality of the results. On the one hand, automatic similarity metrics such as ROUGE and BERTScore were employed to quantify information loss and semantic coherence between the generated and reference texts. On the other hand, a semantic retention evaluation system was implemented based on multiple-choice questions automatically generated from the original text. These questions were then applied to each generated version to evaluate the model's ability to retain and comprehend the information.

The results show a significant degradation of information in the early iterations, which tends to stabilize in later cycles. ROUGE and BERTScore metrics reflect an initial sharp decline in precision and F1-score, followed by a plateau that suggests a new semantic equilibrium. This trend is confirmed by the test-based evaluation, where a progressive decrease is observed in the number of correct answers as the text moves further away from the original.

This work provides a reproducible and extensible methodology to analyze LLM behavior in iterative text generation scenarios, complementing traditional evaluation methods with tools focused on semantic understanding and reading comprehension. The modular structure of the system and the availability of scripts and resources on GitHub and Zenodo enable future reuse and extension in research or industrial contexts.

## PALABRAS CLAVE

Modelos de Lenguaje, LLMs, Resumen automático, Expansión de texto, Evaluación semántica, Métricas ROUGE, BERTScore, Comprensión lectora, Generación iterativa, GPT-4 Mini, Gemini, Python, API, Evaluación de modelos.

## KEYWORDS

Large Language Models, LLMs, Summarization, Text Expansion, Semantic Evaluation, ROUGE, BERTScore, Reading Comprehension, Iterative Generation, GPT-4 Mini, Gemini, Python, API, Model Evaluation.

# ÍNDICE

# 1. INTRODUCCIÓN Y OBJETIVOS

## 1.1. INTRODUCCIÓN

Los modelos de lenguaje de gran tamaño (Large Language Models, LLMs) han adquirido un papel central en el desarrollo de sistemas inteligentes capaces de comprender y generar lenguaje natural. Su aplicación se ha extendido rápidamente a numerosos dominios, gracias a su capacidad para adaptarse a tareas tan diversas como la traducción automática, la generación de código, la asistencia conversacional o la redacción de textos complejos.

A pesar de su creciente presencia y rendimiento medido en benchmarks estáticos, persiste una necesidad de métodos de evaluación más realistas que permitan analizar cómo se comportan estos modelos en contextos dinámicos o iterativos. En particular, no está suficientemente explorado qué ocurre cuando un modelo reformula su propia salida de forma recursiva, por ejemplo, mediante procesos consecutivos de resumen y expansión.

Con el fin de facilitar la comprensión del proyecto y fomentar la reproducibilidad de los experimentos realizados, se recomienda acompañar la lectura de esta memoria con los recursos complementarios disponibles en GitHub y Zenodo. En el repsitorio de GitHub [1] se encuentran todos los scripts de Python desarrollados para implementar, ejecutar y evaluar el sistema experimental. Por su parte, el depósito de Zenodo [2] recopila los archivos de salida generados a partir de esos scripts, incluyendo los resultados en formato Excel y JSONL, gráficas de evaluación y respuestas obtenidas por los modelos.

Este Trabajo de Fin de Grado propone un enfoque experimental para estudiar la estabilidad y fidelidad de los modelos LLM en ciclos de generación iterativa. El sistema desarrollado permite observar cómo evoluciona un texto al ser resumido y luego expandido varias veces, utilizando modelos accesibles vía API como GPT-4 Mini y Gemini. Para evaluar la calidad de los textos generados, se combinan métricas automáticas (ROUGE, BERTScore) con un sistema de test de comprensión basado en preguntas tipo test generadas a partir del texto original.

El objetivo principal es analizar si los modelos son capaces de conservar la información clave, la coherencia semántica y la estructura original del contenido a lo largo de varias iteraciones de generación.

El presente documento se organiza en seis capítulos principales. Tras esta introducción, se presenta el marco teórico, seguido del desarrollo del sistema experimental. A continuación, se describen los resultados obtenidos y se exponen las conclusiones y líneas futuras del trabajo. Por último se recoge la bibliografía utilizada. Posteriormente, se incluyen dos anexos: el Anexo A, dedicado al análisis de impactos éticos, sociales, económicos y medioambientales del proyecto, y el Anexo B, que presenta una estimación del presupuesto económico.

## 1.2. OBJETIVOS

Aunque los Grandes Modelos de Lenguaje (LLMs) han alcanzado niveles de rendimiento extraordinarios en una amplia variedad de tareas, la evaluación rigurosa de su comportamiento real y su fiabilidad sigue representando un desafío importante. Habitualmente, esta evaluación se basa en pruebas estáticas o benchmarks, donde el modelo genera una única respuesta ante una entrada concreta. Sin embargo, este enfoque no permite analizar si el modelo es capaz de conservar la información original cuando reformula un texto de forma iterativa ni en qué medida introduce invenciones al expandirlo.

Este Trabajo de Fin de Grado plantea un enfoque alternativo: estudiar el comportamiento de los LLMs en **bucles de generación iterativa**, combinando procesos de resumen y extensión textual. El **objetivo** es estudiar cómo se transforma el contenido a lo largo de varias generaciones, y estudiar si el modelo es capaz de conservar la coherencia, la fidelidad semántica y la estructura del texto original, o si por el contrario introduce ruido, redundancia o pérdida de información progresiva.

Para ello, se desarrollará un sistema experimental que permita cargar y evaluar distintos modelos de lenguaje ofrecidos por proveedores como Google o OpenAI, accediendo a ellos mediante sus APIs. Las pruebas se aplicarán sobre el dataset **CNN/DailyMail** [3], ampliamente utilizado en tareas de resumen automático, y se automatizarán mediante scripts desarrollados en Python.

Los resultados obtenidos serán evaluados a través de métricas de similitud como **ROUGE**, **BERTScore**, que permitirán cuantificar la calidad, la consistencia y la degradación de la información generada en cada iteración. Adicionalmente, se implementará una evaluación automatizada de la retención semántica mediante tests de comprensión generados por los propios modelos. Estos tests tipo "pregunta-respuesta" permitirán verificar si la información clave del texto original sigue presente en las versiones resumidas o extendidas, proporcionando una medida complementaria de comprensión y fidelidad semántica más allá de las métricas puramente comparativas.

En conjunto, este trabajo pretende aportar una **contribución práctica y original** al análisis del comportamiento de los LLMs, proponiendo un marco experimental reproducible que complemente las evaluaciones clásicas y ofrezca una visión más dinámica y realista de su desempeño en tareas complejas de generación de lenguaje.

# 2. MARCO TEÓRICO

## 2.1 FUNDAMENTOS DE LOS GRANDES MODELOS DE LENGUAJE

En los últimos años, los **Grandes Modelos de Lenguaje** (*Large Language Models*, LLMs) [4, 5, 6] han revolucionado el campo de la inteligencia artificial gracias a su capacidad para generar, comprender y manipular texto de manera sorprendentemente coherente y contextualizada. Estos modelos, basados en arquitecturas neuronales profundas, han demostrado resultados notables en tareas tan diversas como la traducción automática, la generación de código, la respuesta a preguntas, la redacción de resúmenes o incluso el razonamiento lógico [5, 6].

Un **LLM** puede entenderse de forma simplificada como un sistema que, dado un texto de entrada, predice el siguiente fragmento más probable. Esta tarea, que en apariencia puede parecer simple, obliga al modelo a capturar una gran cantidad de conocimiento semántico, sintáctico y factual sobre el lenguaje y el mundo. En esencia, el modelo ha aprendido a escribir a base de leer Internet [5].

Para llevar a cabo esta predicción, el modelo no trabaja directamente con palabras completas, sino con unidades mínimas de texto denominadas **tokens**. Un token puede ser una palabra entera, una sílaba o incluso una fracción de palabra. Esta fragmentación se realiza mediante un componente llamado *tokenizador*, que transforma el texto en una secuencia de unidades manejables por el modelo. Por ejemplo, la frase “Estoy estudiando inteligencia artificial.” puede tokenizarse como ["Estoy", " estudi", "ando", " inteligencia", " artificial", "."], dependiendo del tipo de tokenizador empleado. El hecho de dividir palabras en subcomponentes permite trabajar con un vocabulario reducido de decenas de miles de tokens capaz de generar combinaciones para reconstruir cualquier palabra. Este enfoque resulta fundamental para optimizar el almacenamiento y el rendimiento del modelo, especialmente en entornos multilingües o con un amplio espectro léxico [6].

La creación de un LLM comienza con una fase extremadamente costosa y computacionalmente intensiva conocida como **pre-entrenamiento** [4]. Este proceso puede entenderse como un gigantesco experimento de predicción de texto, en el que el modelo aprende a generar texto tras haber visto millones de ejemplos.

Para entrenar un modelo como **GPT-4 mini** [4], se recopilan **terabytes de texto** provenientes de múltiples fuentes: páginas web, artículos de Wikipedia, libros digitalizados, publicaciones científicas, código fuente de repositorios abiertos, redes sociales o foros técnicos. Todo este contenido se limpia, filtra y estructura para construit un corpus adecuado. No obstante, este contenido no puede utilizarse directamente, ya que suele contener ruido, errores de codificación o fragmentos irrelevantes. Por ello, antes de formar parte del corpus final, se somete a un riguroso proceso de preprocesamiento.

Este preprocesamiento se organiza en tres fases principales: limpieza, filtrado y estructuración. En la fase de limpieza, se eliminan elementos no textuales como etiquetas HTML, scripts o anuncios, además de corregir errores de codificación y suprimir duplicados. El filtrado se encarga de descartar contenido ofensivo, tóxico o ilegal, así como textos de muy baja calidad o generados de forma automática. También se controla la repetición excesiva de ciertos fragmentos para evitar el sobreajuste. Por último, el contenido restante se estructura en unidades coherentes, como párrafos, asegurando una adecuada diversidad temática y lingüística. En algunos casos, se añaden etiquetas que identifican el dominio o la fuente del texto, lo cual puede ser útil para el ajuste posterior del modelo [6, 7].

El objetivo final de este proceso es construir un corpus amplio, representativo y de alta calidad, que abarque desde lenguaje formal y técnico hasta registros conversacionales o creativos [4].

Una vez recopilado el corpus textual, comienza el entrenamiento del modelo, que consiste en **ajustar miles de millones de parámetros** para que el modelo aprenda a predecir el siguiente token con la mayor precisión posible. Estos parámetros son los valores internos (pesos y sesgos) que controlan las conexiones entre las neuronas artificiales en la red. Cada uno de ellos representa una pequeña fracción del conocimiento adquirido por el modelo, y su ajuste durante el entrenamiento permite aprender patrones complejos del lenguaje. En un modelo como GPT-4 mini, puede haber entre 10.000 y 70.000 millones de parámetros [4, 8]. Una forma intuitiva de entender su función es imaginar que el modelo es una gran orquesta, y los parámetros son la partitura: cada número indica cómo debe "responder" cada instrumento (neurona) ante una nota (entrada textual).

Este proceso requiere una infraestructura computacional masiva: grandes clústeres de GPUs (tarjetas gráficas especializadas), que trabajan en paralelo durante días o semanas. Por ejemplo, entrenar un modelo como GPT-4 mini [4], con decenas de miles de millones de parámetros, puede implicar utilizar miles de GPUs durante más de una semana, con un coste total que alcanza los millones de euros [9].

Durante el entrenamiento, el modelo analiza fragmentos de texto y predice cuál sería el siguiente token. Si se equivoca, **ajusta ligeramente sus parámetros internos** mediante algoritmos de optimización como el **descenso de gradiente**. Este algoritmo es el mecanismo principal mediante el cual la red neuronal aprende: primero se calcula la pérdida o error entre la predicción del modelo y la respuesta correcta. A continuación, se determina el gradiente, es decir, cuánto contribuye cada parámetro al error cometido. Por último, se actualizan todos los parámetros en la dirección opuesta al gradiente para reducir el error. Este ciclo de predicción, cálculo del error y ajuste de parámetros se repite millones de veces, permitiendo al modelo mejorar su capacidad de predicción con cada iteración [5].

Una vez finalizado el entrenamiento, el resultado no es un programa clásico, sino un **archivo binario masivo**, que contiene los **pesos del modelo neuronal**: es decir, miles de millones de números reales (usualmente en formato float16, de 16 bits) que codifican todo lo aprendido por el modelo.

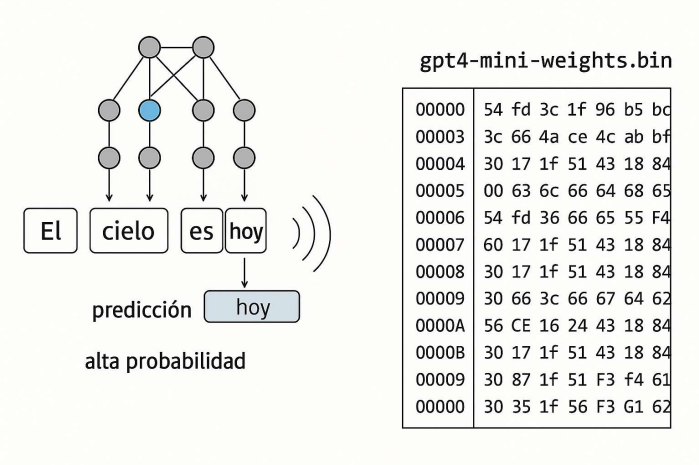


*Figura 1: Esquema simplificado del funcionamiento de un modelo de lenguaje. Elaboración propia basada en* [4] *OpenAI (2023) y* [5] *Brown et al. (2020)*

Por ejemplo, un modelo como **GPT-4 mini** [4] puede generar un archivo de muchos GB, que representa su conocimiento comprimido del lenguaje. Este archivo, junto con unas pocas **cientas de líneas de código** que implementan la arquitectura, constituye un **modelo funcional**. A partir de ahí, el modelo está listo para hacer **inferencia**, es decir, **generar texto nuevo** a partir de entradas proporcionadas por el usuario [4].

El modelo obtenido tras el pre-entrenamiento, aunque potente en términos de conocimiento, **no está preparado para interactuar con usuarios** de forma natural. Este modelo base tiende a generar texto con la misma estructura que el corpus en el que fue entrenado (páginas web, artículos, foros, etc.), pero no tiene por qué responder preguntas de manera clara, seguir instrucciones o comportarse de forma

segura y útil. Para convertir ese modelo base [4] en un **asistente conversacional alineado**, se realiza una segunda fase denominada **fine-tuning** [4, 6].

El fine-tuning es un proceso de ajuste adicional, más corto y específico, donde el modelo vuelve a entrenarse, pero esta vez **con datos cuidadosamente seleccionados**. Estos datos ya no son textos de Internet en general, sino **ejemplos de conversaciones, preguntas y respuestas bien formuladas**, instrucciones y tareas específicas [4, 5, 6].

Este proceso de ajuste fino suele dividirse en varias fases, cada una con un propósito específico. En primer lugar, se aplica el **fine-tuning supervisado (SFT)**, una etapa en la que el modelo es entrenado utilizando ejemplos concretos de instrucciones y respuestas redactadas por humanos. El objetivo es que aprenda a imitar el estilo, el formato y la estructura de una respuesta adecuada, alineándose con las expectativas del usuario [6, 10].

A continuación, se lleva a cabo el **aprendizaje por refuerzo con retroalimentación humana (RLHF)**. En esta fase, se generan múltiples respuestas para una misma instrucción y se solicita a evaluadores humanos que seleccionen la más apropiada. Esta elección se convierte en una señal de recompensa que el modelo utiliza para ajustar sus parámetros y mejorar su comportamiento. La finalidad de esta etapa es alinear la toma de decisiones del modelo con valores humanos deseables, como la utilidad, la veracidad y la seguridad [10, 11, 12]

Posteriormente, se recurre al **entrenamiento mediante comparaciones**, un método más eficiente en términos de coste y tiempo. En lugar de requerir respuestas nuevas, los evaluadores comparan dos o más opciones generadas y eligen la mejor. Esta estrategia permite refinar el modelo de forma iterativa, mejorando su desempeño sin necesidad de una gran carga de etiquetado adicional [10, 11].

Como resultado de estas fases, el modelo deja de limitarse a predecir el siguiente token de forma aislada y se transforma en un **asistente conversacional funcional**. Esta nueva capacidad le permite comprender instrucciones en lenguaje natural, generar respuestas coherentes, precisas y adaptadas al contexto, y ajustar su estilo de interacción según el tipo de usuario. Además, incorpora mecanismos para evitar, en la medida de lo posible, respuestas dañinas, sesgadas o incorrectas. Modelos como ChatGPT, Claude, Gemini o Copilot son ejemplos actuales de esta tecnología, y se emplean ampliamente en campos como la educación, la atención al cliente, la programación, la redacción de contenidos o la investigación científica [4, 6].

No obstante, el uso intensivo de datos generados por otros modelos LLM durante el entrenamiento o el ajuste fino ha generado una creciente preocupación en la comunidad investigadora, debido a un fenómeno conocido como **model collapse** [13, 6]. Este concepto hace referencia a la degradación progresiva en la calidad y fiabilidad de los modelos cuando se entrenan utilizando de forma predominante datos sintéticos, es decir, generados previamente por otros modelos en lugar de proceder de fuentes humanas originales. A medida que el volumen de contenido generado por modelos se incrementa y circula por internet, existe el riesgo de que estos textos sean recopilados e incluidos inadvertidamente en nuevos corpus de entrenamiento, lo que puede generar un ciclo de retroalimentación donde los errores, sesgos o repeticiones del pasado se amplifican [13].

Estudios recientes han demostrado que la exposición acumulada a datos sintéticos puede reducir la diversidad semántica de los modelos, empobrecer su capacidad de generalización y limitar su creatividad. En, trabajos como el de Shumailov [13] muestran cómo modelos entrenados con proporciones crecientes de contenido generado por IA tienden a perder precisión en tareas complejas, a reproducir patrones lingüísticos cada vez más predecibles y a degradarse en su rendimiento fuera de distribución [13].


*Figura 2: Ciclo de retroalimentación en el entrenamiento con datos generados por modelos LLM* [13]

Al mismo tiempo, la generación sintética de datos ha sido también explorada como una estrategia positiva para crear corpus balanceados, ampliar recursos en lenguas poco representadas o reforzar tareas específicas en escenarios de pocos ejemplos. Sin embargo, su aplicación indiscriminada plantea el riesgo de empobrecer la calidad del corpus global, especialmente cuando no se distingue claramente entre texto humano y texto generado. La formación de modelos sobre este tipo de datos puede llevar a una homogeneización del lenguaje, una pérdida de matices culturales y una mayor dificultad para mantener la precisión factual [6, 13].

Desde una perspectiva más amplia, este problema no solo tiene implicaciones técnicas, sino también éticas, sociales y económicas. Si los modelos se entrenan cada vez más sobre sí mismos, se genera una dependencia circular que reduce la diversidad de perspectivas, dificulta la innovación y puede afectar a la fiabilidad general de la información producida por sistemas automatizados. Además, se pierde el incentivo para recopilar y etiquetar nuevos datos originales, lo cual podría frenar el progreso hacia modelos más inclusivos, robustos y alineados con los valores humanos [6].

Con el objetivo de mitigar estos efectos, se están explorando diversas estrategias en el diseño de datasets y procesos de entrenamiento. Algunas de ellas se centran en filtrar activamente los textos generados por IA, entrenar clasificadores que detecten contenido sintético, o etiquetar los textos con marcas de agua para facilitar su identificación posterior [13]. Otras proponen reponderar los datos según su origen o diversidad, o incluso introducir mecanismos adversariales que obliguen al modelo a diferenciar entre ejemplos auténticos y generados, reforzando así su capacidad de razonamiento. Aunque estas líneas de investigación están todavía en desarrollo, representan un paso fundamental para garantizar que la evolución de los modelos LLM se base en principios de sostenibilidad, transparencia y calidad lingüística.

## 2.2 EVALUACIÓN DE MODELOS LLM: BENCHMARKS Y COMPARATIVAS

Una vez completado el proceso de fine-tuning, resulta esencial evaluar de forma rigurosa la capacidad del modelo para generalizar y desempeñarse en tareas reales. Para ello, se utilizan diversas estrategias que permiten medir su rendimiento en escenarios tanto estáticos como dinámicos.

Uno de los enfoques más comunes son los **benchmarks estandarizados**, conjuntos de pruebas públicas diseñadas para evaluar habilidades específicas del modelo, como razonamiento, comprensión del lenguaje o conocimiento factual. Estos benchmarks ofrecen un marco homogéneo y objetivo para comparar modelos de forma cuantitativa. Entre los más representativos se encuentra **MMLU (Massive Multitask Language Understanding)** [14] que abarca más de 50 disciplinas académicas y profesionales, evaluando el razonamiento y la erudición del modelo. Por su parte, **GSM8K** [15] se centra en la resolución de problemas matemáticos de nivel escolar, siendo una

referencia clave para medir el razonamiento lógico secuencial. Otros benchmarks como **ARC (AI2 Reasoning Challenge)** y **HellaSwag** están orientados a evaluar tareas de sentido común, inferencia y completado de frases en contextos ambiguos [16].

No obstante, estos benchmarks presentan una limitación importante: se desarrollan en contextos cerrados y estáticos, lo cual no siempre refleja el rendimiento real de los modelos en situaciones interactivas, como la generación conversacional o la comprensión de textos largos. Por ello, en trabajos recientes se han propuesto alternativas más dinámicas y empíricas.

Uno de los métodos más populares es la **evaluación por pares mediante sistema ELO**, inspirado en el ranking de jugadores de ajedrez. Este enfoque consiste en presentar una misma instrucción a dos modelos diferentes y ocultar su identidad. A continuación, se solicita a evaluadores humanos que escojan cuál de las respuestas es mejor. Con base en estas comparaciones acumuladas, se calcula una puntuación relativa para cada modelo, generando un ranking continuo de calidad conversacional. Esta metodología es empleada por la plataforma **Chatbot Arena**, desarrollada por **LMSYS** en colaboración con la Universidad de Berkeley [17], y ha sido utilizada para comparar modelos como Claude, GPT-4, Gemini o LLaMA en condiciones reales de interacción [18].

A diferencia de los benchmarks tradicionales, esta evaluación basada en preferencias humanas refleja mejor la utilidad, naturalidad y alineación de las respuestas con los valores del usuario. Además, permite capturar matices subjetivos difíciles de evaluar con métricas automáticas, como la claridad, la fluidez o el tono.

En el contexto de este trabajo, resulta especialmente interesante señalar cómo este tipo de evaluación puede complementarse con métricas automáticas de similitud semántica, como **ROUGE** [19], **BERTScore** [20] o comparativas por comprensión textual mediante test de preguntas, tal y como se implementa en el sistema desarrollado en los apartados experimentales. Mientras los benchmarks tradicionales evalúan habilidades "de examen", y las comparativas por pares valoran preferencia humana, las métricas de similitud ofrecen una visión objetiva de cómo el modelo mantiene el contenido y la coherencia durante procesos iterativos como resumen y expansión.

