Departamento de
Ingeniería Industrial
Universidad de Concepción

Optimización del proceso de clasificación, distribución y orden de solicitudes de clientes mediante modelos de lenguaje en Andritz Chile

Por

Marcelo Adrián Muñoz Ulloa

Memoria de Título presentada a la Facultad de Ingeniería de la Universidad de Concepción para optar al título profesional de Ingeniero Civil Industrial

Profesor guía

Carlos Camilo Navarrete Lizama

Agosto 2025

Concepción Chile

RESUMEN

Esta memoria de título presenta el diseño e implementación de una solución para optimizar la organización de solicitudes de clientes en el área de ventas de repuestos de Andritz Chile. El trabajo se divide en dos ejes principales: el desarrollo de un sistema automatizado que mejora el proceso operativo actual y un estudio experimental que evalúa el desempeño de distintos modelos de lenguaje y técnicas de prompting en la extracción de información técnica.

El problema radica en la alta carga operativa asociada a la revisión manual de solicitudes, provocando pérdida de tiempo y duplicación de esfuerzos, que se puede dedicar a tareas más relevantes.

La solución desarrollada integra herramientas como Power Automate, Python, SharePoint y modelos de lenguaje (LLMs), automatizando la detección de correos, extracción de datos en formato XML y su visualización en Excel complementado con datos históricos desde archivos maestros.

Paralelamente, se comparan cuatro modelos de lenguaje (GPT-4, Gemini 2.0 Flash, LLaMA 3 8B y Mistral 7B) combinados con cuatro técnicas de prompting (uso de roles, Chain of Thought, formato estructurado y ejemplos (few-shot)), evaluados con documentos reales y métricas como exactitud, consistencia, percepción de exactitud, tokens generados, costo y tiempo de inferencia.

Los resultados destacan a GPT-4 como el modelo más preciso (98,5 %), con una consistencia del 100 % y una percepción de exactitud del 95 %, aunque con el mayor costo. Gemini 2.0 Flash se consolida como la alternativa más eficiente, alcanzando un rendimiento del 94,6 %, misma consistencia, percepción del 89,2 % y costos mínimos. Por otro lado, los modelos de código abierto ejecutados localmente (LLaMA 3 8B y Mistral 7B) ofrecen ventajas en autonomía y costos directos nulos, pero presentan limitaciones claras en precisión, estabilidad y velocidad. En cuanto a las técnicas de prompting, las más efectivas fueron el formato estructurado y los ejemplos, que permitieron guiar mejor la salida de los modelos mejorando la exactitud y percepción subjetiva.

Finalmente, el sistema fue implementado utilizando el modelo Gemini 2.0 Flash en conjunto con la técnica de prompting de formato estructurado, combinación que ofreció un excelente equilibrio entre exactitud, velocidad y costo. Se concluye que es posible automatizar de forma confiable procesos operativos complejos mediante el uso de LLMs, siempre que se cuente con una arquitectura modular, modelos bien seleccionados y una adecuada ingeniería de prompts. Esta experiencia abre el camino hacia una adopción más amplia de inteligencia artificial en procesos industriales cotidianos.

ABSTRACT

This thesis presents the design and implementation of a solution to optimize the organization of customer requests in the spare parts sales area of Andritz Chile. The work is divided into two main areas: the development of an automated system that improves the current operational process, and an experimental study that evaluates the performance of different language models and prompting techniques in the extraction of technical information.

The problem lies in the high operational burden associated with manually reviewing applications, causing wasted time and duplication of effort that could be devoted to more important tasks.

The solution developed integrates tools such as Power Automate, Python, SharePoint, and language models (LLMs), automating email detection, data extraction in XML format, and visualization in Excel, complemented by historical data from master files.

At the same time, four language models (GPT-4, Gemini 2.0 Flash, LLaMA 3 8B, and Mistral 7B) are compared in combination with four prompting techniques (use of roles, Chain of Thought, structured format, and examples (few-shot)), evaluated with real documents and metrics such as accuracy, consistency, perceived accuracy, generated tokens, monetary cost, and inference time.

The results highlight GPT-4 as the most accurate model (98.5%), with 100% consistency and 95% perceived accuracy, although it has the highest cost. Gemini 2.0 Flash stands out as the most efficient alternative, achieving 94.6% performance, the same consistency, 89.2% perception, and