En resumen, la evaluación de LLMs debe abordarse desde múltiples ángulos: pruebas estandarizadas, comparativas humanas, métricas de fidelidad semántica y experimentos aplicados como los desarrollados en este trabajo. Cada enfoque aporta una perspectiva única y complementaria sobre la capacidad real del modelo.

## 2.3 MÉTRICAS AUTOMÁTICAS DE EVALUACIÓN

Tal y como se anticipó en el apartado anterior, una de las formas más efectivas para evaluar automáticamente las capacidades de generación de texto de los modelos LLM consiste en utilizar métricas que cuantifiquen el grado de similitud entre un texto generado y su referencia. En este trabajo se han empleado dos métricas ampliamente reconocidas y complementarias: **ROUGE** [19], **BERTScore** [20].

La métrica **ROUGE (Recall-Oriented Understudy for Gisting Evaluation)** es una familia de métricas ampliamente utilizada para evaluar tareas de resumen automático [19]. Su principio es contar cuántos fragmentos (n-gramas) del texto generado coinciden con los del texto de referencia. n *n-grama* es una secuencia de *n* elementos consecutivos en un texto; estos elementos suelen ser palabras, aunque también pueden ser caracteres. Entre sus variantes se encuentran ROUGE-N, que evalúa la superposición de n-gramas como unigrama (n=1) o bigrama (n=2), y ROUGE-L, que mide la longitud de la subsecuencia común más larga (LCS). El proceso consiste en generar todos los n-gramas posibles de ambos textos, calcular las coincidencias y reportar tres métricas:

- **Precisión:** Proporción de n-gramas en el texto generado que coinciden con el texto de referencia.

$$\text{Precisión} = \frac{\text{n-gramas coincidentes}}{\text{n-gramas del texto generado}} \quad (1)$$

- **Recall:** Proporción de n-gramas en el texto de referencia que están presentes en el texto generado.

$$\text{Recall} = \frac{\text{n-gramas coincidentes}}{\text{n-gramas del texto de referencia}} \quad (2)$$

- **F1-score:** Promedio armónico entre Recall y Precisión.

$$F1 = 2 \cdot \frac{\text{Precisión} \cdot \text{Recall}}{\text{Precisión} + \text{Recall}} \quad (3)$$

Esta métrica es simple y rápida de calcular, aunque no capta relaciones semánticas como sinónimos o paráfrasis, y es sensible a ligeros cambios de redacción [19, 21].

Por ejemplo, si el texto de referencia es "El gato negro está en el tejado" y el generado es "Un felino oscuro se encuentra sobre el tejado", la coincidencia literal es baja y el F1-score con ROUGE se situaría aproximadamente en 0.27.

En cambio, **BERTScore** evalúa la similitud semántica entre textos utilizando representaciones vectoriales (embeddings) [20] derivadas de modelos de lenguaje preentrenados como BERT o RoBERTa. Este enfoque permite comparar palabras incluso cuando no coinciden exactamente, ya que cada palabra se representa mediante un vector que varía según su contexto [22].

El cálculo de **BERTScore** [23] se desarrolla en varias etapas fundamentales. La primera consiste en la **obtención de embeddings contextuales**, donde cada palabra es transformada en un vector numérico cuyo valor depende de su contexto específico. Por ejemplo, el término "banco" adopta vectores distintos en frases como "el banco de madera" o "el banco financiero". Modelos preentrenados como **BERT**, **RoBERTa**, **XLNet** o **XLM** son responsables de generar estos embeddings, representando tanto el significado como las relaciones sintácticas entre palabras.

Una vez que tenemos los embeddings (vectores) de cada palabra, se calcula la **similitud coseno** para medir qué tan similares son dos palabras, una del texto generado (candidato) y otra del texto de referencia. La similitud del coseno es una forma de medir la similitud entre dos vectores en un espacio multidimensional. Se utiliza ampliamente en procesamineto del lenguaje natural (NLP), incluido el cálculo de similitud entre embeddings.

Mide el coseno del ángulo entre dos vectores en un espacio vectorial:

$$\cos(\theta) = \frac{\vec{A} \cdot \vec{B}}{\left|\left|\vec{A}\right|\right| \cdot \left|\left|\vec{B}\right|\right|} \quad (4)$$

$\vec{A} \cdot \vec{B}$ : producto escalar entre los vectores.

$|\vec{A}|$ : norma (longitud) del vector $\vec{A}$

$|\vec{B}|$ : norma (longitud) del vector $\vec{B}$

El resultado de esta fórmula oscila entre -1 y 1, indicando si los vectores son opuestos, ortogonales o apuntan en la misma dirección. De manera equivalente, puede expresarse como:

$$\cos(\theta) = \frac{\sum_{i=1}^{n} a_i b_i}{\sqrt{\sum_{i=1}^{n} a_i^2} \cdot \sqrt{\sum_{i=1}^{n} b_i^2}} \quad (5)$$

A continuación, se lleva a cabo la **coincidencia de tokens**, donde cada palabra del texto generado busca su mejor coincidencia semántica con alguna del texto de referencia, y viceversa. Este proceso permite calcular las siguientes métricas:

- Precisión: Qué tan bien el texto generado representa las palabras del texto de referencia. Se calcula encontrando la mejor coincidencia (más alta similitud coseno) para cada palabra en el texto.

$$P_{\text{BERT}} = \frac{1}{|\hat{x}|} \sum_{\hat{x_j} \in \hat{x}} \max_{x_i \in x} \vec{x_i} \cdot \vec{\hat{x}_j} \quad (6)$$

- Recall: Qué tan bien las palabras del texto de referencia están representadas en el texto generado. Se calcula encontrando la mejor coincidencia para cada palabra en el texto de referencia.

$$R_{\text{BERT}} = \frac{1}{|x|} \sum_{x_i \in x} \max_{\hat{x_j} \in \hat{x}} \vec{x_i} \cdot \vec{\hat{x}_j} \quad (7)$$

- F1-Score: Combina Precisión y Recall.

$$F_{\text{BERT}} = \frac{2 \cdot P_{\text{BERT}} \cdot R_{\text{BERT}}}{P_{\text{BERT}} + R_{\text{BERT}}} \quad (8)$$

Opcionalmente, se puede aplicar una **ponderación por IDF** (Frecuencia Inversa de Documento) para dar más peso a palabras significativas y menos a términos comunes. Por ejemplo, palabras comunes como "el", "y", o "es" aportan poca información, mientras que palabras raras como "fotovoltaico" o "cuántico" son más significativas. ¿Cómo se ajusta? La Frecuencia Inversa de Documento (IDF) mide qué tan rara es una palabra en un conjunto de datos. Las palabras con alta IDF tienen más peso en el cálculo de similitud.

$$\text{idf}(w) = -\log\left(\frac{1}{M} \sum_{i=1}^{M} I\left[w \in x^{(i)}\right]\right) \quad (9)$$

Donde M es el número total de oraciones de referencia y I\mathbb{I}I es una función indicadora. El recall ponderado por IDF se define como:

$$R_{\text{BERT}}^{\text{idf}} = \frac{\sum_{x_i \in x} \text{idf}(x_i) \cdot \max_{\hat{x_j} \in \hat{x}} \vec{x_i} \cdot \vec{\hat{x}_j}}{\sum_{x_i \in x} \text{idf}(x_i)} \quad (10)$$

Por último, se realiza un **reescalado final** de los resultados. Los valores crudos de BERTScore pueden ser difíciles de interpretar porque varían según el modelo de embeddings y el tipo de texto. Para hacerlos más intuitivos, se ajustan a un rango más manejable basado en datos grandes como los del Common Crawl (un corpus de texto masivo). Este reescalado permite comparar puntajes de manera más directa y facilita su uso como referencia en distintas tareas se normaliza a un rango interpretativo entre 0 y 1.

$$\widehat{R_{\text{BERT}}} = \frac{R_{\text{BERT}} - b}{1 - b} \qquad (11)$$

Este enfoque ofrece ventajas claras, como su capacidad para captar relaciones semánticas profundas y su robustez frente a reformulaciones. No obstante, implica un mayor coste computacional y depende del modelo preentrenado utilizado. A modo de ejemplo, si el texto de referencia es "El gato negro está en el tejado" y el generado es "Un felino oscuro se encuentra sobre el tejado", BERTScore será capaz de reconocer equivalencias como "gato" ↔ "felino", "negro" ↔ "oscuro" y "en el tejado" ↔ "sobre el tejado", arrojando un F1-score > 0.85.

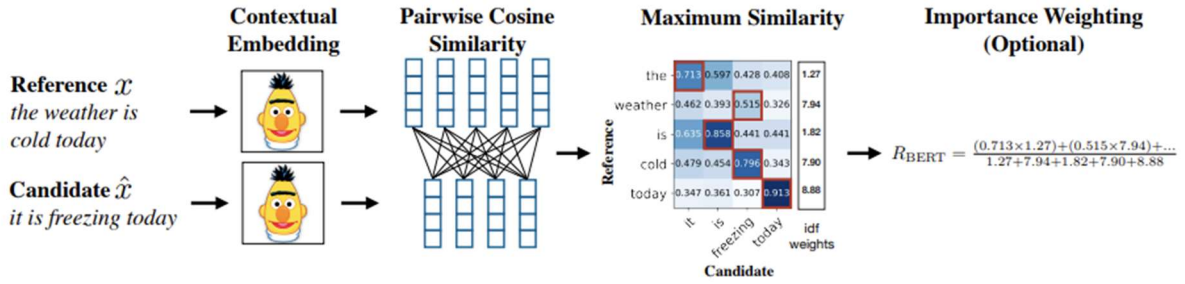


*Figura 3: Proceso de cálculo de BERTScore: embeddings contextuales, cálculo de similitud coseno entre palabras, selección de máximas similitudes y ponderación opcional por IDF* [20].

**Comparativa final**

| Métrica | Nivel de análisis | Captura significado | Sensible a sinónimos | Coste computacional |
|---|---|---|---|---|
| *ROUGE* | Superficial | No | Sí | Bajo |
| *BERTScore* | Semántico | Sí | No | Alto |

*Tabla 1: Comparativa entre las métricas ROUGE y BERTScore*

## 2.4 EVALUACIÓN POR COMPRENSIÓN SEMÁNTICA

Además de las métricas automáticas tradicionales como ROUGE [19] o BERTScore [20], que se centran en la similitud superficial o semántica entre textos, la comunidad investigadora ha comenzado a explorar métodos más orientados a la **evaluación de la comprensión real del contenido**. En este contexto, una línea emergente de trabajo propone **evaluar modelos LLM mediante pruebas de**

**comprensión lectora estructuradas, como las preguntas tipo test**, como una alternativa eficaz para medir la capacidad del modelo de retener, razonar y extraer información relevante a partir de un texto.

Este enfoque se basa en una idea sencilla pero poderosa: si un sistema es capaz de responder correctamente a una serie de preguntas generadas a partir de un contenido original, puede inferirse que ha comprendido, al menos en parte, ese contenido. Esta evaluación se realiza de manera explícita, exigiendo al modelo que extraiga hechos, relacione conceptos y aplique inferencias a partir del texto leído.

Diversos benchmarks existentes ya adoptan este paradigma, como **RACE** (Reading Comprehension from Examinations) o **DROP**, que evalúan la comprensión a partir de textos académicos o de pasajes con preguntas inferenciales. De forma similar, **datasets como ARC (AI2 Reasoning Challenge)** y **MMLU (Massive Multitask Language Understanding)** también incluyen tareas donde la comprensión semántica se pone a prueba mediante preguntas de opción múltiple, abarcando desde cultura general hasta razonamiento abstracto [14, 16].

Estos métodos presentan varias ventajas frente a las métricas tradicionales. En primer lugar, permiten evaluar habilidades cognitivas complejas como la inferencia, el resumen, la deducción o la integración de información, aspectos que escapan a los enfoques puramente estadísticos [6]. Además, son más robustos frente al uso de sinónimos o paráfrasis, a diferencia de métricas como ROUGE [19], que se basan en coincidencias literales de n-gramas. Por último, tienen una aplicación directa en contextos educativos y profesionales, donde pueden emplearse para tareas como la asistencia al estudio o la implementación de sistemas de tutoría automática [10, 24].

No obstante, también presentan desafíos importantes. La **generación automática de preguntas fiables** no es trivial: requiere preservar la información clave sin introducir ambigüedad, y mantener un equilibrio entre dificultad, claridad y variedad. Además, la **evaluación automática de las respuestas** debe realizarse con precisión, lo que en muchos casos obliga a definir esquemas estructurados (por ejemplo, en formato JSON) y a implementar validaciones robustas.

Desde un punto de vista teórico, este enfoque conecta con trabajos previos sobre evaluación semántica profunda. Investigaciones como las de Shumailov [13] advierten de que los modelos pueden perder progresivamente capacidad de generalización si no se evalúan con tareas exigentes en comprensión. Asimismo, Bommasani [6] argumentan que los modelos fundacionales deben ser evaluados desde múltiples dimensiones, y que la comprensión semántica es una de las más críticas para su aplicación responsable.

En definitiva, la evaluación basada en comprensión lectora ofrece un marco alternativo y complementario a las métricas convencionales. Permite analizar no solo qué tan similar es un texto al original, sino **cuánto ha entendido realmente el modelo del contenido que procesa**, aportando así una dimensión más rica y centrada en el propósito último del lenguaje: la comunicación de significado.

# 3. DESARROLLO DEL SISTEMA EXPERIMENTAL

## 3.0 ADQUISICIÓN DE COMPETENCIAS TÉCNICAS PREVIAS

Antes de abordar el desarrollo del sistema experimental, fue necesario adquirir una base sólida de competencias técnicas en programación y análisis de datos. Esta fase inicial de proyecto, equivalente al primer sprint de trabajo, se centró en el aprendizaje del lenguaje Python y el manejo de bibliotecas fundamentales como Pandas, Numpy y MatplotLib.

Para garantizar una base técnica sólida en el desarrollo del proyecto, se realizaron distintas actividades formativas orientadas a consolidar competencias clave en programación y análisis de datos. En primer lugar, se completó un **curso introductorio de Python** [25] que permitió afianzar los fundamentos del lenguaje, incluyendo estructuras de control, funciones, listas, diccionarios, comprensión de listas y gestión de archivos. Además, este curso sirvió como punto de partida para familiarizarse con bibliotecas ampliamente utilizadas como **NumPy**, especializada en operaciones numéricas sobre arrays, y **Matplotlib**, empleada para la creación de visualizaciones y gráficos.

A continuación, se abordó un curso específico sobre la biblioteca **Pandas** [26] accesible a través de GitHub. Este material ofrecía una introducción progresiva al tratamiento de datos estructurados, cubriendo aspectos esenciales como la manipulación de tablas, filtrado de registros, agrupaciones, gestión de valores nulos y ejecución de operaciones estadísticas básicas.

Finalmente, se llevaron a cabo diversos ejercicios prácticos [27], que permitieron aplicar los conocimientos adquiridos en un contexto realista. Estas actividades facilitaron el uso combinado de Pandas y Matplotlib para el análisis exploratorio de datos, el diseño de visualizaciones personalizadas y la implementación de técnicas de agrupamiento y transformación de conjuntos de datos.

Esta etapa formativa resultó crucial para asegurar un uso eficaz de Python en las fases posteriores del trabajo, en particular para el desarrollo de scripts destinados a la interacción con APIs, el procesamiento masivo de textos y la evaluación automática mediante métricas. El dominio de estas herramientas proporcionó una base técnica robusta sobre la que construir un sistema experimental flexible, reproducible y escalable.

## 3.1 DESARROLLO DEL SISTEMA DE EVALUACIÓN BASADO EN BUCLES DE RESUMEN Y EXPANSIÓN

Una vez consolidada la base teórica sobre el funcionamiento de los LLMs, se procedió a desarrollar un sistema experimental diseñado específicamente para observar cómo se comportan estos modelos cuando son sometidos a ciclos iterativos de **compresión (resumen) y expansión (ampliación)** textual. El objetivo principal de este apartado es describir el diseño, implementación y ejecución de dicho sistema, analizando cada una de sus etapas: desde la recopilación y preparación del corpus de entrada hasta la generación automática de textos mediante modelos de lenguaje accesibles por API. Este enfoque permite evaluar si, tras varias transformaciones, los modelos mantienen la coherencia semántica, la fidelidad a la información original y la calidad lingüística del contenido.

El procedimiento consiste en tomar un texto original y generar a partir de él un resumen. Posteriormente, ese resumen es expandido por el modelo hasta alcanzar una longitud similar a la del texto original. Sin embargo, el proceso no se detiene ahí: el nuevo texto generado es nuevamente resumido, y este nuevo resumen vuelve a expandirse. De este modo, se establece un ciclo de iteraciones sucesivas en el que cada paso se construye sobre el resultado anterior, no sobre el texto original. Esto permite analizar cómo se degrada o transforma el contenido a lo largo del tiempo, revelando tanto la capacidad de retención semántica del modelo como su tendencia a introducir errores acumulativos,

alteraciones estilísticas o desviaciones informativas. El ciclo completo se repite un número predefinido de veces **(diez iteraciones)**, almacenando los resultados de cada etapa para su posterior análisis cuantitativo y cualitativo.

Para llevar a cabo este experimento, se ha trabajado con arquitecturas de distintos proveedores, como LLaMA, GPT-4 o Gemini, integradas mediante sus respectivas APIs. Además, se ha desarrollado una lógica de generación recursiva completamente automatizada que permite ejecutar estos procesos en bloque sobre grandes volúmenes de datos, facilitando el análisis comparativo entre modelos, iteraciones y estrategias de generación.

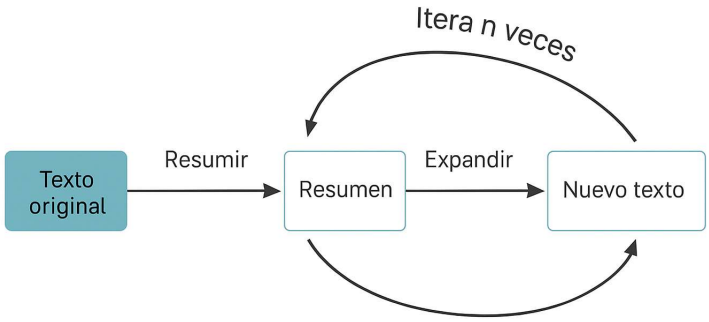


*Figura 4: Esquema del ciclo iterativo de resumen y expansión textual.*

### 3.1.1 Obtención del corpus de prueba

Para la evaluación del sistema desarrollado, se empleó el conjunto de datos **CNN/DailyMail** [3], una referencia ampliamente utilizada en tareas de resumen automático. Este dataset contiene noticias acompañadas de sus respectivos resúmenes redactados por humanos, lo que lo convierte en un recurso idóneo para experimentar con modelos de compresión y expansión textual.

Con el objetivo de automatizar la descarga y estructuración del corpus de prueba, se desarrolló **"Descarga_textos.py"** [1] que se conecta a la API pública de **Hugging Face**, plataforma que alberga miles de datasets de libre acceso. Para ello, se configura una clave de autenticación personal que permite realizar solicitudes autorizadas a través de dicha API.

Una de las funciones principales desarrolladas durante la implementación del sistema experimental es ***"fetch_single_text()"***, cuya finalidad es descargar un único texto del dataset a partir de un índice (offset) determinado. Esta función realiza una solicitud HTTP a la fuente de datos y, en caso de obtener una respuesta satisfactoria, procesa el contenido recibido para extraer información clave. Concretamente, recupera el identificador único del artículo (id), el texto completo del mismo (article), el resumen asociado (highlights) y la longitud en palabras tanto del texto original como del resumen.

Una vez descargados los textos, estos se almacenan en un archivo estructurado en formato Excel, lo que permite un análisis posterior ordenado y reproducible. Para ello, se utiliza la función "***save_to_excel()"***, que convierte la lista de artículos en un DataFrame y los guarda en el disco.

El control general del script recae en el bloque principal main, que coordina el proceso completo de recopilación. A través de un bucle while, se descargan automáticamente textos del dataset incrementando el índice offset, lo que garantiza que se obtengan artículos distintos sin repeticiones.

El objetivo es reunir un corpus de 1000 textos, partiendo desde un punto determinado del dataset (definido por el offset). A medida que se descargan los textos válidos, estos se almacenan en una lista. Periódicamente se imprime el progreso por consola, y al finalizar, se invoca la función "***save_to_excel()"*** para guardar el resultado en disco.

### 3.1.2 Implementación preliminar de generación iterativa con Groq

En esta fase del proyecto se implementaron los bucles de resumen y expansión textual sobre el corpus previamente obtenido, utilizando inicialmente el modelo **llama-3.2-1b-preview** a través de la API de Groq. Sin embargo, debido a que este modelo quedó obsoleto durante el transcurso del trabajo, se migró la lógica experimental a otros modelos más actuales: **gpt-4o-mini-2024-07-18**, accesible mediante OpenRouter, y **gemini-1.5-flash**, ofrecido por Google. Esta evolución permitió mantener la continuidad del experimento y, al mismo tiempo, comparar los resultados entre diferentes arquitecturas de LLMs.

El script "**bucle_resumen_expansion_groq.py**" [1] comienza estableciendo la conexión con el cliente Groq (en la versión inicial) mediante la configuración de la clave API. A continuación, se define la función *"**summary_text()**"*, encargada de generar un resumen del texto original. Esta función recibe como parámetros el modelo de lenguaje a utilizar, el texto original (original_text), la longitud aproximada del resumen en palabras (max_summary_length) y el parámetro temperature, que controla el grado de aleatoriedad en la generación.

El proceso que sigue la función *"**summary_text()**"* consiste en construir una solicitud al modelo de lenguaje, donde se incluye el texto original junto con una instrucción clara que le indica resumirlo hasta una longitud determinada. Esta instrucción se envía como mensaje de tipo "system", seguido del mensaje del usuario que contiene el texto a resumir. Una vez recibida la solicitud, el modelo genera una versión resumida del contenido, ajustada a la longitud indicada en palabras. Este resumen se devuelve como resultado de la función y se conserva para ser utilizado posteriormente en la siguiente fase del ciclo iterativo de generación.

La función *"**complete_text()**"* se encarga de realizar la expansión textual, transformando un resumen previamente generado en un texto completo de longitud aproximada especificada. Al igual que en el proceso de resumen, se construye una solicitud estructurada que incluye una instrucción (system message) indicando al modelo que expanda el contenido del resumen hasta alcanzar un número determinado de palabras. A esta instrucción le sigue el resumen original en el rol de usuario.

El modelo de lenguaje, a partir de esta entrada, genera una versión extendida del texto, cuyo contenido se aproxima a la longitud deseada. Esta fase simula el proceso inverso al de compresión, y permite evaluar si el modelo es capaz de mantener la coherencia temática, la fidelidad informativa y la continuidad narrativa al volver a extender el contenido resumido. El texto generado es almacenado y puede ser utilizado como nuevo punto de partida para posteriores iteraciones dentro del ciclo de evaluación.

Una vez definidas las funciones encargadas de generar resúmenes y textos expandidos, se implementa la función ***"run_iterations()"***, que permite aplicar de forma recursiva ambos procesos sobre un conjunto de entradas. Esta función constituye el núcleo experimental del sistema desarrollado, ya que reproduce un escenario de transformación iterativa de textos que permite evaluar la estabilidad, fidelidad y degradación de los modelos de lenguaje a lo largo de sucesivas generaciones.

La ejecución comienza recorriendo cada texto del conjunto. Para cada uno de ellos, se almacenan sus propiedades básicas (identificador, longitud y resumen original), y se inicializa la cadena de transformación con el texto original.

En cada ciclo, el modelo produce un nuevo resumen del contenido actual, y luego lo expande de nuevo hasta alcanzar una longitud similar a la del texto inicial. Así, se construye un bucle controlado en el que el modelo reformula su propia salida en varias ocasiones consecutivas.

Además, se han introducido medidas de robustez para asegurar el correcto funcionamiento del proceso: cada llamada a la API externa se ejecuta dentro de un bloque de reintento que permite realizar hasta tres intentos antes de registrar un fallo. Esto permite mitigar errores puntuales debidos a desconexiones o sobrecarga de los servicios remotos. Por otro lado, tras cada iteración se actualiza el

texto de entrada, de modo que la siguiente generación parte del contenido ya transformado. Esta propiedad es clave para evaluar la acumulación de errores o desviaciones en los modelos.

Una vez completadas todas las iteraciones para un texto, los resultados se agregan a una tabla estructurada. Finalmente, la función devuelve un DataFrame con todas las generaciones asociadas.

Una vez preparado el corpus de prueba y almacenado en un archivo Excel, fue necesario implementar una función que permitiera cargar de forma controlada un subconjunto de textos a procesar. Para ello se definió la función ***"read_texts_from_excel()"***, cuya finalidad es leer los textos a partir de una fila específica (start_row) y limitar el número total de entradas mediante el parámetro limit. Este diseño permite lanzar los experimentos en bloques, facilitando tanto la trazabilidad como la gestión de recursos en ejecuciones sucesivas.

El bloque principal del script tiene como función orquestar todo el proceso descrito anteriormente: carga del corpus, configuración de los parámetros de prueba, ejecución de los bucles de resumen y expansión, y almacenamiento final de los resultados.

En primer lugar, se leen desde un archivo Excel los textos que van a ser procesados. A continuación, se configuran los parámetros de la prueba: número de iteraciones, temperatura y modelo a utilizar. Con esta configuración, se calcula el número total de pasos para llevar un seguimiento preciso del progreso.

Acto seguido, se ejecuta la función ***"run_iterations()"***, que genera las versiones resumidas y expandidas de cada texto en cada iteración. Finalmente, todos los resultados se almacenan de forma estructurada en una hoja Excel para su posterior análisis.

Este enfoque permite observar cómo evoluciona el contenido generado por el modelo a lo largo de múltiples transformaciones, lo cual resulta útil para evaluar su coherencia, estabilidad semántica y conservación de la información.

### 3.1.3 Estructura del archivo de resultados

Tras ejecutar el experimento de resumen y expansión textual, los resultados se almacenan automáticamente en un archivo Excel [2]. Este archivo constituye la base de datos sobre la que se realizarán los análisis posteriores, ya que recoge todas las generaciones intermedias producidas por el modelo en cada iteración.

El archivo de resultados adoptado en este trabajo posee una estructura tabular, en la que cada fila corresponde a un texto procesado del corpus de entrada y cada columna almacena información relevante sobre las distintas transformaciones sufridas por dicho texto a lo largo del bucle experimental. Las columnas pueden clasificarse en dos grandes grupos.

En primer lugar, se incluyen los **identificadores y metadatos originales**, como el id del texto, el contenido completo (original_text), el número de palabras que lo componen (original_text_length), el resumen elaborado por humanos (original_summary) y la longitud de dicho resumen (original_summary_length).

En segundo lugar, se encuentran los **resultados obtenidos en cada iteración**, organizados en columnas del tipo summary_1_temp_1.0, new_text_1_temp_1.0, ..., hasta summary_10_temp_1.0 y new_text_10_temp_1.0. Cada una de estas columnas contiene los resúmenes y expansiones generados en una iteración concreta, habiéndose fijado la temperatura de generación en 1.0. El número que aparece en el sufijo permite identificar fácilmente a qué iteración corresponde cada contenido generado.

Gracias a este diseño, se facilita el análisis comparativo entre versiones sucesivas de un mismo texto, permitiendo observar con detalle cómo evoluciona la información a medida que el modelo alterna entre compresión y expansión. Esta disposición es clave para estudiar fenómenos

como la pérdida semántica, la introducción de ruido o la reformulación progresiva del contenido original.

### 3.1.4 Migración a modelos GPT-4-mini y Gemini

Durante el desarrollo del proyecto, el modelo inicial seleccionado **LLaMA-3.2-1b-preview**, accesible a través de la API de Groq fue retirado por el proveedor, lo que obligó a adaptar el sistema experimental para su ejecución con modelos alternativos. Con el doble propósito de mantener la continuidad de los experimentos y enriquecer el análisis comparativo, se migró la lógica implementada a dos modelos más recientes y ampliamente disponibles: **GPT-4o-mini-2024-07-18**, accesible mediante la plataforma OpenRouter y la biblioteca openai, y **Gemini-1.5-flash**, ofrecido por Google AI Studio e integrado con google. generativeai.

En el caso de **GPT-4 Mini** se implemento **"bucle_resumen_expansion_gpt4.py"** [1], donde la estructura general del código se conservó sin modificaciones sustanciales. Las adaptaciones clave incluyeron la configuración de la API mediante un cliente persistente de OpenAI, en el que la clave (api_key) se proporcionó de forma explícita en el constructor en lugar de utilizar variables de entorno. Las llamadas al modelo se gestionaron mediante el método client. chat. completions. create(), y se incorporó además una función auxiliar, ensure_excel_file_exists (), que garantiza la existencia previa del archivo de salida antes de almacenar los resultados generados. Estas modificaciones permitieron ejecutar las mismas iteraciones de resumen y expansión manteniendo intacta la lógica experimental, lo que aseguró la comparabilidad directa de los resultados con los obtenidos previamente.

La adaptación al modelo **Gemini** se materializó en **"bucle_resumen_expansion_gemini.py"** [1] e implicó algunos ajustes adicionales debido a diferencias en la interfaz y el entorno de ejecución. Se utilizó la biblioteca oficial google. generativeai y el modelo fue instanciado a través de genai. GenerativeModel. Las solicitudes de inferencia se realizaron con el método generate_content(), que acepta directamente el prompt como una cadena de texto. Para gestionar adecuadamente las limitaciones del servicio, se implementaron mecanismos de control de cuota, incluyendo temporizadores que pausaban automáticamente la ejecución en caso de superar el número máximo de solicitudes por minuto o por día. Asimismo, se incorporó una gestión específica de excepciones como ResourceExhausted, propias del entorno de Google, para evitar interrupciones en la ejecución masiva de pruebas.

Gracias a esta migración, se consiguió ampliar la cobertura experimental del sistema, obteniendo resultados comparables bajo arquitecturas diferentes de LLMs, lo cual enriquece significativamente el análisis posterior.

**Diferencias clave entre implementaciones**

| Elemento | Groq (LLaMA) | GPT-4 Mini (OpenRouter) | Gemini (Google) |
|---|---|---|---|
| *Cliente API* | Groq () | OpenAI (api_key=...) | genai. GenerativeModel(...) |
| *Llamada de resumen* | groq_client. chat. completions. create | client. chat.completions.create | generate_content () |
| *Manejo de cuotas* | No requerido | No requerido | Sí, con lógica de control por tiempo |

| Elemento | Groq (LLaMA) | GPT-4 Mini (OpenRouter) | Gemini (Google) |
|---|---|---|---|
| *Estilo de prompts* | Instrucciones separadas por roles | Igual | Prompt en texto completo |
| *Gestión de errores* | Reintentos manuales | Reintentos manuales | Reintentos + espera automática |

*Tabla 2: Comparativa de implementación entre clientes API utilizados en el experimento: LLama, GPT-4 Mini y Gemini.*

## 3.2 EVALUACIÓN DE MÉTRICAS

Para poder comparar cuantitativamente los efectos del ciclo iterativo de resumen y expansión sobre los textos generados, fue necesario desarrollar un sistema automatizado de evaluación. Este sistema tenía como objetivo aplicar distintas métricas de comparación entre versiones generadas por los modelos y sus respectivos textos de referencia, permitiendo así analizar con precisión cómo evolucionan los contenidos generados por los LLMs tras múltiples transformaciones. En este apartado se describe la implementación del script encargado de calcular estas métricas y su integración dentro del flujo general del experimento.

### 3.2.1 Implementación

Para evaluar cuantitativamente la calidad de los textos generados por los modelos LLM en los ciclos de resumen y expansión, se desarrolló **"calculo_metricas.py"** [1] que permite calcular dos tipos de métricas: ROUGE, BERTScore. La implementación está orientada a un procesamiento incremental que evita repetir cálculos innecesarios, optimizando así el uso de recursos computacionales y facilitando múltiples ejecuciones controladas del experimento.

El núcleo del sistema está compuesto por funciones especializadas que encapsulan cada una de las métricas. Para el cálculo de **ROUGE-1** se utiliza la clase ***"RougeScorer"*** de la biblioteca oficial "***rouge_score"***, desarrollada y mantenida por investigadores de Google Research [28]. Esta herramienta permite calcular precisión, recall y F1-score en función de la superposición literal de n-gramas entre los textos de referencia y los generados. La función definida como ***"calculate_rouge ()"***, que actúa como envoltorio simplificado. Esta función recibe como entrada un texto de referencia y una hipótesis (resumen o expansión), invoca internamente a ***"RougeScorer"*** con la opción de stemming activada (use_stemmer=True), y devuelve un diccionario con los tres valores métricos fundamentales: precisión, recall y F1. Este diseño facilita su aplicación sistemática sobre grandes conjuntos de comparaciones textuales.

Para evaluar la similitud semántica entre textos, se utilizó la biblioteca ***"bert_score",*** implementada por **Hugging Face** y publicada originalmente como parte del trabajo de Zhang et al. (2020) en ICLR [20]. Esta herramienta convierte los textos en vectores de embeddings contextuales generados por modelos preentrenados como BERT o RoBERTa, y compara cada palabra con su mejor equivalente semántico en el texto contrario utilizando la similitud del coseno. La función ***"calculate_bertscore()"*** encapsula esta lógica, llamando a ***"bert_score. score()"*** y extrayendo las medias de precisión, recall y F1-score del conjunto de comparaciones, lo que permite obtener una medida fiable de la cercanía semántica entre cada par de textos [29].

Una vez definidas las funciones específicas para calcular las distintas métricas (ROUGE [19], BERTScore [20]), se requiere una función genérica que pueda aplicarlas de forma sistemática

sobre un conjunto de comparaciones. Para ello, se implementó ***"process_comparisons ()"***, una función que automatiza el proceso de evaluación entre textos de referencia y múltiples versiones generadas (resúmenes o expansiones) contenidas en un DataFrame.

La función recibe como entrada el propio DataFrame con los datos (df), el nombre de la columna que contiene el texto de referencia (reference_col), una lista de columnas con las hipótesis generadas (comparison_cols), y una función de métrica (metric_func) que se aplicará a cada par de textos. Por cada fila del DataFrame, se extrae el texto original y se compara con cada una de las versiones generadas disponibles, siempre que su valor no sea nulo (NaN). Para cada par referencia–hipótesis, se ejecuta la función de evaluación correspondiente, obteniendo las métricas de precisión, recall y F1-score, que se almacenan en un diccionario estructurado con nombres de columna codificados según el tipo de comparación.

El resultado final es un nuevo DataFrame donde cada fila representa las métricas asociadas a un único texto original, incluyendo todas sus comparaciones posibles. Esta estructura permite un análisis posterior detallado y reproducible del rendimiento de los modelos bajo estudio.

Esta función es esencial para escalar la evaluación a cientos de textos y múltiples versiones generadas por modelos LLM, integrando el cálculo de métricas de forma automatizada y ordenada dentro del flujo experimental.

El bloque principal del script (main) se encarga de coordinar la ejecución completa del proceso de evaluación automática de los textos generados. Su diseño está orientado a la eficiencia y a la reproducibilidad, permitiendo reanudar el procesamiento de forma incremental sin repetir evaluaciones ya realizadas.

La ejecución comienza leyendo el archivo de entrada, que contiene los textos generados por los modelos de lenguaje tras las iteraciones de resumen y expansión. A continuación, se verifica si existe un archivo de resultados previo. En caso afirmativo, se recopilan los identificadores (id) de los textos ya evaluados consultando cada una de las hojas del archivo Excel existente. Esta lógica permite filtrar y procesar únicamente aquellos textos nuevos que aún no han sido evaluados, evitando así redundancias y ahorrando tiempo computacional.

El script define dos conjuntos de columnas para las comparaciones: aquellas que contienen los resúmenes generados (summary_*) y aquellas que contienen los textos expandidos (new_text_*). A partir de estas columnas, se aplican de forma sistemática las tres métricas de evaluación definidas previamente:

- **ROUGE**: Aplicada tanto entre el texto original y los resúmenes, como entre el texto original y los textos generados, y también entre el resumen original y los resúmenes generados.
- **BERTScore**: Utilizada con la misma lógica comparativa, capturando similitud semántica entre pares de textos.

Finalmente, los resultados de cada tipo de comparación se almacenan en hojas separadas dentro de un archivo Excel. El modo de escritura es incremental ('a'), lo que permite mantener ejecuciones sucesivas del script sin sobrescribir información previa. Esto convierte al sistema en una herramienta robusta y flexible para la evaluación automatizada de experimentos de generación textual con LLMs

### 3.1.3 Estructura del archivo de resultados

Tras la ejecución del script de evaluación automática, los resultados se almacenan de forma incremental en un archivo Excel estructurado en múltiples hojas [2]. Cada una de estas hojas corresponde a una combinación específica de comparación entre versiones de texto y la métrica de evaluación aplicada, lo que permite llevar a cabo análisis detallados y diferenciados para cada escenario experimental. Esta organización facilita la comparación sistemática entre las distintas versiones generadas por los modelos de lenguaje y sus respectivos textos de referencia.

Dentro de cada hoja, los datos se presentan en formato tabular. Las columnas incluyen, en primer lugar, un identificador único (id) que permite establecer correspondencia entre cada texto evaluado y su entrada original en el archivo de resultados base. A continuación, se presentan las métricas calculadas para cada par de comparación. Las columnas siguen una nomenclatura clara y coherente que combina el tipo de transformación, la iteración correspondiente, el parámetro de temperatura y la métrica evaluada. Por ejemplo, la columna summary_1_temp_1.0-precision recoge la precisión (según ROUGE o BERTScore) entre el texto original y el primer resumen generado; new_text_3_temp_1.0-f1 representa el F1-score correspondiente a la tercera iteración de expansión; y summary_7_temp_1.0-recall muestra el recall calculado entre el resumen de la séptima iteración y el texto de referencia.

El archivo está organizado en hojas independientes, cada una dedicada a una combinación concreta de comparación y métrica. Algunos ejemplos incluyen orig_txt-summ_ROUGE_temp1.0, que evalúa con ROUGE la similitud entre los textos originales y sus resúmenes generados; orig_txt-txts_BERT_temp1.0, que aplica BERTScore sobre los textos expandidos. Esta estructura modular permite no solo una consulta eficiente, sino también una carga selectiva de los datos necesarios para cada tipo de visualización o análisis posterior.

| **Hoja de Excel** | **Métrica** | **Texto de Referencia** | **Texto Comparado** | **Descripción** |
|---|---|---|---|---|
| *orig_txt-summ_ROUGE_temp1.0* | ROUGE | original_text | summary_* | Evalúa la calidad de los resúmenes generados comparándolos con el texto original. |
| *orig_txt-txts_ROUGE_temp1.0* | ROUGE | original_text | new_text_* | Evalúa los textos expandidos respecto al texto original. |
| *orig_sum-summ_ROUGE_temp1.0* | ROUGE | original_summary | summary_* | Evalúa los resúmenes generados respecto al resumen humano original. |
| *orig_txt-summ_BERT_temp1.0* | BERTScore | original_text | summary_* | Compara semánticamente el texto original con los resúmenes generados. |
| *orig_txt-txts_BERT_temp1.0* | BERTScore | original_text | new_text_* | Compara semánticamente el texto original con los textos expandidos. |
| *orig_sum-summ_BERT_temp1.0* | BERTScore | original_summary | summary_* | Compara semánticamente el resumen humano con los resúmenes generados. |

Tabla 3: *Estructura del archivo Excel generado en el proceso de evaluación automática.*

## 3.3 EVALUACIÓN DE LA RETENCIÓN SEMÁNTICA A TRAVÉS DE TEST AUTOMATIZADOS

Tras haber generado múltiples versiones de los textos originales mediante ciclos iterativos de resumen y expansión, se plantea la necesidad de evaluar en qué medida se conserva la información a lo largo del proceso. Para abordar esta cuestión de forma sistemática y cuantificable, se diseñó un sistema de evaluación automatizado basado en preguntas tipo test.

Este enfoque parte de una idea sencilla pero eficaz: si un modelo de lenguaje puede responder correctamente a una batería de preguntas sobre un texto original, ¿sigue siendo capaz de hacerlo tras haberlo reformulado en una, cinco o diez iteraciones? Al comparar las respuestas del modelo ante distintas versiones de un mismo contenido, se puede observar el grado de retención semántica y fidelidad informativa tras cada transformación.

La evaluación se estructura en tres etapas: primero, se generan automáticamente diez preguntas de opción múltiple a partir del texto original; en segundo lugar, esas preguntas se aplican a cada iteración del texto generado por el modelo (original, resumen expandido 1, 5 y 10); y finalmente, se analizan las respuestas ofrecidas por el modelo para comprobar si coinciden con las respuestas correctas definidas durante la fase de generación.

Este procedimiento ofrece una forma novedosa y escalable de evaluar modelos LLM más allá de métricas tradicionales como ROUGE [19] o BERTScore [20], al requerir que el modelo demuestre comprensión y conservación de la información a través del lenguaje natural.

### 3.3.1 Generación de preguntas tipo test

**Generación de preguntas con OpenAI**

El primer paso en el sistema de evaluación consiste en generar automáticamente una batería de preguntas tipo test basadas en el contenido original de los textos. Para ello, se implementó ***"generación_preguntas_test_gpt4.py"*** [1] que procesa cada entrada del corpus y construye una solicitud estructurada para ser enviada por lotes (batch) a la API de un modelo LLM.

Este proceso se apoya en el formato JSONL (JSON Lines), ampliamente utilizado en contextos de inferencia masiva debido a su simplicidad y eficiencia para representar múltiples objetos JSON en un único archivo de texto plano. Cada línea de este archivo contiene una petición independiente que puede ser procesada por el modelo en paralelo.

El sistema está diseñado para enviar solicitudes de forma masiva a la ruta /v1/chat/completions, utilizando una estructura estándar de tipo POST. Esta configuración permite un procesamiento eficiente de múltiples entradas en paralelo, aprovechando las capacidades de inferencia del modelo a gran escala.

Cada solicitud contiene varios elementos clave. En primer lugar, se incluye un custom_id, que actúa como identificador único para cada texto del corpus, permitiendo su trazabilidad a lo largo del proceso de evaluación.

Además, se incorpora un mensaje del sistema (system) que instruye explícitamente al modelo para que genere un total de 10 preguntas tipo test. Cada pregunta debe contar con cuatro opciones posibles (A, B, C y D), y únicamente una de ellas debe marcarse como correcta.

El mensaje del usuario (user) contiene el texto original que sirve como base para la generación de las preguntas. Este texto es el punto de partida sobre el cual el modelo debe razonar para construir las cuestiones planteadas.

Por último, se define un response_format basado en una estructura json_schema, que especifica cómo debe organizarse la respuesta del modelo. Esta debe consistir en un objeto con una

propiedad questions, dentro de la cual se encuentra una lista de preguntas correctamente formateadas, incluyendo sus opciones y la respuesta correcta asociada. Esta validación garantiza una salida estructurada y coherente para su posterior análisis automatizado.

Este diseño garantiza que el modelo no solo genere preguntas coherentes y variadas, sino que lo haga respetando una estructura JSON válida y verificable, lo cual es fundamental para su posterior procesamiento automático.

El script recorre uno a uno los textos presentes en el archivo Excel de entrada, y para cada uno escribe una línea en el archivo **"batch_input_test_questions_gpt4.jsonl"** [2], que se convierte en el lote de entrada que será enviado al modelo. Este archivo resultante puede ser utilizado directamente por servicios de inferencia en batch como OpenRouter, OpenAI Batch API, o entornos personalizados compatibles con el formato OpenAI API.

### Generación de preguntas con Gemini

En el caso de Gemini, el proceso difiere ligeramente en cuanto a la estructura de comunicación, aunque persigue el mismo objetivo: generar diez preguntas tipo test con formato controlado. Se emplea el modelo gemini-1.5-flash, con peticiones realizadas de forma secuencial (una por texto), respetando los límites de solicitudes por minuto definidos por la API.

El script correspondiente **"generación_preguntas_test_gemini.py"** [1] lee los textos originales desde un archivo Excel y genera las preguntas utilizando un prompt cuidadosamente diseñado que instruye al modelo para mantener un formato estructurado.

Este enfoque permite mantener un control total sobre la consistencia estructural de las preguntas generadas, asegurando su validez para posteriores etapas de testeo. La escritura incremental también garantiza la posibilidad de reanudar el proceso sin duplicar resultados ya procesados.

### Diferencias clave respecto a la generación con GPT-4

Aunque el objetivo y el formato final son equivalentes, el sistema de generación de preguntas con Gemini presenta importantes diferencias en su funcionamiento respecto al diseñado para GPT-4:

| Aspecto | GPT-4 (OpenAI Batch API) | Gemini (Google Generative AI) |
|---|---|---|
| **Tipo de API** | chat/completions con estructura por roles | generate_content con prompt plano |
| **Estructura de entrada** | JSONL con mensajes system y user diferenciados | Prompt textual completo embebido en una sola cadena |
| **Validación de respuesta** | Validación automática mediante json_schema | Validación manual por script (reformat_questions) |
| **Formato de salida** | JSON estructurado directamente por el modelo | Texto plano parseado para reconstrucción en JSON |
| **Modo de ejecución** | Procesamiento masivo por lotes (batch input completo) | Procesamiento secuencial con control de RPM |
| **Fiabilidad estructural** | Alta, controlada por OpenAI | Moderada, dependiente del cumplimiento del prompt |

| Aspecto | GPT-4 (OpenAI Batch API) | Gemini (Google Generative AI) |
|---|---|---|
| **Escalabilidad** | Muy alta | Limitada por RPM (60/minuto en cuentas estándar) |

*Tabla 4: Comparativa entre los procesos de generación de preguntas tipo test con GPT-4 y Gemini*

### 3.3.2 Procesamiento de las respuestas generadas

Una vez completada la generación de preguntas tipo test por parte de los modelos LLM, es necesario transformar sus salidas en un formato tabular reutilizable que permita su análisis y evaluación en fases posteriores del experimento. A continuación, se detallan los procesos específicos seguidos para cada modelo:

**Procesamiento de resultados generados con GPT-4**

El modelo GPT-4, al recibir las solicitudes en formato JSONL mediante la API de OpenAI, devuelve un archivo de salida igualmente estructurado línea a línea: **"batch_output_test_questions_gpt4.jsonl"** [2]. Cada entrada contiene la respuesta a una solicitud individual, organizada en el campo **response.body.choices[0].message.content**, donde se encuentra un bloque JSON con las diez preguntas de opción múltiple, sus respectivas opciones (A–D) y la respuesta correcta.

**Procesamiento de resultados generados por Gemini**

Una vez ejecutado el script de generación de preguntas con Gemini, el modelo devuelve las respuestas directamente en formato estructurado, utilizando un archivo denominado **"answer_input_gemini.jsonl"** [2]. Este archivo sigue la convención de JSONL, donde cada línea representa un objeto JSON independiente que contiene dos campos clave: **id**, que actúa como identificador único del texto original y permite rastrear el origen de cada conjunto de preguntas, y **questions_output**, una lista de objetos JSON con la estructura completa de cada pregunta, incluyendo enunciado, opciones y respuesta correcta. A diferencia del sistema empleado con la API de OpenAI, donde las preguntas se generaban en un campo de texto con contenido JSON serializado, en el caso de Gemini las preguntas se almacenan directamente en formato estructurado. Esta diferencia reduce significativamente la complejidad del proceso de extracción, eliminando la necesidad de deserialización manual y facilitando el análisis automático de las salidas generadas.

Para garantizar la coherencia estructural de estas preguntas, se utiliza la función ***"reformat_questions ()"***, en el script previo, que transforma la respuesta cruda generada por el modelo Gemini (texto plano) en una lista de objetos JSON.

Gracias a esta estructura, el contenido es inmediatamente utilizable para fases posteriores del experimento, como la evaluación del rendimiento del modelo al responder estas preguntas tras sucesivas iteraciones de resumen o expansión.

Este diseño elimina la necesidad de validar el formato mediante esquemas JSON externos o de realizar deserializaciones adicionales, y permite trabajar directamente con objetos Python (dict, list) para su análisis. Asimismo, la escritura incremental en el archivo JSONL facilita la reanudación del proceso en caso de interrupciones, asegurando que no se repliquen entradas ya procesadas.

Este archivo, estructurado y validado durante su generación, constituye la **base de datos de referencia** para las preguntas asociadas a cada texto del corpus. En fases posteriores, servirá para

comparar las respuestas generadas por modelos LLM sobre los textos modificados con las respuestas correctas generadas en este paso inicial.

### 3.3.3 Extracción de preguntas y almacenamiento estructurado

**Procesamiento de respuestas generadas por OpenAi**

Una vez completado el envío batch a la API de OpenAI y recibidas las respuestas generadas por el modelo, el siguiente paso consiste en extraer las preguntas en formato estructurado para su análisis y evaluación. Para ello, se diseñó "**json_output_to_excel_gpt4.py"** [1] encargado de transformar las respuestas contenidas en el archivo **"batch_output_test_questions_gemini. jsonl"** [2] en un archivo Excel fácilmente interpretable.

El archivo de entrada contiene una respuesta por línea, donde cada línea es un objeto JSON que encapsula, entre otros campos, el contenido generado por el modelo bajo **response.body.choices[0].message.content**. Este campo contiene una cadena JSON serializada, que representa una lista de 10 preguntas con sus respectivas opciones y respuestas correctas.

El script abre el archivo **"batch_output_test_questions_gemini. jsonl"** [2] línea por línea, deserializando cada entrada mediante ***"json. loads()"*** y almacenando los resultados en una lista de diccionarios.

Para cada entrada del archivo JSONL, el sistema extrae dos elementos fundamentales. En primer lugar, el custom_id, que actúa como identificador único vinculado al texto original y permite relacionar de forma fiable las preguntas generadas con las respuestas del modelo en sus distintas versiones (original, resumida o expandida). En segundo lugar, se obtiene la lista de preguntas mediante la deserialización del campo content, que contiene la cadena JSON generada por el modelo. En caso de que este proceso genere algún error —por ejemplo, debido a un formato incorrecto o a caracteres inválidos—, el sistema captura la excepción correspondiente, lo que evita que el fallo de una entrada interrumpa el procesamiento completo del conjunto de datos.

Cada conjunto de 10 preguntas se transforma en una única fila del DataFrame, lo que permite una representación compacta y estructurada de la información. En esta organización, las columnas Q1, Q2, ..., Q10 almacenan los enunciados de las preguntas, mientras que las columnas Options_1, Options_2, ..., Options_10 contienen las cuatro opciones posibles (A, B, C y D) para cada pregunta, concatenadas en una única cadena de texto. Finalmente, las columnas Correct_Option_1, ..., Correct_Option_10 especifican cuál es la opción correcta correspondiente a cada pregunta. Este diseño proporciona una estructura clara y directa que facilita tanto el acceso individual a las preguntas como su integración automatizada en la fase de evaluación del modelo.

Una vez procesadas todas las líneas, el script exporta los datos a un archivo Excel. Este archivo actúa como base de datos de preguntas generadas, asociadas a los textos del corpus original. El resultado del proceso de extracción se almacena en un archivo Excel denominado **"questions_gpt4.xlsx"** [2]. Este archivo contiene, para cada texto original del experimento, las preguntas de opción múltiple generadas automáticamente por el modelo de lenguaje. Cada fila del archivo corresponde a un único texto y se identifica mediante un id único (custom_id).

Las columnas están organizadas de la siguiente manera:

| Columna | Descripción |
|---|---|
| id | Identificador del texto original, heredado del campo custom_id. |
| Q1, Q2, ... | Enunciado de la pregunta número 1, 2, etc. generada por el modelo. |

| Columna | Descripción |
|---|---|
| Options_1, ... | Opciones A, B, C y D asociadas a cada pregunta, separadas por comas. |
| Correct_Option_1… | Letra correspondiente a la respuesta correcta de cada pregunta (A–D). |

*Tabla 5: Formato de los excel de salida de "questions_gemini" y "questions_gpt4"* [2]

Este archivo es fundamental para la siguiente etapa del experimento, en la que se evaluará si los modelos LLM (en sus distintas iteraciones de resumen y expansión) son capaces de responder correctamente a las preguntas generadas a partir del contenido original. Gracias a su formato estructurado, el archivo permite un emparejamiento automático entre las preguntas y las respuestas proporcionadas por los modelos en la fase de benchmarking.

**Procesamiento de respuestas generadas por Gemini**

En el caso del modelo Gemini, las respuestas generadas se almacenan directamente en un archivo denominado **"output_test_questions_gemini.jsonl"** [2], utilizando una estructura nativa basada en formato JSON por línea. Cada entrada de este archivo incluye dos campos clave: un identificador único (id) asociado al texto original, y el campo questions_output, que contiene la lista de preguntas generadas junto con sus respectivas opciones y la respuesta correcta.

El proceso de transformación y estructuración de estas preguntas se desarrolla en dos fases, implementadas en el script **"json_output_to_excel_gemini.py"** [1]. En primer lugar, se realiza la conversión del archivo JSONL a CSV recorriendo línea por línea, extrayendo los campos id, question, options (concatenadas con separadores) y correct_answer, que se almacenan en un archivo intermedio. Posteriormente, se reorganiza esta información en formato tabular en un archivo Excel, agrupando las preguntas por identificador y reestructurándolas en columnas horizontales del tipo Q1, Options_1, Correct_Option_1, ..., hasta Q10, Options_10, Correct_Option_10.

El resultado final se exporta al archivo **"preguntas_gemini_.xlsx"** [2], que constituye la base de datos de referencia con las preguntas generadas a partir de los textos procesados por el modelo Gemini. Al igual que en el caso del modelo GPT-4, este archivo será utilizado en las fases posteriores de evaluación para medir el rendimiento de los modelos al responder sobre textos modificados.

### 3.3.4 Aplicación de los test a los textos iterados

**Evaluación de respuestas generadas por OpenAI**

Una vez generadas y almacenadas las preguntas de opción múltiple, el siguiente paso fue construir un benchmark, que permitiese evaluar automáticamente la capacidad de comprensión de los modelos LLM sobre las distintas versiones de los textos generados (original, resumido y expandido).

Para ello, se desarrolló **"benchmark_input_gpt4.py"** [1] que, por cada combinación de texto y pregunta, crea una solicitud estructurada en formato JSONL, apta para ser enviada como lote (batch) a través del sistema de inferencia de OpenAI.

El proceso de construcción del benchmark automatizado parte de dos fuentes principales de datos. Por un lado, el archivo "**preguntas_gtp4.xlsx**" [2] contiene las preguntas tipo test generadas a

partir del texto original, incluyendo tanto las opciones de respuesta como la correcta. Por otro lado, el archivo

**"resultados_bucle_resumen_expansion_gpt4.xlsx"** [2] que recoge los textos originales y las distintas iteraciones generadas mediante los procesos de resumen y expansión.

Antes de comenzar la evaluación, el sistema verifica que ambos archivos contengan las columnas necesarias para el emparejamiento y análisis posterior. Una vez completada esta validación, el script recorre uno a uno los textos disponibles —específicamente original_text, new_text_1_temp_1.0, new_text_5_temp_1.0 y new_text_10_temp_1.0— y los empareja con las preguntas correspondientes, utilizando como referencia el identificador único (id) asociado a cada entrada. Este emparejamiento garantiza la coherencia entre las variantes de texto y su batería específica de preguntas, lo que permite construir las solicitudes de evaluación de manera estructurada y consistente.

Este diseño permite controlar de forma estricta el contexto utilizado por el modelo, impidiéndole acceder a conocimiento externo, lo cual garantiza que la respuesta dependa exclusivamente de la información contenida en el texto proporcionado. Cada entrada se identifica de forma única mediante un custom_id que codifica el tipo de texto (original_text o new_text_*), el id del ejemplo, y la pregunta (Q1 a Q10). Esto permite vincular posteriormente cada respuesta generada con su pregunta original y texto de partida.

Finalmente, el conjunto de todas las entradas se guarda en un archivo denominado **"benchmark_input_answer_gpt4. jsonl"** [2] , que puede utilizarse para enviar miles de peticiones en paralelo a la API de OpenAI, facilitando así la evaluación masiva de la comprensión textual bajo condiciones controladas. Tras enviarlo haciendo uso del batch las respuestas se guardan en **"answer_output_gpt4.jsonl"** [2].

**Evaluación de respuestas generadas por Gemini**

Una vez generadas y almacenadas las preguntas tipo test, el siguiente paso consistió en construir un benchmark automatizado que permitiera evaluar la capacidad de comprensión de los modelos LLM (en este caso, Gemini) sobre las distintas versiones de los textos: original, resumida y expandida.

Para ello, se programó **"answer_input_gemini.py"** [1] encargado de generar todas las combinaciones posibles entre textos y preguntas. Este proceso parte de dos archivos clave: **"textos_gemini.xlsx"** [2], que contiene tanto el texto original como sus versiones generadas por resumen y expansión; y **"preguntas_gemini.xlsx"** [2], que incluye las diez preguntas generadas a partir del texto original, junto con sus opciones y la respuesta correcta.

El script verifica que ambos archivos contengan las columnas necesarias, como id, original_text, new_text_*, Qn, Options_n, entre otras. A continuación, recorre cuatro variantes del texto: el original, junto con las versiones generadas en la primera, quinta y décima iteración. Para cada combinación de texto y pregunta (de la Q1 a la Q10), se construye una entrada en formato JSONL. Esta entrada incluye un identificador único, denominado custom_id, que codifica el tipo de texto, el identificador del ejemplo y el número de la pregunta, así como un prompt estructurado. Este prompt presenta el texto y la pregunta correspondiente con sus opciones, además de una instrucción explícita al modelo para que responda únicamente con la letra de la opción correcta, sin recurrir a conocimiento externo.

Todas las entradas generadas se almacenan en un archivo denominado **"answer_input_gemini. jsonl"** [2], que está preparado para ser procesado por lotes mediante la API del modelo Gemini. Este enfoque permite realizar evaluaciones masivas de forma eficiente y estructurada.

Una vez preparado el archivo de evaluación, se utiliza un segundo script **"answer_gemini_output.py"** [1]para enviar cada entrada al modelo gemini-1.5-flash y recoger sus

respuestas. Este script incorpora varias funcionalidades críticas para garantizar la robustez del proceso. En primer lugar, evita la duplicación de entradas mediante la omisión automática de aquellas cuyo custom_id ya ha sido procesado y almacenado en el archivo de salida. En segundo lugar, controla la tasa de peticiones para no superar el límite de 60 solicitudes por minuto impuesto por la API.

Adicionalmente, el sistema incluye una función de normalización que aplica expresiones regulares sobre la respuesta devuelta por el modelo para extraer exclusivamente la letra seleccionada (A, B, C o D). También cuenta con mecanismos de gestión de errores, capaces de capturar respuestas inválidas, problemas de conexión o desviaciones en el formato esperado. Todos estos casos se registran en el archivo de salida, denominado **"answer_gemini_output. jsonl"** [2], donde cada línea contiene el custom_id, la opción seleccionada (answer) y el contenido completo de la respuesta original (raw).

A pesar de que el sistema fuerza al modelo a responder únicamente con una letra, en la práctica **no siempre se consigue este comportamiento de forma consistente**. En múltiples ocasiones, el modelo responde con frases explicativas o con observaciones del tipo:

"The answer cannot be determined based on the information provided in the text."

Esto indica que, aunque el prompt esté cuidadosamente diseñado para acotar la respuesta, el modelo sigue evaluando si considera que dispone de suficiente información antes de elegir una opción. En los casos en que no lo considera así, puede **rehusar dar una respuesta directa o simplemente no incluir ninguna letra reconocible**. Para manejar estas situaciones, el sistema implementa una política de fallback: si no se detecta una letra válida mediante expresiones regulares, se selecciona una opción aleatoria entre A y D, registrando este comportamiento. Al final del proceso, se calcula y reporta el porcentaje de respuestas generadas aleatoriamente, lo que permite cuantificar el grado de incertidumbre o ambigüedad del modelo ante las preguntas planteadas.

# 4.RESULTADOS

## 4.1 RESULTADOS DE LA EVALUACIÓN MEDIANTE MÉTRICAS

Para facilitar el análisis cuantitativo del rendimiento de los modelos LLM evaluados (GPT-4 Mini y Gemini 1.5 Flash), se realizó **"gráficas_comparativas_llm.py"** [1] que automatiza tanto el procesamiento de las métricas como la generación de visualizaciones interpretables.

En primer lugar, se estructuraron los resultados de evaluación generados en la fase anterior (almacenados en hojas Excel) en un formato tabular que permite representar la evolución métrica por iteración. Para ello, se implementó la función ***"transform_data_for_iterations()"***, que recorre las columnas de métricas codificadas por iteración (e.g., summary_1_temp_1.0-f1) y construye un nuevo DataFrame plano con las siguientes columnas: id, iteration, metric_type y value. Esta función es clave para analizar tendencias iterativas, ya que convierte el formato horizontal original en una estructura larga adecuada para graficar.

Con los datos transformados, se implementó la función ***"plot_combined_metrics()"*** que permite visualizar, para cada modelo y tipo de comparación (e.g., original_text vs summary_*), la evolución de las tres métricas principales: precisión, recall y F1. La gráfica resultante incluye la media por iteración y barras de error correspondientes a la desviación estándar, permitiendo identificar comportamientos de mejora o deterioro a lo largo de los ciclos. Gracias a esta visualización, es posible comparar con precisión cómo se comporta un modelo en tareas de compresión o expansión de texto a través del tiempo.

Además de las gráficas individuales por modelo, se incorporó una función adicional, ***"plot_model_comparison_bar_chart()"***, que permite comparar de forma directa los valores promedio entre modelos para iteraciones específicas (por ejemplo, 1, 5 y 10), usando gráficos de barras agrupadas. Estas gráficas proporcionan una perspectiva sintética de la **eficacia relativa entre modelos** en distintos momentos del experimento, ayudando a identificar cuál de ellos ofrece mayor fidelidad semántica o literal según el tipo de tarea.

### 4.1.1 Resultados métricos por iteración

Con el objetivo de evaluar el impacto de la generación iterativa sobre la fidelidad de los textos generados por los modelos, se realizó un análisis detallado de métricas de similitud léxica y semántica a lo largo de diez iteraciones. El estudio se llevó a cabo para dos modelos diferentes, **GPT-4 Mini** y **Gemini 1.5 Flash**, utilizando las métricas **ROUGE** [19] y **BERTScore** [20]. Las comparaciones se organizaron en tres escenarios clave: la similitud entre el texto original y las versiones expandidas, entre el texto original y los resúmenes generados por los modelos, y entre el resumen humano de referencia y los resúmenes generados automáticamente. En los tres casos, se explora cómo evolucionan las métricas de precisión, recall y F1-score a medida que se acumulan iteraciones, con el objetivo de identificar patrones de pérdida de calidad o de estabilidad semántica.

**Resultados “orig_sum-summ”: Comparación con resumen humano**

En este escenario, se analizó cómo se degradan los resúmenes generados por los modelos LLM conforme se encadenan múltiples iteraciones de resumen. Las gráficas muestran la evolución de las, tomando como referencia el resumen humano original.

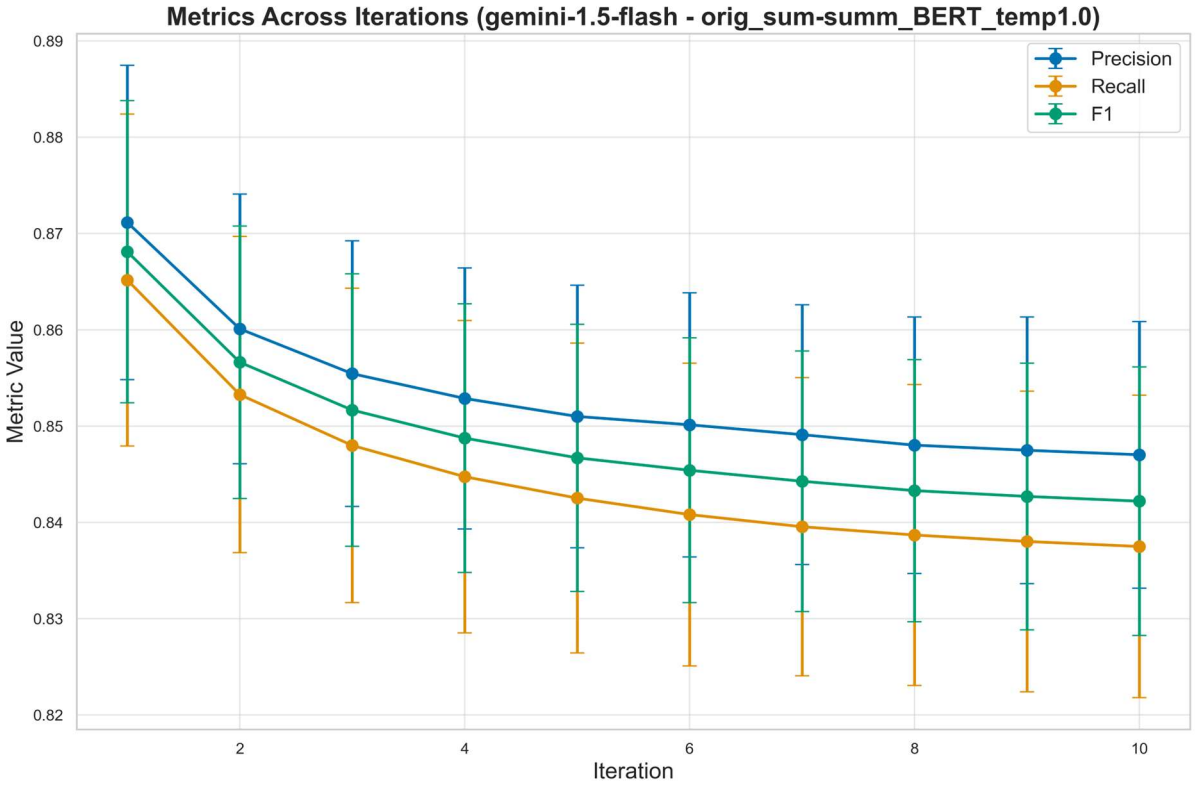


*Figura 5: Evolución de BERTScore (Precisión, Recall y F1) a lo largo de iteraciones sucesivas para el modelo Gemini 1.5 Flash (orig_sum-summ)*

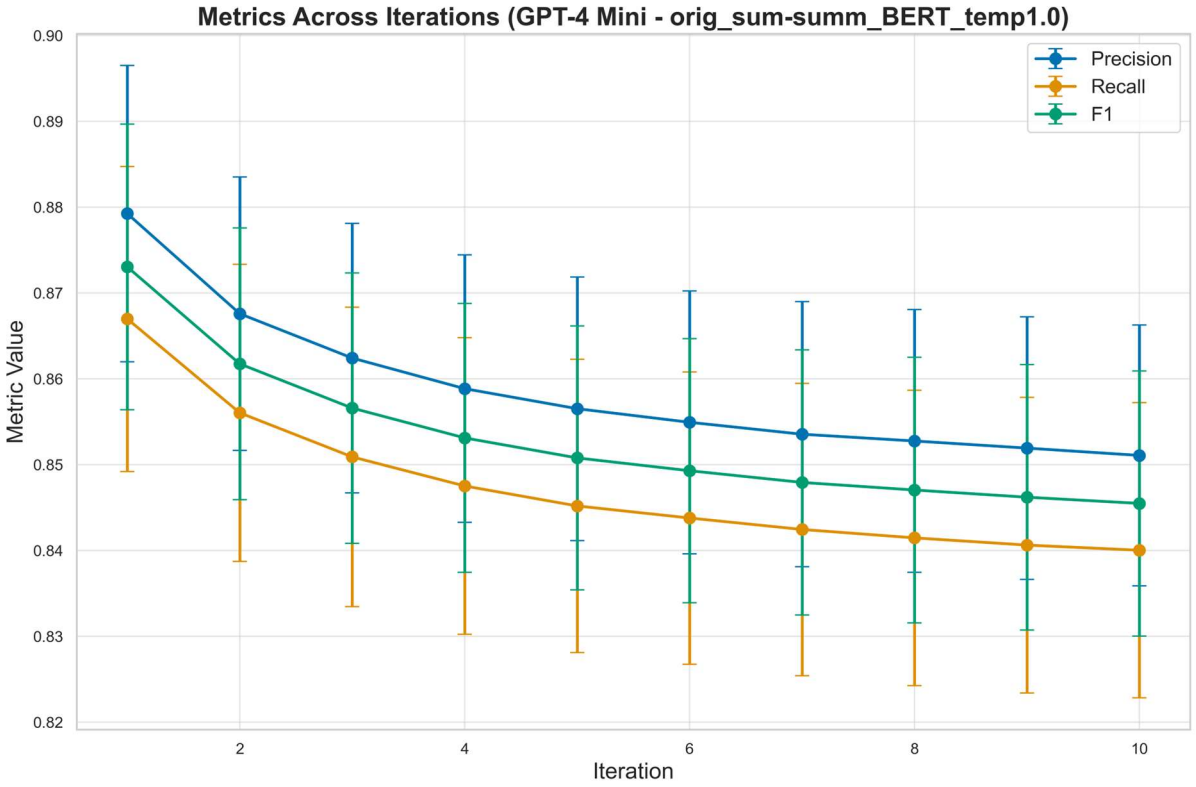


*Figura 6: Evolución de BERTScore (Precisión, Recall y F1) a lo largo de iteraciones sucesivas para el modelo GPT-4 Mini (orig_sum-summ)*

Tanto en el caso de Gemini como en el de GPT-4 Mini, se observa una degradación progresiva en las tres métricas de BERTScore (precisión, recall y F1). No obstante, esta degradación es **más acusada en las primeras iteraciones**, tras las cuales la pendiente se reduce, lo que sugiere una **estabilización parcial del contenido semántico** a partir de la mitad del proceso (iteración 5-6). En términos absolutos, **GPT-4 Mini** parte de valores ligeramente más altos que Gemini, lo que indica una mejor alineación semántica inicial con el resumen humano. Sin embargo, ambos modelos muestran **una tendencia muy similar** en cuanto a pérdida de significado acumulada.

En los dos modelos, la **precisión es sistemáticamente superior al recall**, lo que sugiere que tienden a preservar mejor fragmentos concretos del resumen original, aunque sacrificando parte de la cobertura global de la información.

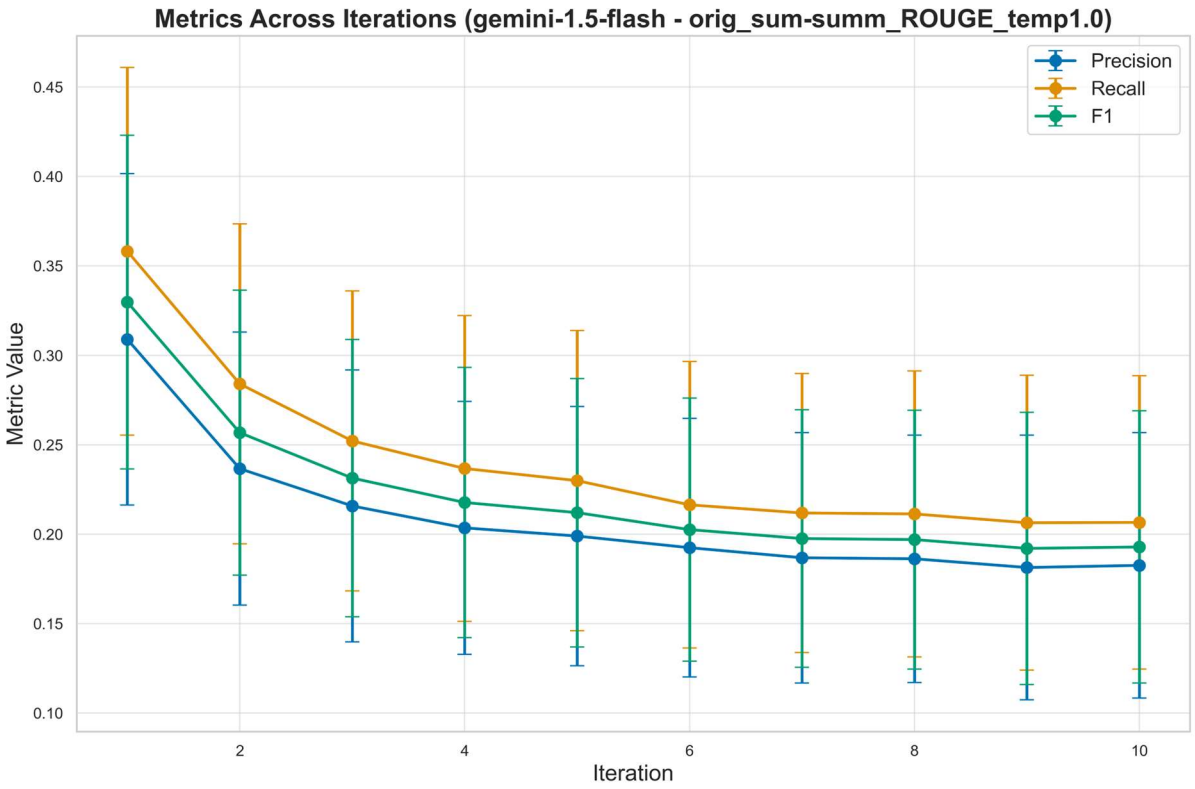


*Figura 7: Evolución de Rouge (Precisión, Recall y F1) a lo largo de iteraciones sucesivas para el modelo Gemini 1.5 Flash (orig_sum-summ)*

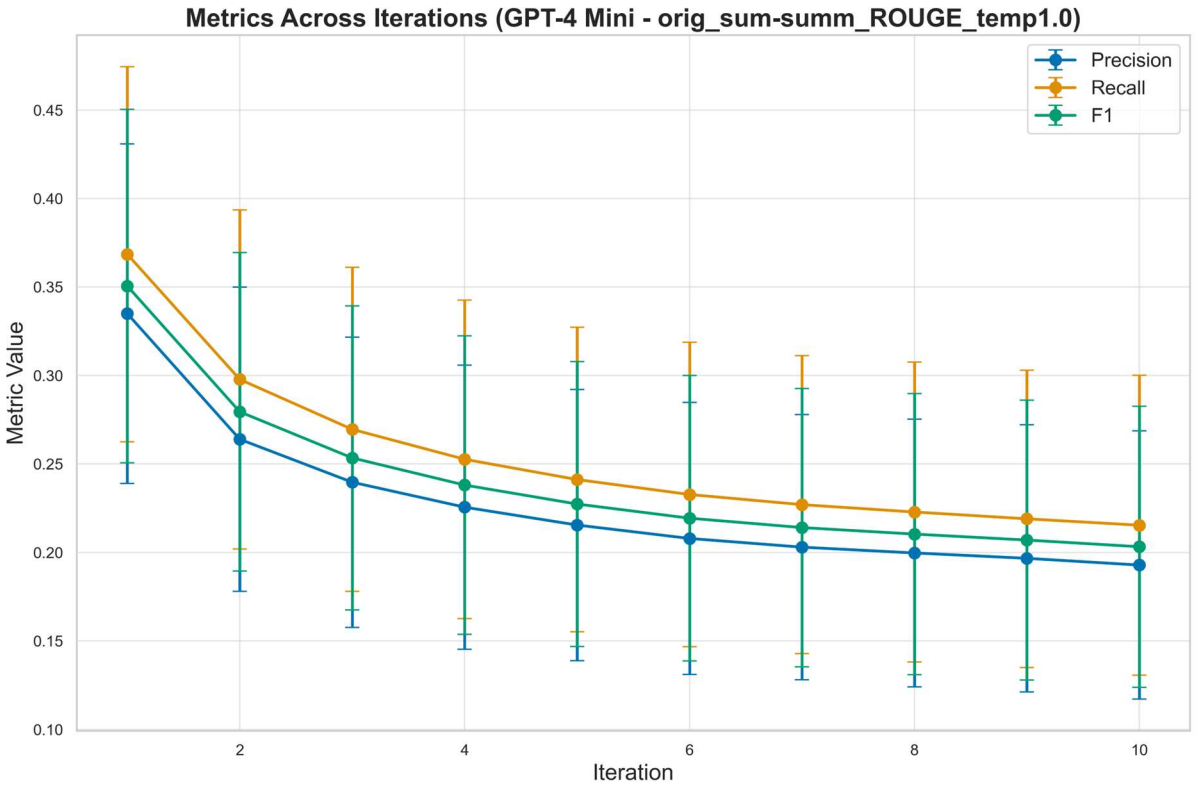


*Figura 8: Evolución de Rouge (Precisión, Recall y F1) a lo largo de iteraciones sucesivas para el modelo GPT-4 Mini (orig_sum-summ)*

En las gráficas de ROUGE se aprecia un patrón similar de **descenso pronunciado al inicio**, seguido de una meseta en las últimas iteraciones. Aquí la pérdida es más drástica que en BERTScore, lo que evidencia que la **fidelidad léxica** se ve más afectada que la semántica por el proceso iterativo.

En valores absolutos, **GPT-4 Mini vuelve a destacar en la iteración inicial**, pero sufre un descenso más abrupto, mientras **que Gemini muestra una evolución más suave y homogénea**. Esto podría interpretarse como una **mayor robustez estructural** por parte de Gemini cuando se enfrenta a múltiples ciclos de resumen.

**Resultados "txt-txts": Comparación con texto original**

Este escenario evalúa la evolución de los textos generados tras varias iteraciones de expansión, comparándolos directamente con el texto original. El objetivo es medir la fidelidad léxica y semántica a lo largo del proceso de ampliación recursiva, y determinar hasta qué punto los modelos son capaces de mantener el contenido original sin derivar significativamente.

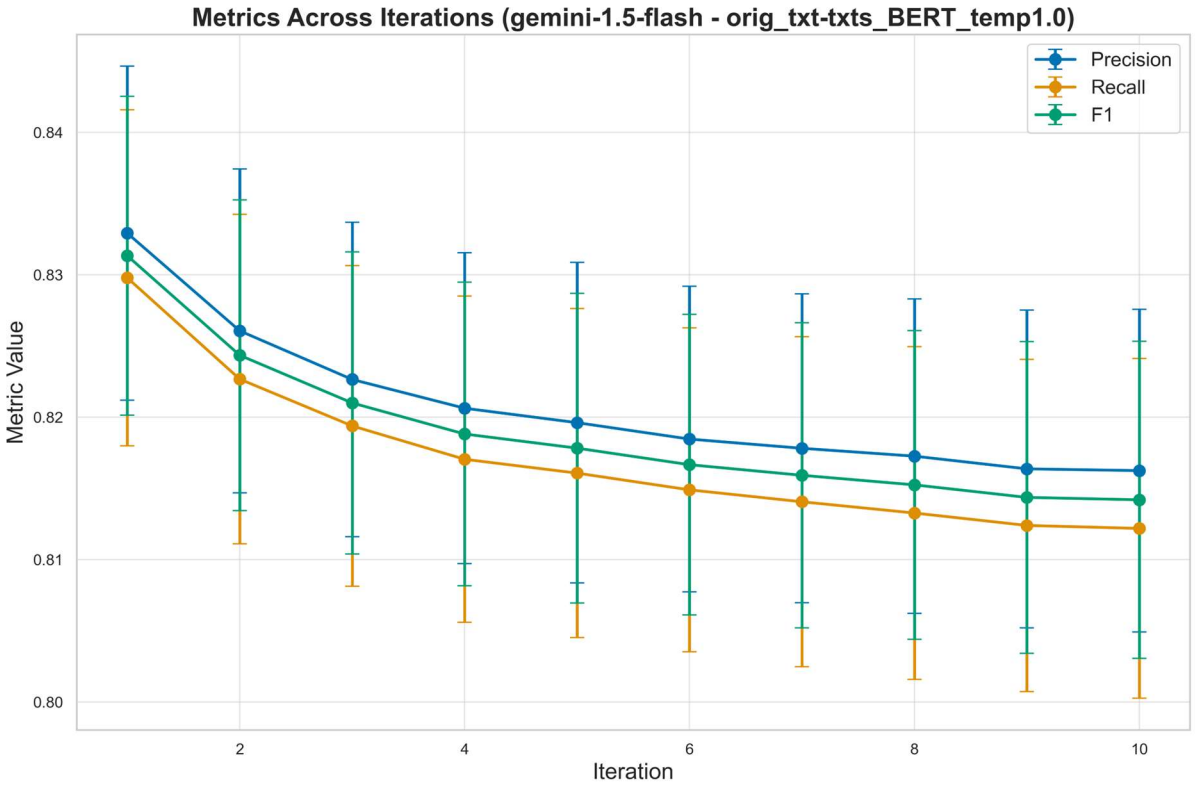


*Figura 9: Evolución de BERTScore (Precisión, Recall y F1) a lo largo de iteraciones sucesivas para el modelo Gemini 1.5 Flash (orig_txt-txts)*

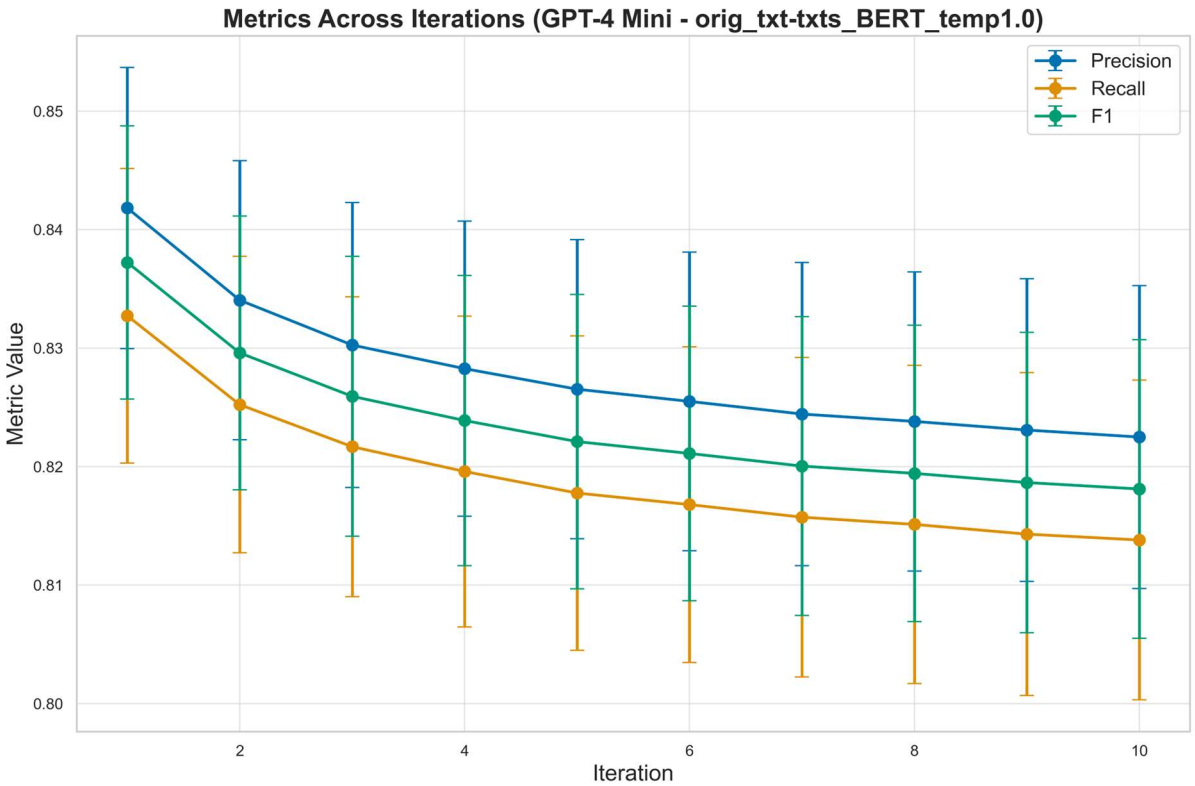


*Figura 10: Evolución de BERTScore (Precisión, Recall y F1) a lo largo de iteraciones sucesivas para el modelo GPT-4 Mini (orig_txt-txts)*

En las gráficas de BERTScore se aprecia una degradación continua en las métricas de precisión, recall y F1 para ambos modelos. GPT-4 Mini parte de valores ligeramente superiores a Gemini, con una F1 cercana a 0.84, mientras que Gemini comienza algo por debajo (~0.83). En ambos casos, la pérdida de calidad es notable durante las primeras iteraciones, estabilizándose en torno a la iteración 6.

Se observa de nuevo que la precisión se mantiene sistemáticamente por encima del recall, lo que indica que los modelos tienden a conservar expresiones específicas del texto original, aunque pierden parte de la cobertura semántica global. Este comportamiento sugiere que, en procesos de expansión, los modelos priorizan **la retención de fragmentos literales frente a una reexpresión más integral del contenido.**

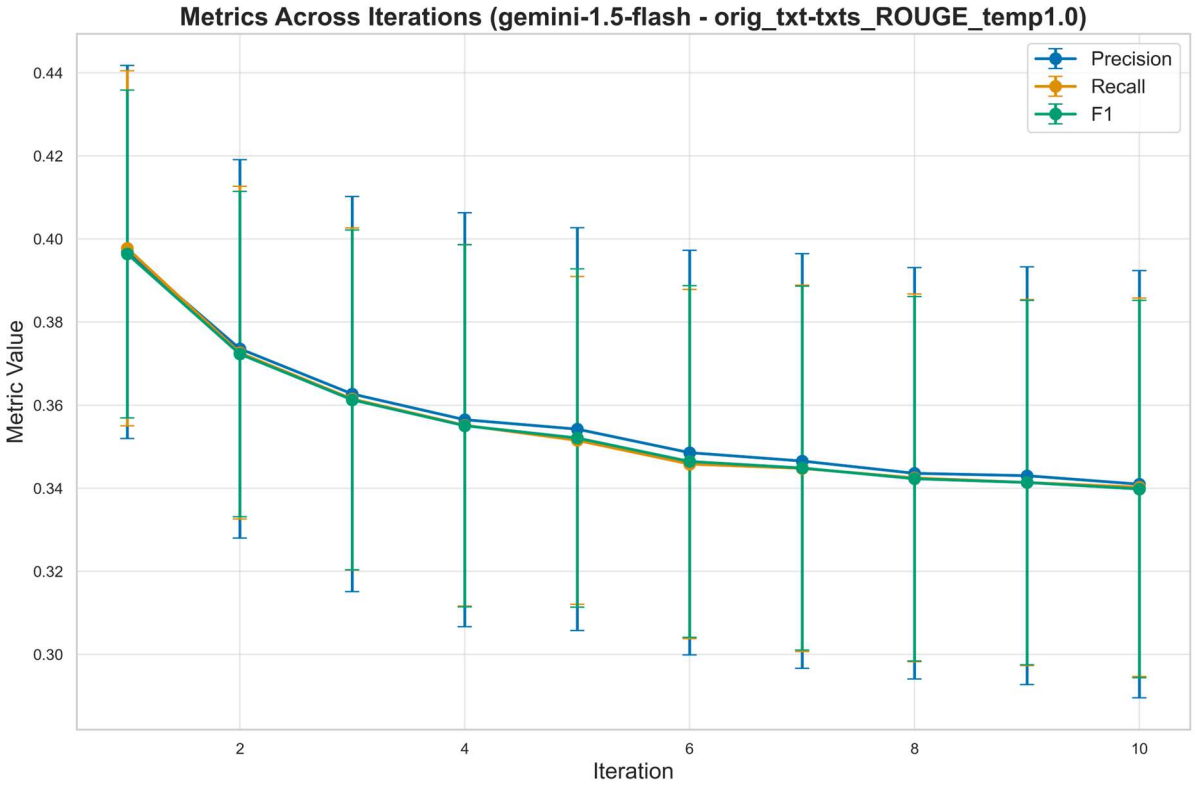


*Figura 11: Evolución de Rouge (Precisión, Recall y F1) a lo largo de iteraciones sucesivas para el modelo Gemini 1.5 Flash (orig_txt-txts)*

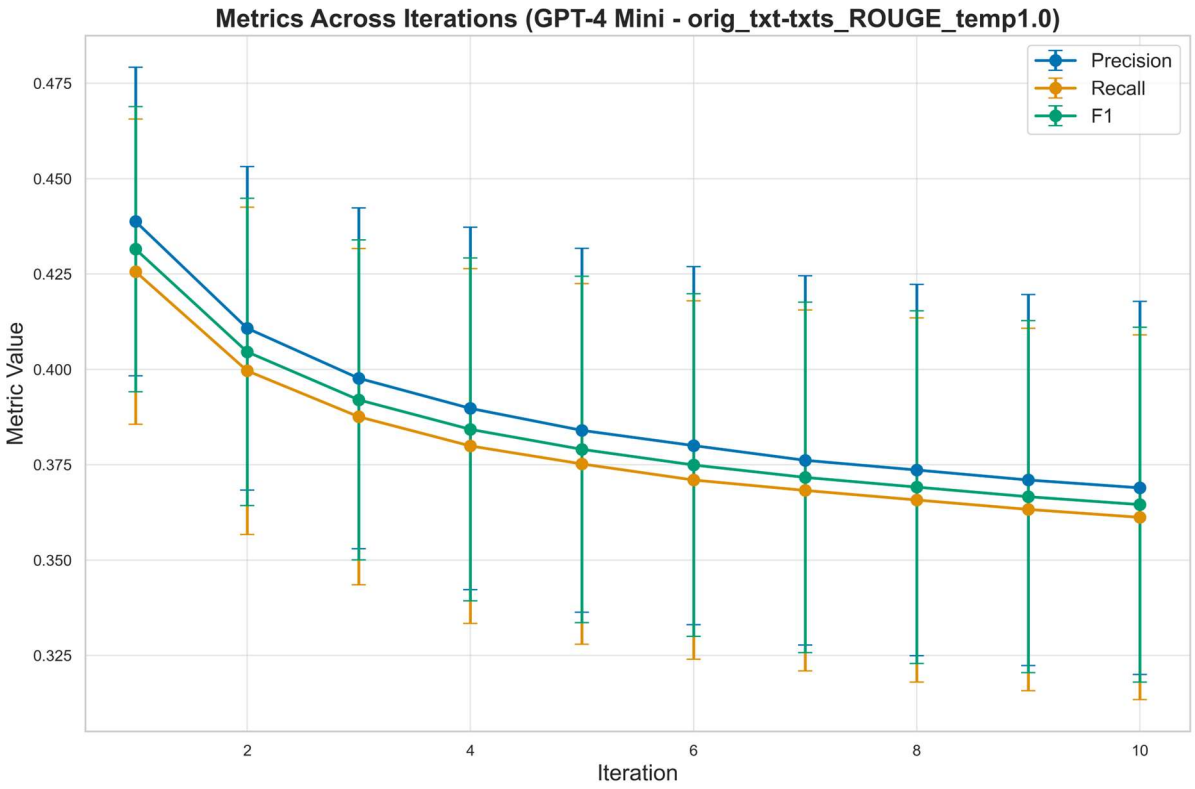


*Figura 12: Evolución de Rouge (Precisión, Recall y F1) a lo largo de iteraciones sucesivas para el modelo GPT-4 Mini (orig_txt-txts)*

Las gráficas de ROUGE revelan una degradación considerable de la similitud léxica entre el texto original y los resúmenes generados. En el caso de GPT-4 Mini, los valores parten en torno a 0.43, mientras que Gemini 1.5 Flash comienza ligeramente por debajo, en aproximadamente 0.40. En ambos casos, se observa una caída sostenida y relativamente lineal a lo largo de las iteraciones, que termina con puntuaciones cercanas a 0.36 y 0.30, respectivamente.

Este descenso progresivo pone de manifiesto que los modelos tienden a alejarse cada vez más del contenido léxico original, especialmente en las primeras fases del proceso. La mayor caída en el caso de GPT-4 Mini sugiere que su estrategia de compresión **introduce alteraciones significativas desde las primeras iteraciones**. Gemini, por su parte, muestra **un descenso algo más moderado**, aunque sus valores iniciales ya eran más bajos, lo que sugiere una mayor estabilidad estructural, pero a costa de un rendimiento léxico base más limitado.

**Resultados txt-summ: Comparación de resumenes con texto original**

En este escenario se evalúa hasta qué punto los resúmenes generados por los modelos LLM, tras sucesivas iteraciones de compresión, mantienen la fidelidad léxica y semántica con respecto al texto original completo. Este análisis permite determinar el grado de condensación efectiva sin pérdida de información crítica.

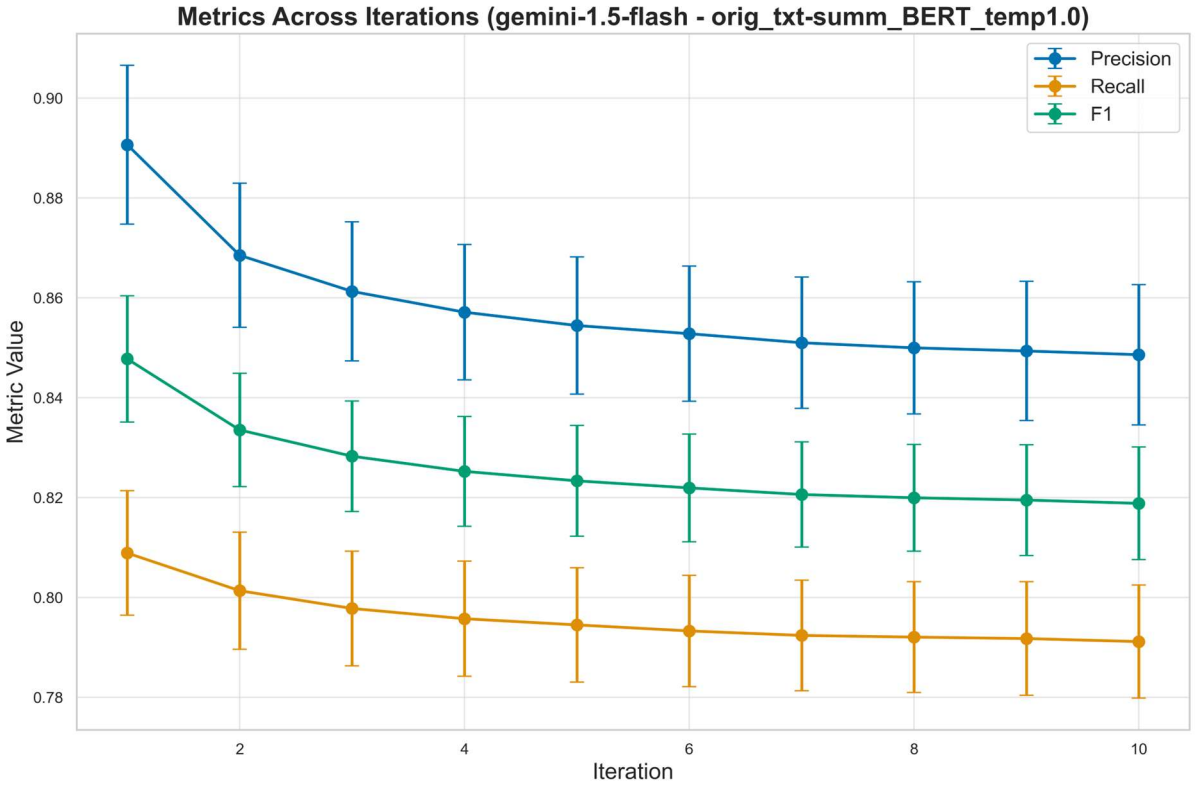


*Figura 13: Evolución de BERTScore (Precisión, Recall y F1) a lo largo de iteraciones sucesivas para el modelo Gemini 1.5 Flash (orig_txt-summ)*

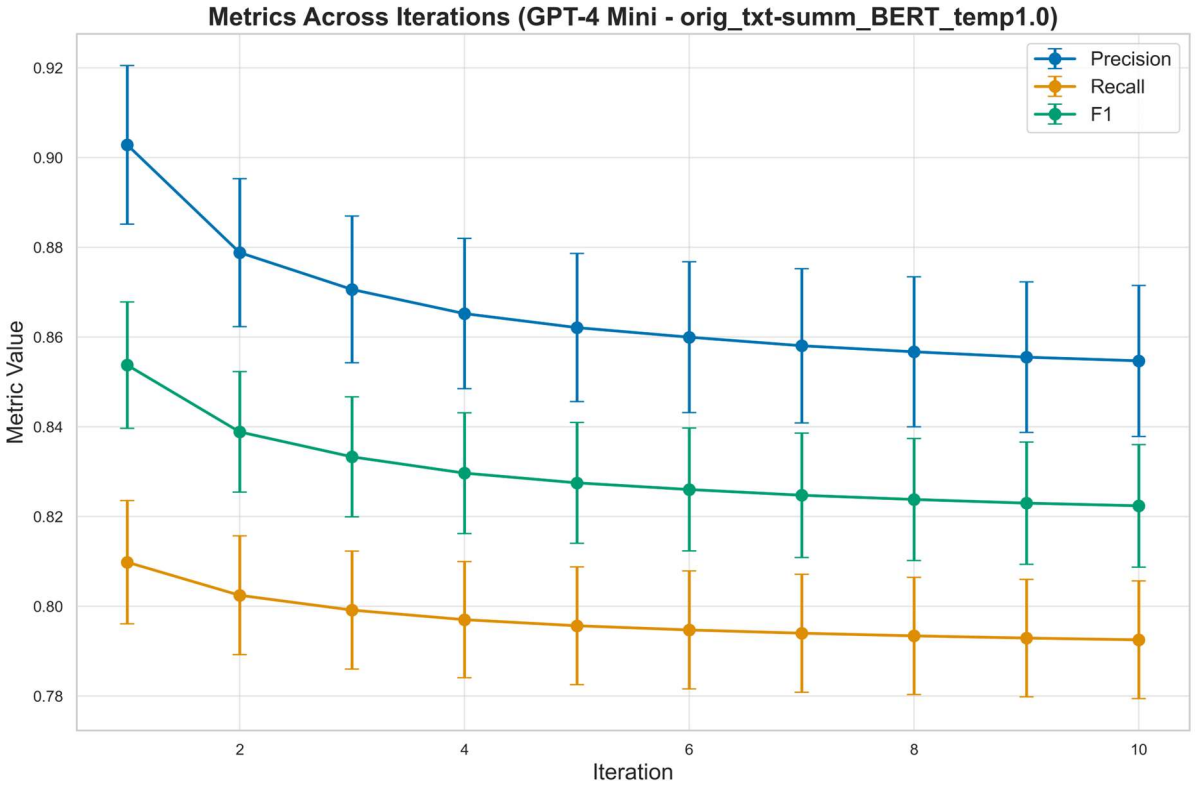


*Figura 14: Evolución de BERTScore (Precisión, Recall y F1) a lo largo de iteraciones sucesivas para el modelo GPT-4 Mini (orig_txt-summ)*

Las gráficas correspondientes a BERTScore muestran una degradación progresiva en las métricas de precisión, recall y F1 a medida que se encadenan iteraciones de resumen. Ambos modelos parten de valores elevados en F1 alrededor de 0.85 para GPT-4 Mini y ligeramente inferiores para Gemini, lo que indica que los primeros resúmenes aún conservan un alto grado de alineación semántica con el texto original.

Sin embargo, la pérdida de calidad semántica es especialmente marcada en las primeras iteraciones, estabilizándose a partir de la iteración 5 o 6, lo que sugiere que gran parte de la información clave se pierde tempranamente. Esta estabilización podría indicar que, tras eliminar los contenidos más redundantes o superficiales, los modelos tienden a mantener un núcleo semántico más estable, aunque más limitado.

Como en otros escenarios, la precisión se mantiene sistemáticamente por encima del recall. Esto revela un patrón interesante: los modelos son capaces de reproducir expresiones relevantes del original con cierta fidelidad, pero omiten fragmentos esenciales que afectan a la cobertura global del significado. Es decir, se aferran a elementos concretos, pero sacrifican contexto.

GPT-4 Mini presenta valores ligeramente superiores a Gemini en todas las iteraciones, lo que sugiere una mayor capacidad de este modelo para retener información semántica incluso tras múltiples resúmenes. Este comportamiento puede deberse a una mejor representación interna de las relaciones contextuales del texto, lo que lo convierte en una opción preferente en tareas que requieran compresión sin pérdida significativa de sentido.

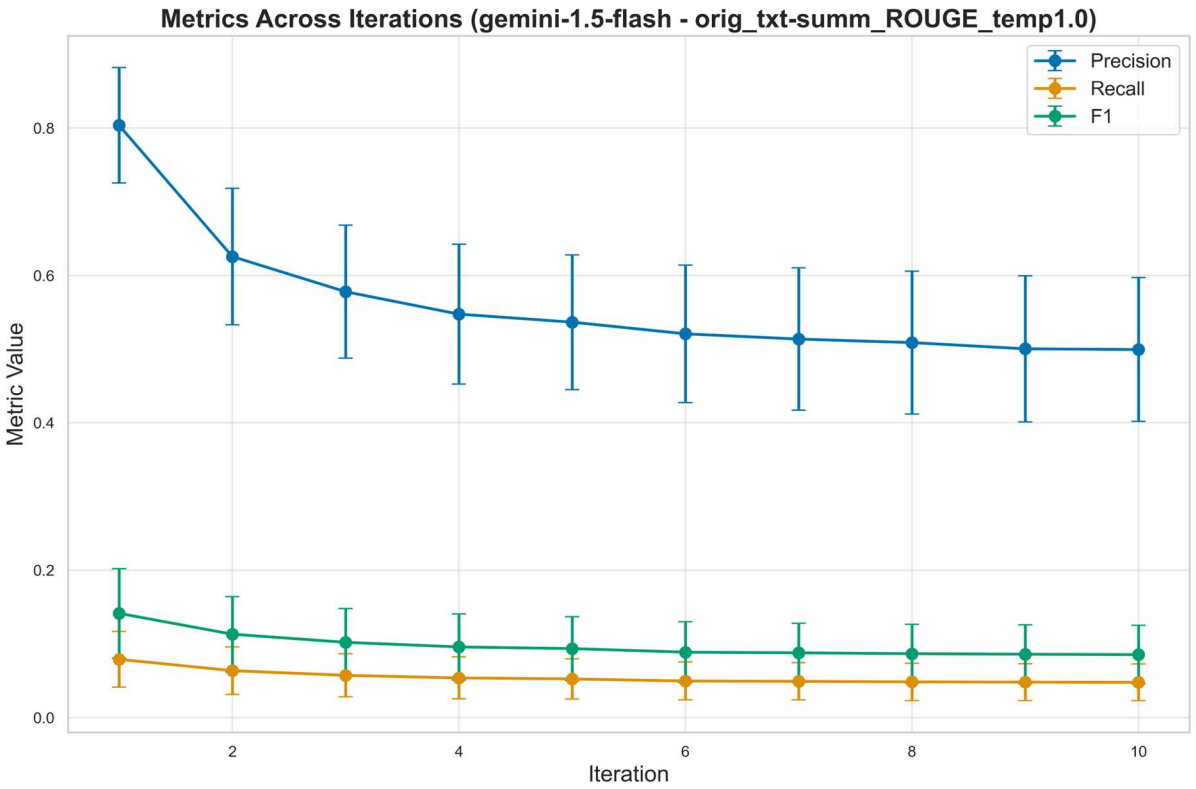


*Figura 15: Evolución de Rouge (Precisión, Recall y F1) a lo largo de iteraciones sucesivas para el modelo Gemini 1.5 Flash (orig_txt-summ)*

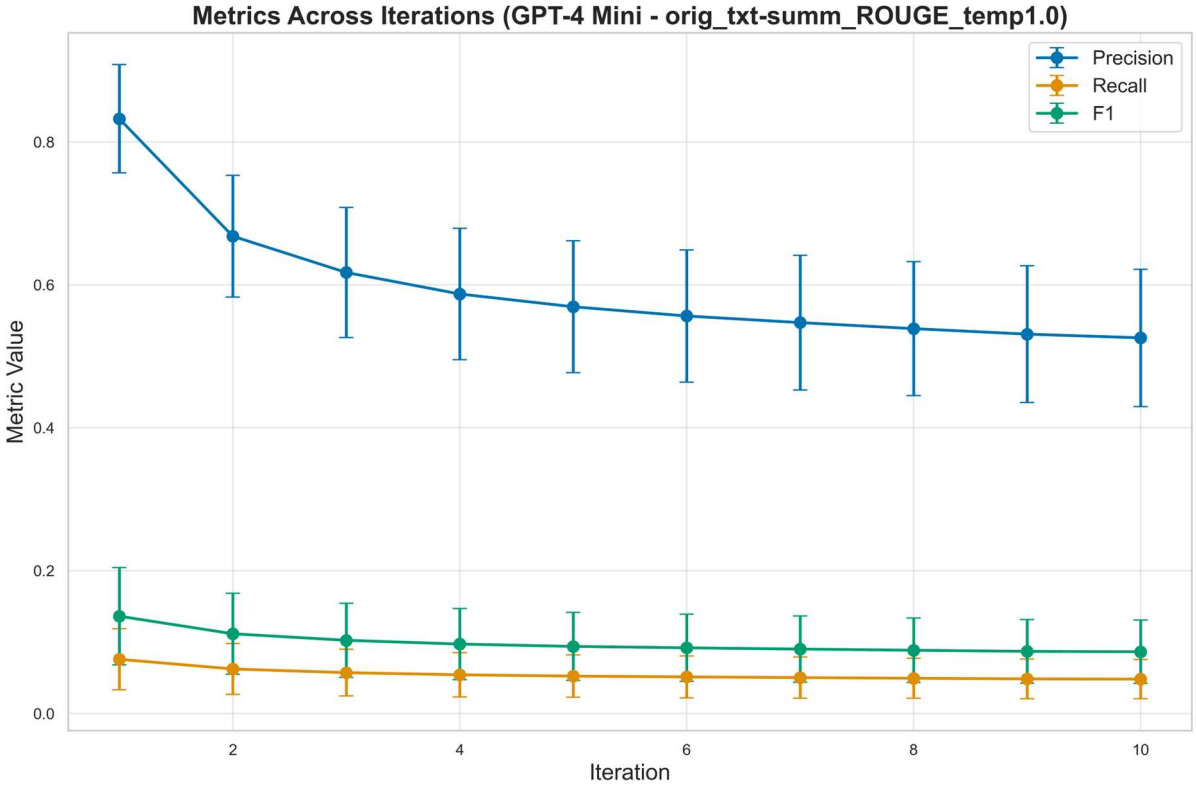


*Figura 16: Evolución de Rouge (Precisión, Recall y F1) a lo largo de iteraciones sucesivas para el modelo GPT-4 Mini (orig_txt-summ)*

Las gráficas de ROUGE-1 reflejan una pérdida todavía más acusada en términos léxicos. Ambos modelos comienzan con valores de precisión próximos a 0.80, pero sufren una caída rápida a partir de la primera iteración. En el caso de Gemini, el descenso inicial es especialmente brusco, cayendo cerca de 20 puntos porcentuales, lo que sugiere una pérdida drástica de coincidencias léxicas con respecto al texto original.

Este comportamiento podría explicarse por varios factores. En primer lugar, cabe destacar que los modelos tienden a emplear una estrategia de compresión agresiva en la primera iteración, especialmente cuando no se imponen restricciones explícitas sobre la longitud o la cobertura informativa. Esto provoca que se omitan bloques enteros de información superficial (nombres propios, adjetivos calificativos, detalles numéricos), que tienen un alto peso en las métricas basadas en n-gramas como ROUGE, pero que no siempre afectan a la coherencia semántica global. En consecuencia, la métrica penaliza duramente cualquier alteración en la superficie textual.

En segundo lugar, es probable que el **prompt inicial del resumen incentive una mayor reescritura o reformulación en la primera iteración**, lo que incrementa la distancia léxica respecto al texto de origen. Esta distancia puede no ser semánticamente problemática (como indican los mejores resultados en BERTScore), pero sí representa una pérdida sustancial en términos de solapamiento literal de palabras.

Además, este descenso abrupto sugiere que los modelos aplican sus transformaciones más radicales en las primeras fases del proceso, posiblemente porque consideran que en ese punto hay mayor redundancia que eliminar. A medida que las iteraciones avanzan, el contenido se vuelve más compacto y los cambios se reducen, lo que explica la meseta que aparece a partir de la tercera o cuarta iteración.

GPT-4 Mini presenta una evolución más estable, aunque también descendente, con un comportamiento similar al de Gemini, pero menos pronunciado. Esta diferencia puede deberse a un sesgo más conservador en su generación, que prioriza mantener estructuras sintácticas familiares y un mayor grado de literalidad. Aun así, en ambos casos los valores de recall son bajos en todo el proceso, lo que indica que los resúmenes pierden una porción significativa del contenido léxico original.

En resumen, la caída pronunciada en ROUGE-1 (especialmente en la primera iteración) responde tanto a las **estrategias de síntesis iniciales de los modelos como a la naturaleza de la propia métrica, altamente sensible a sustituciones léxicas y omisiones literales**. Aunque esta degradación no siempre implica una pérdida de calidad semántica, sí señala una considerable **transformación superficial** del texto que puede **comprometer la trazabilidad de la información original.**

Este escenario evidencia que los resúmenes generados tras varias iteraciones tienden a perder tanto cohesión léxica como fidelidad semántica respecto al texto original, con una degradación más severa en los primeros pasos. Aunque GPT-4 Mini mantiene ligeramente mejores métricas a lo largo de las iteraciones, ninguno de los dos modelos escapa a la erosión progresiva del contenido. El análisis refuerza la idea de que la generación iterativa impone un coste significativo en la calidad informativa, especialmente si no se implementan mecanismos de control de temperatura o de verificación intermedia.

### 4.1.2 Comparación entre modelos

Para evaluar el rendimiento relativo entre los modelos *GPT-4 Mini* y *Gemini-1.5-Flash*, se seleccionaron tres iteraciones clave del experimento (1, 5 y 10), representativas del inicio, punto medio y final del proceso de generación iterativa. Se compararon los valores promedio de las métricas **precisión**, **recall** y **F1** en cada iteración, con el objetivo de determinar cuál de los modelos logra preservar mejor la fidelidad semántica y léxica con respecto a los textos originales a lo largo del proceso de resumen.

**Precisión**

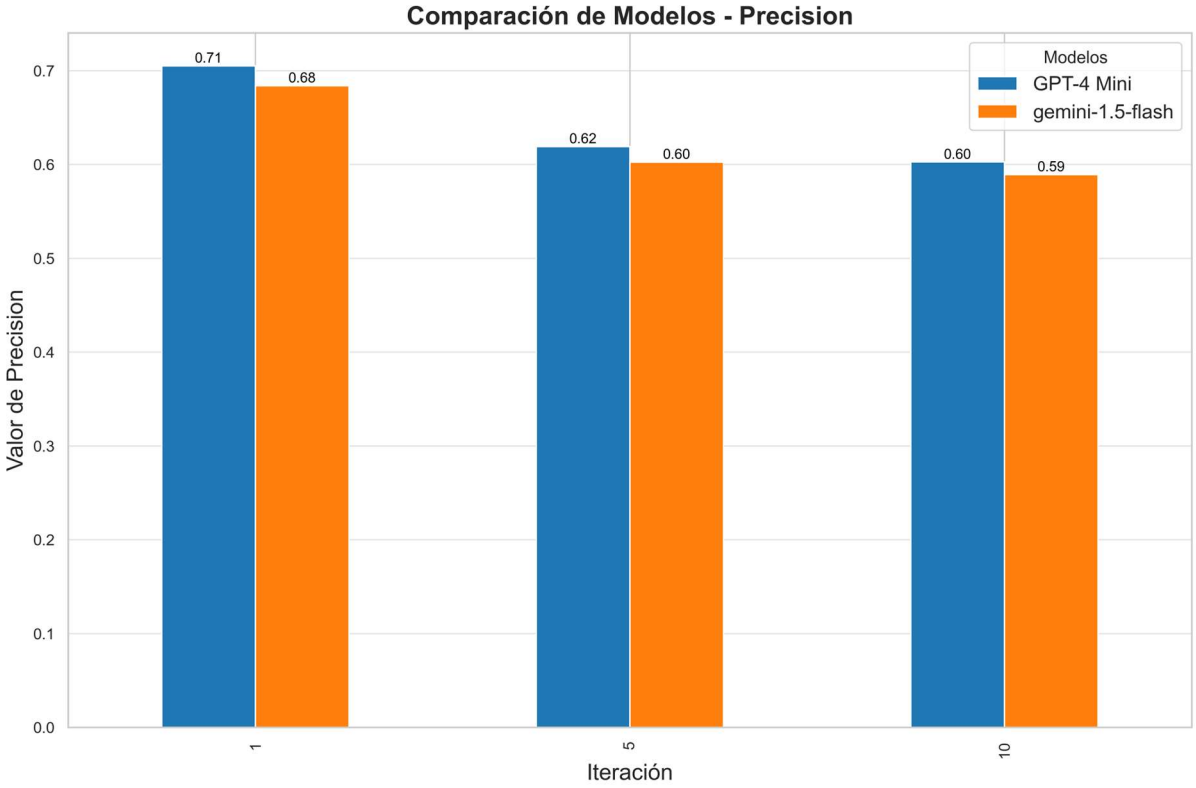


*Figura 17: Comparación de los valores de precisión entre GPT-4 Mini y Gemini-1.5 Flash en las iteraciones 1, 5 y 10.*

La **precisión** mide cuánta de la información generada por el modelo coincide con el texto de referencia. En este caso, ambos modelos comienzan con valores relativamente altos en la iteración 1: 0.71 para GPT-4 Mini y 0.68 para Gemini. Esta diferencia inicial, aunque moderada, sugiere que **GPT-4 Mini es más eficaz seleccionando tokens clave del original desde el principio**, lo que puede ser indicativo de una mayor sensibilidad a los términos más representativos.

A medida que avanza el proceso iterativo, la precisión disminuye en ambos casos, lo cual es esperable dado que los resúmenes tienden a simplificar o reformular la información original. Sin embargo, la caída es más pronunciada en GPT-4 Mini (de 0.71 a 0.60), mientras que Gemini muestra una degradación más contenida (de 0.68 a 0.59). Esta diferencia de comportamiento podría estar relacionada con el enfoque de resumen adoptado por cada modelo: **GPT-4 Mini parece realizar una selección más agresiva de información relevante al principio, mientras que Gemini opta por una reestructuración más conservadora y progresiva.**

Por tanto, GPT-4 Mini destaca en precisión inicial, pero Gemini muestra una mayor estabilidad. Esto sugiere que **GPT-4 Mini es más eficaz en resúmenes de pocas iteraciones**, mientras que **Gemini podría ser más robusto en escenarios prolongados**.

**Recall**

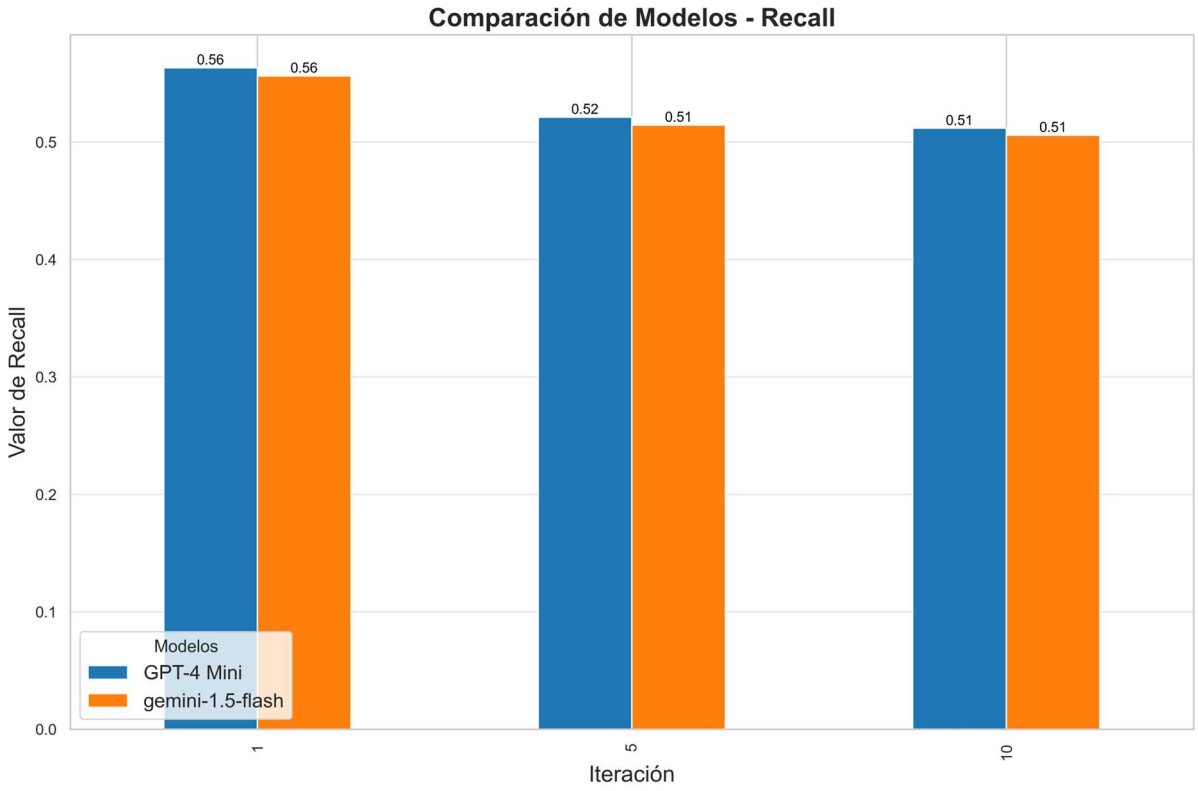


*Figura 18: Comparación de los valores de recall entre GPT-4 Mini y Gemini-1.5 Flash en las iteraciones 1, 5 y 10.*

El **recall** evalúa qué proporción de la información relevante del texto original ha sido recuperada en el resumen. En este caso, los dos modelos se comportan de forma **extraordinariamente similar**, con valores iniciales de **0.56 para ambos** y una ligera caída hasta **0.51 en la iteración 10**.

Esta equivalencia indica que **ninguno de los modelos consigue superar al otro en términos de cobertura global de información**. Ambos logran mantener una recuperación parcial de los contenidos originales, aunque sacrifican cierta amplitud a medida que avanzan las iteraciones. La caída es progresiva pero estable, lo que refleja que los modelos tienden a perder gradualmente elementos informativos conforme se aplican múltiples resúmenes.

La similitud en recall confirma que **ambos modelos tienen limitaciones parecidas a la hora de mantener cobertura de contenido**, especialmente cuando se encadenan múltiples resúmenes. No obstante, su estabilidad sugiere que estos modelos aplican estrategias de compresión que afectan de forma controlada al contenido global.

**F1-Score**

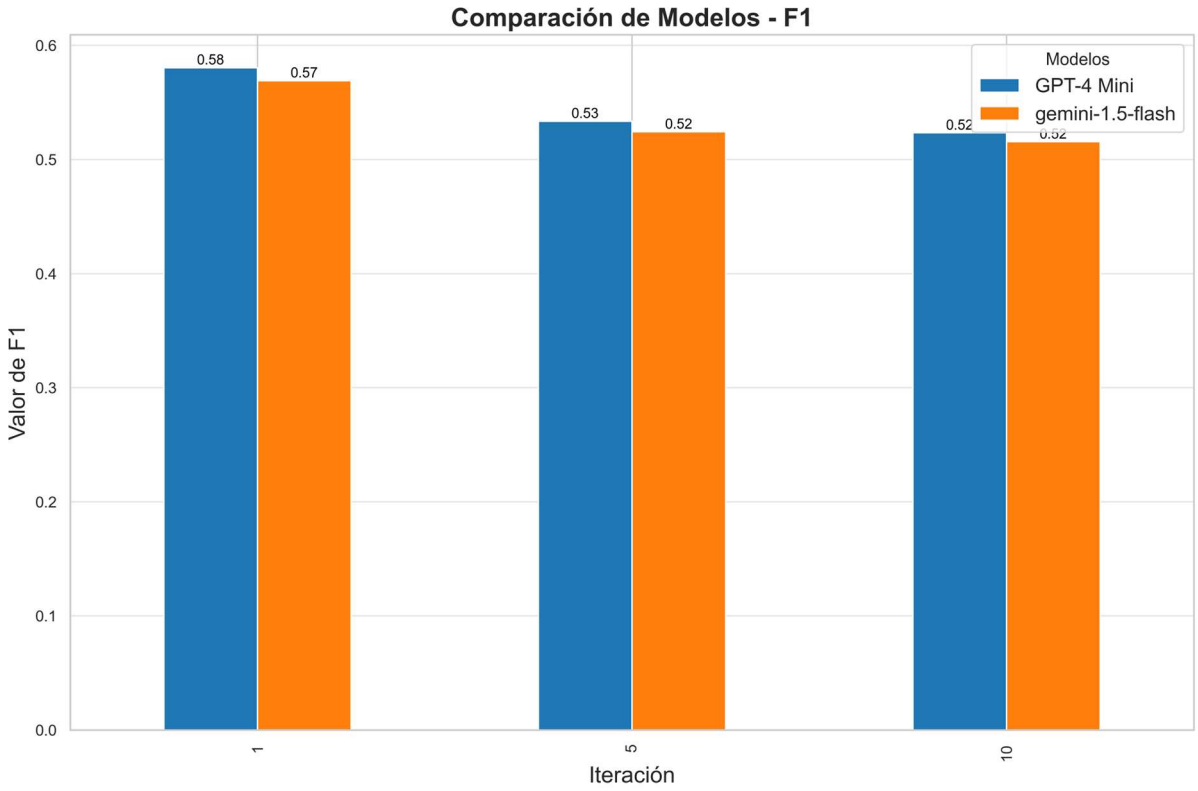


*Figura 19: Comparación de los valores de F1 entre GPT-4 Mini y Gemini-1.5 Flash en las iteraciones 1, 5 y 10.*

El **F1-score**, que representa la media armónica entre precisión y recall, permite valorar el equilibrio entre ambas dimensiones. En todas las iteraciones analizadas, **GPT-4 Mini mantiene una ligera ventaja sobre Gemini**: comienza con **0.58 frente a 0.57**, y termina con **0.52 frente a 0.52**, aunque la diferencia se reduce progresivamente.

Este comportamiento indica que, si bien ambos modelos sacrifican precisión y recall de forma similar, **GPT-4 Mini logra un mejor compromiso entre ambas métricas**, lo que se traduce en resúmenes algo más balanceados. Esta ventaja puede resultar especialmente útil en tareas donde no se puede priorizar exclusivamente la fidelidad superficial (precisión) ni la cobertura (recall), sino que se requiere una síntesis representativa del contenido original.

La ventaja consistente en F1 refuerza la idea de que **GPT-4 Mini ofrece un mejor rendimiento global en términos de equilibrio informativo**, aunque con una caída más visible en precisión a lo largo del tiempo.

### 4.1.3 Consideraciones sobre la validez del enfoque

La estrategia empleada para evaluar el rendimiento de los modelos de lenguaje se basa en el cálculo automatizado de métricas ampliamente aceptadas, como ROUGE [19], BERTScore [20] y, en fases posteriores, la exactitud medida mediante preguntas tipo test. Este enfoque presenta como

principal ventaja su escalabilidad y reproducibilidad, ya que permite comparar de forma sistemática cientos de pares de texto generados por los modelos bajo distintas variantes, sin necesidad de intervención humana directa.

Además, la combinación de métricas complementarias ROUGE, que evalúa la coincidencia textual literal, y BERTScore, que mide la similitud semántica mediante embeddings proporciona una perspectiva más completa sobre la fidelidad y coherencia de los textos generados. La posibilidad de evaluar múltiples versiones (resúmenes y expansiones) a lo largo de iteraciones sucesivas introduce una dimensión temporal que permite analizar cómo se degrada progresivamente el contenido con cada transformación.

No obstante, esta aproximación también presenta limitaciones importantes. En primer lugar, la ausencia de una evaluación cualitativa impide capturar aspectos subjetivos pero relevantes, como la fluidez, la coherencia discursiva o la naturalidad del lenguaje. En segundo lugar, existe un posible desajuste con la interpretación humana: valores elevados en métricas automáticas no garantizan que un texto resulte comprensible, útil o adecuado desde el punto de vista de un lector. Finalmente, algunas métricas, especialmente ROUGE, dependen fuertemente del texto de referencia, penalizando aquellas formulaciones válidas que no coinciden literalmente con el original.

En conjunto, este método resulta eficaz como herramienta de diagnóstico preliminar, ya que permite realizar comparaciones controladas entre modelos de forma automatizada. Sin embargo, debería ser complementado con evaluaciones humanas siempre que se requiera una valoración más fina, especialmente en contextos donde la calidad del lenguaje generado tenga implicaciones prácticas o comunicativas directas.

## 4.2 RESULTADOS DE LA RETENCIÓN SEMÁNTICA A TRAVÉS DE TEST AUTOMATIZADOS

### 4.2.1 Metodología del sistema de evaluación por test

Con el objetivo de evaluar la retención de conocimiento semántico por parte de los modelos tras sucesivas transformaciones del texto, se diseñó un sistema automatizado basado en la generación y corrección de preguntas tipo test. Este procedimiento permite cuantificar la comprensión mantenida por los modelos en diferentes iteraciones (texto original, resumen 1, 5, 10...), y se estructura en varias fases técnicas.

Las respuestas generadas por los modelos de lenguaje, originalmente almacenadas en formato JSONL, fueron procesadas mediante scripts específicos (“**respuestas_to_excel_gpt4.py**” [1] y **“respuestas_to_excel_gemini.py”** [1]) para convertirlas a archivos Excel fácilmente manipulables. Cada entrada se estructuró en columnas que incluyen: el identificador del texto original (id), el número de la pregunta (pregunta, como Q1, Q2, etc.), el tipo de texto evaluado (tipo_texto, que distingue entre original, resumen o expansión) y la respuesta generada por el modelo (respuesta).

Una vez convertidos los datos, se procede a evaluar la exactitud de las respuestas mediante comparación con los ficheros de referencia **(“preguntas_gpt4.xlsx”** [2] y **“preguntas_gemini.xlsx”** [2]). Este análisis se lleva a cabo mediante los scripts **“resultados_gpt-4.py”** [1] y

**“resultados_gemini.py”** [1], los cuales identifican la pregunta correspondiente (Q1–Q10), localizan la opción correcta esperada (por ejemplo, Correct_Option_1) y comparan únicamente la letra de la opción (A, B, C, D), descartando cualquier texto adicional. El resultado de cada comparación se etiqueta como "Correcta" o "Incorrecta", permitiendo así una cuantificación binaria de la respuesta.

Finalmente, se representa gráficamente la efectividad de los modelos según el tipo de texto evaluado. Esta visualización se realiza mediante **“representacion_resultados_gpt-4.py”** [1] y **“representacion_resultados_gemini.py”** [1], que calculan el porcentaje de aciertos para cada iteración del texto y lo representan en forma de gráficas de barras con la biblioteca Matplotlib. Esta representación permite visualizar de forma clara la evolución del rendimiento del modelo, mostrando cómo se conserva o degrada el conocimiento semántico a medida que se aplican resúmenes o expansiones en cadena.

### 4.2.2 Análisis de los resultados obtenidos

Las gráficas de efectividad por tipo de texto permiten observar con claridad cómo se degrada la capacidad de los modelos para responder correctamente a preguntas tipo test a medida que los textos son transformados iterativamente mediante ciclos de resumen y expansión. Este análisis resulta clave para comprender hasta qué punto los modelos mantienen la retención semántica bajo condiciones de reescritura progresiva.

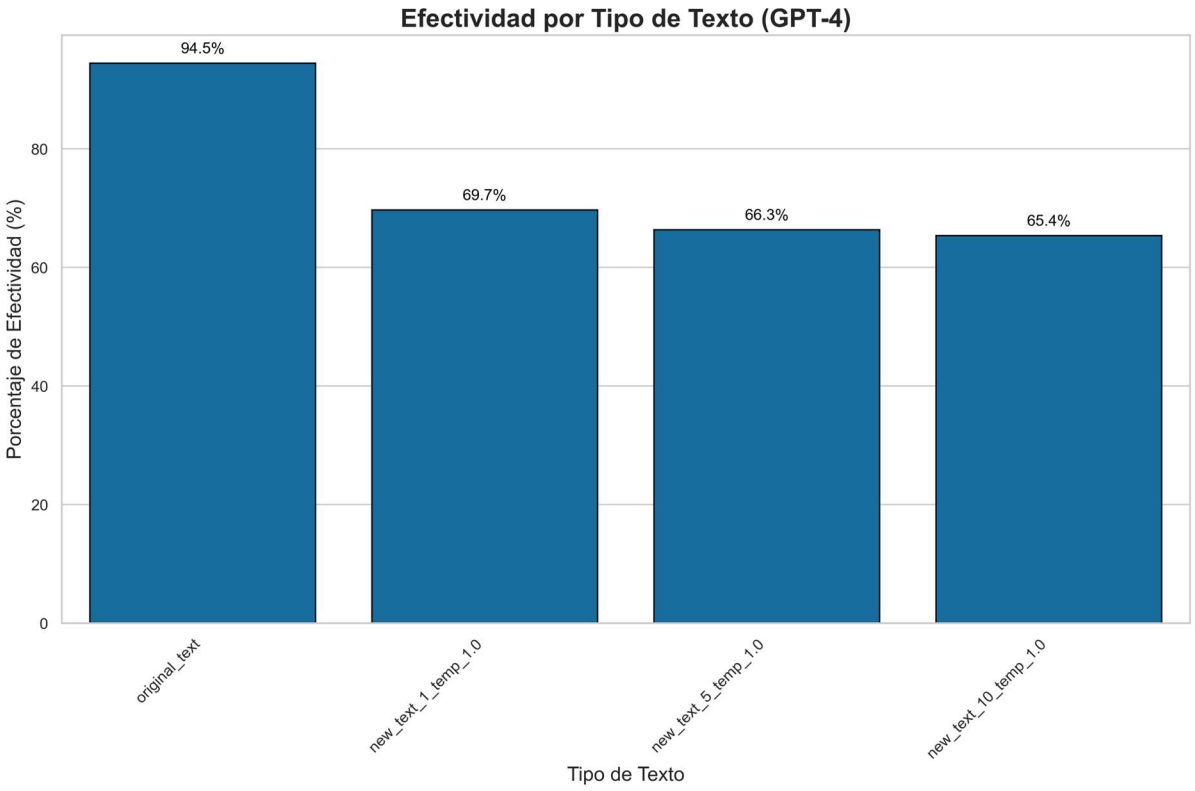


*Figura 20: Porcentaje de aciertos en preguntas tipo test según el tipo de texto procesado por GPT-4 Mini.*

En el caso de **GPT-4 Mini**, la figura muestra una degradación moderada pero estable. El texto original alcanza un 94,5 % de efectividad, lo que refleja una comprensión sólida del contenido

base. Tras una sola iteración de transformación, el porcentaje desciende al 69,7 %, lo que ya sugiere cierta pérdida de información clave. Sin embargo, en las iteraciones posteriores —la 5 y la 10— los resultados se mantienen relativamente constantes, con valores de 66,3 % y 65,4 % respectivamente. Esta evolución sugiere que la mayor parte de la degradación ocurre en la primera fase, estabilizándose posteriormente. En conjunto, estos datos indican que GPT-4 Mini es capaz de conservar una proporción significativa del contenido semántico incluso después de múltiples ciclos de reformulación, lo cual apunta a una arquitectura más resiliente en términos de preservación del significado.

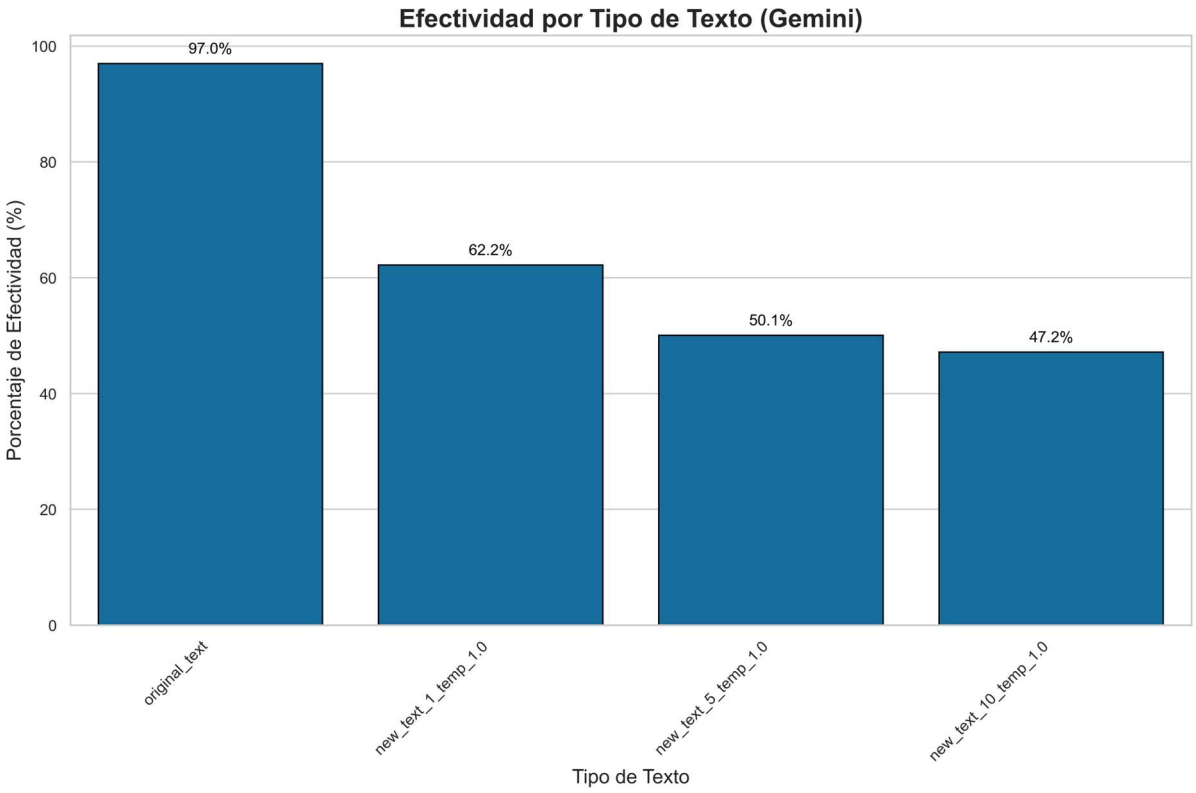


*Figura 21: Porcentaje de aciertos en preguntas tipo test según el tipo de texto procesado por Gemini 1.5 Flash.*

Por el contrario, la evolución de **Gemini 1.5 Flash** presenta una caída mucho más pronunciada. Si bien el texto original parte de un rendimiento excelente (97 % de efectividad), el descenso tras la primera iteración es muy acusado: cae hasta el 62,2 %. A partir de ahí, la pérdida de precisión continúa con valores de 50,1 % en la iteración 5 y 47,2 % en la iteración 10. Esta evolución refleja una degradación mucho más marcada del contenido original, lo que sugiere que el modelo tiene más dificultades para mantener la coherencia y la información esencial cuando los textos son sometidos a varias capas de compresión y expansión. La rapidez de esta pérdida indica que el modelo podría estar priorizando la generación superficialmente coherente sobre la fidelidad al mensaje original.

Comparando ambos modelos de forma global, puede observarse que ambos sufren una caída en la retención semántica a medida que los textos se alejan del original, pero lo hacen con distinta intensidad. GPT-4 Mini muestra una pérdida más gradual y controlada, mientras que Gemini evidencia una fragilidad mayor ante transformaciones iterativas. Esta diferencia podría estar relacionada con

variaciones en los datos de entrenamiento, los mecanismos de atención empleados o la gestión de contexto a largo plazo.

| Tipo de Texto | GPT-4 Mini (%) | Gemini 1.5 Flash (%) |
|---|---|---|
| Original | 94.5 | 97.0 |
| Iteración 1 | 69.7 | 62.2 |
| Iteración 5 | 66.3 | 50.1 |
| Iteración 10 | 65.4 | 47.2 |

*Tabla 6: Comparación del rendimiento en preguntas tipo test entre GPT-4 Mini y Gemini 1.5 Flash para distintos niveles de transformación del texto*

### 4.2.3 Consideraciones sobre la validez del enfoque

El uso de preguntas tipo test como herramienta para evaluar la retención semántica en modelos de lenguaje representa una propuesta novedosa, eficaz y cuantificable. Esta metodología permite medir directamente la capacidad funcional de los modelos para recuperar información precisa tras haber sido sometidos a transformaciones lingüísticas. Esto convierte al enfoque en una alternativa especialmente valiosa en escenarios donde la fidelidad del contenido es crítica.

Una de las principales fortalezas de este método es su escalabilidad. Al automatizar tanto la generación como la evaluación de respuestas, el sistema puede aplicarse fácilmente a cientos de textos y múltiples versiones generadas por los modelos sin necesidad de intervención humana directa. Además, los resultados obtenidos (expresados como porcentajes de acierto) ofrecen una interpretabilidad inmediata e intuitiva, facilitando la comparación entre modelos o configuraciones experimentales sin requerir conocimientos técnicos profundos sobre las métricas subyacentes.

Sin embargo, esta estrategia no está exenta de **limitaciones importantes**. Uno de los factores más sensibles es la calidad de las propias preguntas: si estas no representan con fidelidad la información esencial del texto, la evaluación puede arrojar resultados sesgados. Aunque en este estudio se ha aplicado un sistema de validación automática para asegurar su consistencia, una revisión humana podría reforzar la fiabilidad del conjunto. Otro aspecto relevante es la reducción del conocimiento a decisiones discretas; es posible que un modelo comprenda de forma general el contenido, pero no seleccione la opción exacta, lo que introduce falsos negativos en la medición.

Además, el formato de respuesta puede influir en los resultados. Aunque se ha diseñado el sistema para que los modelos respondan exclusivamente con la letra de la opción correcta, en

múltiples casos (especialmente con Gemini) los modelos generaron frases completas que explicaban su razonamiento, indicaban que no había una respuesta correcta en el texto, o expresaban incertidumbre. Estas respuestas, al no coincidir con el formato esperado, no pudieron ser evaluadas correctamente, penalizando al modelo a pesar de reflejar razonamientos válidos o incluso una cautela fundamentada.

Por último, este enfoque no permite evaluar inferencias complejas o comprensión profunda. Las preguntas están diseñadas para comprobar la retención de información explícita, pero no son adecuadas para analizar habilidades como la deducción, el sentido común o la integración contextual más abstracta, aspectos también fundamentales en la evaluación de modelos de lenguaje.

En conjunto, el uso de preguntas tipo test aporta un valor añadido al estudio de la comprensión en LLMs. Se trata de una herramienta complementaria a las métricas tradicionales, útil especialmente para estudiar la degradación del contenido informativo tras procesos iterativos de resumen o expansión. Sin pretender sustituir otros métodos, este enfoque amplía el espectro de análisis desde la similitud textual hacia la funcionalidad semántica, lo cual resulta especialmente relevante en aplicaciones como la educación, la generación de contenido preciso o entornos jurídicos donde la fiabilidad informativa es esencial.

# 5. CONCLUSIONES Y LÍNEAS FUTURAS

## 5.3 CONCLUSIONES

Este trabajo ha abordado el diseño e implementación de un sistema experimental para evaluar la capacidad de los grandes modelos de lenguaje (LLMs) a la hora de conservar la información semántica a través de procesos de resumen y expansión iterativos. Para ello, se han desarrollado herramientas automáticas que permiten tanto la generación de nuevos textos como su análisis mediante métricas cuantitativas y técnicas de evaluación basadas en comprensión lectora.

Los resultados revelan un fenómeno de degradación progresiva del contenido a medida que aumentan las iteraciones. Esta degradación no se comporta de manera lineal: en las primeras fases (1-3 iteraciones) se observa una pérdida mucho más pronunciada de fidelidad tanto léxica como semántica. Posteriormente, la caída tiende a estabilizarse, sugiriendo que, a partir de cierto punto, los modelos se ajustan a un nuevo equilibrio informativo menos fiel al original, pero más estable en sus transformaciones.

Las métricas **ROUGE** (orientada a similitud léxica) y **BERTScore** (enfocada a similitud semántica mediante embeddings) han sido fundamentales para visualizar estas tendencias. Mientras que ROUGE muestra caídas más abruptas, BERTScore evidencia una pérdida más gradual y matizada, lo que indica que los modelos aún conservan cierta coherencia conceptual incluso cuando las palabras exactas difieren.

Sin embargo, para complementar estas métricas, se ha implementado un sistema de **evaluación mediante preguntas tipo test**, con el que se ha verificado hasta qué punto los modelos pueden responder correctamente sobre textos resumidos o expandidos. Los resultados de esta evaluación muestran una similituf con las métricas automáticas, pero también revelan aspectos que estas no detectan: por ejemplo, modelos que mantienen alta similitud semántica pueden fallar preguntas específicas por omitir detalles clave.

## 5.4 LÍNEAS FUTURAS

El sistema desarrollado abre múltiples vías de trabajo que pueden enriquecer y ampliar los resultados obtenidos:

**Ampliación del corpus experimental**: el presente trabajo ha evaluado un subconjunto de 1000 textos del dataset CNN/DailyMail. Aplicar el sistema a un volumen mayor por ejemplo, 10.000 textos o corpus de distinta naturaleza permitiría obtener resultados más robustos estadísticamente y analizar comportamientos sobre diferentes estilos discursivos.

**Estudio del parámetro *temperature***: los modelos de lenguaje permiten controlar la aleatoriedad en la generación mediante el parámetro *temperature*. Valores bajos generan respuestas más deterministas y fieles al texto, mientras que valores altos introducen más variabilidad. Analizar cómo afecta este parámetro al rendimiento en resumen y expansión podría aportar información clave sobre la relación entre diversidad creativa y retención informativa. Los scripts implementados están preparados para aceptar este parámetro como argumento, por lo que su integración en nuevos experimentos sería directa.

**Evaluación con nuevos modelos**: incluir en el estudio otros LLMs como Claude, LLaMA 3 o Mistral ayudaría a construir una comparativa más amplia entre arquitecturas, estrategias de entrenamiento y comportamientos emergentes.

**Extensión a otros dominios textuales**: aplicar el sistema a textos especializados (jurídicos, técnicos, médicos o narrativos) permitiría explorar cómo se comportan los modelos ante información densa, ambigua o estructuralmente diferente.

**Mejora en la generación de preguntas**: incorporar controles de calidad sobre las preguntas generadas (detección de ambigüedad, redundancia, cobertura temática, etc.) o usar un segundo modelo como verificador mejoraría la fiabilidad del sistema de evaluación tipo test.

# 6. BIBLIOGRAFÍA


[1] J. G. Pérez, «Github,» [En línea]. Available: https://github.com/JAVIERTEL/TFG/tree/main.

[2] J. G. Pérez. [En línea]. Available: https://zenodo.org/records/15714532.

[3] «Hugging Face. CNN/DailyMail Dataset,» [En línea]. Available: https://huggingface.co/datasets/cnn_dailymail.

[4] «OpenAI. (2023). GPT-4 Technical Report,» [En línea]. Available: https://cdn.openai.com/papers/gpt-4.pdf.

[5] «Brown, T. et al. (2020). Language Models are Few-Shot Learners. NeurIPS.,» [En línea]. Available: https://arxiv.org/abs/2005.14165.

[6] «Bommasani, R. et al. (2021). On the Opportunities and Risks of Foundation Models. Stanford CRFM.,» [En línea]. Available: https://arxiv.org/abs/2108.07258.

[7] Y. Zhang, «OPT: Open Pre-trained Transformer Language Models,,» [En línea]. Available: https://arxiv.org/abs/2205.01068.

[8] J. M. S. H. T. Kaplan, «Scaling Laws for Neural Language Models,» [En línea]. Available: https://arxiv.org/abs/2001.08361.

[9] A. G. E. Strubell, «Energy and Policy Considerations for Deep Learning in NLP,» [En línea]. Available: https://aclanthology.org/P19-1355/.

[10] L. Ouyang, «Training language models to follow instructions with human feedback",» [En línea]. Available: https://arxiv.org/abs/2203.02155.

[11] Y. Bai, «Constitutional AI: Harmlessness from AI feedback,» [En línea]. Available: https://arxiv.org/abs/2212.08073.

[12] N. Askell, «A General Language Assistant as a Laboratory for Alignmen,» [En línea]. Available: https://www.anthropic.com/index/alignment-lab.

[13] I. Shumailov, «The Curse of Recursion: Training on Generated Data Makes Models Forget.,» [En línea]. Available: https://arxiv.org/abs/2305.17493.

[14] «Hendrycks, D. et al. (2021). Measuring Massive Multitask Language Understanding.,» [En línea]. Available: https://arxiv.org/abs/2009.03300.

[15] «Cobbe, K. et al. (2021). Training Verifiers to Solve Math Word Problems.,» [En línea]. Available: https://arxiv.org/abs/2110.14168.

[16] P. Clark, «Think you have Solved Question Answering? Try ARC, the AI2 Reasoning Challenge.,» [En línea]. Available: https://arxiv.org/abs/1803.05457.

[17] «LMSYS Chatbot Arena.,» [En línea]. Available: https://chat.lmsys.org/.

[18] J. Zheng, «Judging LLM-as-a-judge with MT-Bench and Chatbot Arena,» [En línea]. Available: https://arxiv.org/abs/2306.05685.

[19] C. Y. Lin, «ROUGE: A Package for Automatic Evaluation of Summaries.,» 2004. [En línea]. Available: https://aclanthology.org/W04-1013.

[20] T. K. V. W. F. W. K. Q. &. A. Y. Zhang, «Evaluating Text Generation with BERT.,» 2020. [En línea]. Available: https://arxiv.org/abs/1904.09675.

[21] Y. Graham, «Re-evaluating Automatic Summarization with BLEU and ROUGE," in Proceedings of the 2015 Conference on Empirical Methods in Natural Language Processing (EMNLP),,» 2015. [En línea]. Available: https://aclanthology.org/D15-1020.

[22] N. R. a. I. Gurevych, «Sentence-BERT: Sentence Embeddings using Siamese BERT-Networks," in Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing (EMNLP-IJCNLP),» 2019. [En línea]. Available: https://arxiv.org/abs/1908.10084.

[23] «Hugging Face. (n.d.). BERTScore documentation. Hugging Face Spaces.,» [En línea]. Available: https://huggingface.co/spaces/evaluate-metric/bertscore.

[24] S. D. Bowman, «Can You Put It All Together: Evaluating Conversational Agents by Their Ability to Reason and Draw Conclusions,» [En línea]. Available: https://aclanthology.org/2023.tacl-1.23.

[25] «W3Schools. Python Tutorial.,» [En línea]. Available: https://www.w3schools.com/python/.

[26] «Tirendaz Academy. Pandas Tutorial.,» [En línea]. Available: https://github.com/TirendazAcademy/PANDAS-TUTORIAL.

[27] «GitHub. Pandas Exercises Repository.,» [En línea]. Available: https://github.com/guipsamora/pandas_exercises.

[28] «Google Research. (s.f.). rouge-score Python library.,» [En línea]. Available: https://github.com/google-research/google-research/tree/master/rouge.

[29] «Hugging Face. (s.f.). bert_score Python library.,» [En línea]. Available: https://github.com/Tiiiger/bert_score.

[30] «Talent.com, “Salario ingeniero técnico en España,”,» [En línea]. Available: https://es.talent.com/salary?job=ingeniero+t%C3%A9cnico.

[31] «OpenAI, “Pricing,” OpenAI.com,» [En línea]. Available: https://openai.com/pricing.

# ANEXO A: ASPECTOS, ÉTICOS, ECONÓMICOS, SOCIALES Y AMBIENTALES

## 5.1 INTRODUCCIÓN

Este Trabajo de Fin de Grado se enmarca en el contexto del análisis del comportamiento de los Grandes Modelos de Lenguaje (LLMs), mediante la implementación de un sistema de evaluación basado en resúmenes, extensiones de texto y preguntas tipo test automatizadas. El objetivo principal es estudiar el grado de retención semántica e informativa de estos modelos tras someterlos a transformaciones iterativas del contenido original.

En este contexto, emergen una serie de aspectos éticos, sociales, económicos y medioambientales de gran relevancia. La proliferación de los LLMs conlleva implicaciones sobre la fiabilidad de la información generada, el consumo energético asociado a su uso intensivo, su impacto potencial en sectores como la educación o los medios de comunicación, así como su posible sesgo o falta de transparencia en las respuestas. Este proyecto, aunque de carácter académico, tiene como trasfondo estas problemáticas actuales y pretende aportar herramientas que permitan evaluar con mayor rigurosidad y transparencia el desempeño de estos sistemas.

## 5.2 DESCRIPCIÓN DE IMPACTOS RELEVANTES RELACIONADOS CON EL PROYECTO

Durante el desarrollo del trabajo se han identificado diversos impactos relevantes relacionados con la sostenibilidad del uso de modelos generativos, particularmente en el contexto del análisis automatizado de su rendimiento semántico. Estos impactos abarcan dimensiones **éticas, sociales, económicas y ambientales**, y su identificación ha sido clave para orientar los análisis posteriores del proyecto.

En primer lugar, desde el punto de vista **ético**, el uso de modelos de lenguaje generativo plantea importantes interrogantes en torno a la veracidad de los contenidos producidos, el riesgo de desinformación y la dificultad para trazar el origen de las fuentes utilizadas por el modelo. En este sentido, la evaluación rigurosa de la fidelidad informativa tras cada transformación del texto permite mitigar estos riesgos y favorece una aplicación más transparente y responsable de estas tecnologías.

Desde una perspectiva **social**, la creciente integración de grandes modelos de lenguaje en contextos educativos y laborales conlleva el riesgo de generar una dependencia tecnológica sin garantías claras sobre la calidad de la información proporcionada. El proyecto aborda este problema proponiendo un marco sistemático para validar la comprensión y consistencia semántica de las respuestas generadas, lo cual resulta especialmente relevante en escenarios donde los LLMs actúan como tutores virtuales o generadores automáticos de contenido formativo.

En cuanto al impacto **económico**, aunque el presente trabajo no persigue una finalidad comercial directa, sí se centra en optimizar el rendimiento de los LLMs mediante estrategias de evaluación eficientes. Este objetivo tiene implicaciones relevantes para el sector empresarial, ya que una mejora en la eficiencia y calidad de uso de estos modelos puede traducirse en soluciones más fiables, rentables y ajustadas a las necesidades reales del usuario final.

Por último, se ha considerado también el impacto **ambiental** asociado al uso de estos modelos. Ejecutar modelos de gran escala como GPT-4 o Gemini implica un consumo energético significativo,

tanto durante la fase de entrenamiento como en los procesos de inferencia. La propuesta de mecanismos de evaluación más ligeros, basados únicamente en el análisis de texto mediante métricas automáticas, puede contribuir a reducir la necesidad de realizar pruebas extensas o costosos ciclos de entrenamiento adicionales. Este enfoque promueve, por tanto, una aproximación más sostenible al desarrollo y aplicación de tecnologías basadas en inteligencia artificial.

En conjunto, los impactos analizados subrayan la importancia de adoptar un enfoque crítico e informado sobre el uso de LLMs, considerando no solo su rendimiento técnico, sino también sus implicaciones éticas, sociales, económicas y ambientales.

## 5.3 ANÁLISIS DETALLADO DE ALGUNO DE LOS PRINCIPALES IMPACTOS

El impacto ético ha sido uno de los aspectos centrales en el desarrollo de este proyecto. Durante la experimentación, se ha detectado en múltiples ocasiones que el modelo evitaba dar una respuesta directa a las preguntas tipo test. En lugar de seleccionar una opción concreta, como se le solicitaba expresamente, generaba respuestas explicativas en las que afirmaba no disponer de suficiente información en el texto o directamente rechazaba realizar una elección. Aunque este comportamiento puede parecer prudente desde una perspectiva de seguridad, también puede interpretarse como una forma de evasión o una manifestación de una comprensión incompleta del contenido, poniendo en duda la fiabilidad del modelo para tareas que exigen precisión informativa.

Este tipo de resultado pone de relieve una de las principales problemáticas éticas asociadas al uso de modelos generativos: su tendencia a responder de forma plausible sin una verdadera comprensión del contenido subyacente. La confianza excesiva en estas respuestas, por parte de usuarios, puede conducir a la aceptación de afirmaciones erróneas o engañosas como si fueran verdades absolutas. Este fenómeno se agrava por la forma en que muchos LLMs presentan sus salidas: con un lenguaje fluido, estructurado y con apariencia de autoridad, lo que aumenta su potencial de influencia sobre quienes los consultan.

Tal como demuestra este trabajo, los modelos pueden perder gradualmente información clave tras múltiples iteraciones de transformación textual, hasta el punto de no poder responder correctamente a preguntas simples sobre el contenido original. Sin embargo, durante todo este proceso siguen generando respuestas bien formuladas que aparentan consistencia. Esto evidencia un riesgo real de **desinformación**, no intencionada pero igualmente peligrosa, en especial en contextos educativos, médicos, legales o científicos, donde una interpretación errónea puede tener consecuencias graves.

En este sentido, el trabajo propone y valida una estrategia de evaluación centrada en la **verificación funcional del conocimiento retenido**, en lugar de limitarse al análisis de coincidencias léxicas o semánticas. Esta metodología ofrece una vía más transparente y directa para determinar si un modelo mantiene la información necesaria para responder con precisión, promoviendo así prácticas más responsables y explicables en la implementación de inteligencia artificial.

Por otro lado, el uso masivo de LLMs merece una reflexión profunda. Estos modelos están cada vez más presentes en aplicaciones de uso cotidiano, desde asistentes virtuales hasta motores de búsqueda o plataformas de generación de contenidos y son utilizados por millones de personas en todo el mundo. El problema radica en que muchos usuarios no comprenden cómo funcionan estos sistemas, ni son plenamente conscientes de sus limitaciones. Como consecuencia, tienden a atribuirles una autoridad desmesurada, asumiendo que todo lo que "dice la IA" es correcto o verificable.

Este fenómeno convierte a los LLMs en herramientas de influencia social de gran alcance. Pueden moldear opiniones, reforzar sesgos, propagar errores o simplificaciones, y generar falsas certezas. Si no se implementan mecanismos adecuados de validación, supervisión y educación del usuario, la proliferación de estos sistemas puede derivar en una **crisis de confianza en la información**, donde se difumine la frontera entre conocimiento fiable y generación estadística sin criterio.

.

## 5.4 CONCLUSIONES

El desarrollo de este sistema de evaluación ha permitido no solo cuantificar el rendimiento de los modelos de lenguaje, sino también identificar oportunidades de mejora en la forma en que se analiza su comportamiento, aportando una perspectiva crítica sobre su capacidad real para conservar la fidelidad informativa a lo largo de procesos de generación iterativa.

Desde un punto de vista **ético y social**, este proyecto subraya la necesidad de ir más allá de las métricas superficiales para valorar a los LLMs. Validar no solo lo que estos modelos pueden generar, sino lo que realmente comprenden y son capaces de retener tras múltiples transformaciones, es esencial para garantizar un uso responsable en ámbitos donde la precisión y la veracidad son fundamentales. Asimismo, se ha evidenciado el riesgo de desinformación derivado de una confianza excesiva en los modelos, lo que refuerza la necesidad de estrategias de evaluación centradas en la retención de conocimiento y la comprensión funcional.

Desde la perspectiva **ambiental**, la propuesta de un sistema automatizado, reutilizable y eficiente para evaluar el contenido generado contribuye a reducir la dependencia de procesos costosos en términos energéticos, como la reevaluación manual o el reentrenamiento innecesario de modelos. Apostar por evaluaciones ligeras y escalables permite optimizar el uso de los recursos computacionales, favoreciendo una aproximación más sostenible al desarrollo y la validación de inteligencia artificial.

En términos **económicos**, el enfoque adoptado puede suponer un valor añadido importante, al facilitar la detección de puntos débiles en los modelos sin incurrir en grandes costes de validación. Esta capacidad de diagnóstico automatizado puede integrarse en entornos de desarrollo de aplicaciones basadas en LLMs, mejorando su calidad sin comprometer los recursos de la organización.

En conclusión, la incorporación de criterios de sostenibilidad entendida en un sentido amplio: ético, social, económico y medioambientaal, ha aportado un valor añadido claro al proyecto. No solo ha permitido evaluar de forma más rigurosa la calidad de los modelos, sino también fomentar un uso más consciente, responsable y eficiente de estas tecnologías.

# ANEXO B: PRESUPUESTO ECONÓMICO

| COSTE DE MANO DE OBRA (coste directo) | Horas | Precio/hora | Total |
|---|---|---|---|
| | 360 | 15 € | **5.400 €** |

| COSTE DE RECURSOS MATERIALES (coste directo) | Precio de compra | Uso en meses | Amortización (en años) | Total |
|---|---|---|---|---|
| Ordenador personal | 350,00 € | 8 | 5 | 56,67 € |
| API de OPENAI | | | | 20,00 € |

| COSTE TOTAL DE RECURSOS MATERIALES | **76,67 €** |
|---|---|

| | | | |
|---|---|---|---|
| **GASTOS GENERALES (costes indirectos)** | 15% | sobre CD | **821,50 €** |
| **BENEFICIO INDUSTRIAL** | 6% | sobre CD+CI | **377,89 €** |

| | | |
|---|---|---|
| **SUBTOTAL PRESUPUESTO** | | **6.676,06 €** |
| **IVA APLICABLE** | 21% | **1.401,97 €** |
| **TOTAL PRESUPUESTO** | | **8.078,03 €** |

El presente presupuesto estima los recursos necesarios para el desarrollo del Trabajo de Fin de Grado, considerando tanto el tiempo de dedicación del estudiante como los recursos materiales empleados. El coste de mano de obra se ha calculado en base a una estimación de 360 horas de trabajo, aplicando una tarifa de **15 €/hora**. Esta tarifa está alineada con el salario medio bruto anual de un ingeniero técnico en España (26 000 €/año o 13,33 €/h) [30], ajustada al alza para reflejar la especialización técnica y la dedicación en tareas de desarrollo, análisis y experimentación. Por otro lado, se ha incluido un coste de **20 €** por el uso de la API de OpenAI, empleado para acceder al modelo GPT-4o Mini, durante aproximadamente 30 horas de interacción. Según la tarifa oficial, este modelo tiene un precio de 0,15 US$ por millón de tokens de entrada y 0,60 US$ por millón de tokens de salida [31], lo cual justifica la estimación. Asimismo, se contempla un coste asociado al uso de un ordenador personal, estimado mediante amortización proporcional. Finalmente, el presupuesto incluye un 15 % en concepto de gastos generales (costes indirectos) y un 6 % por beneficio industrial, siguiendo las directrices del GISD, llegando a un total de **8 078,03 €,** IVA del 21 % incluido.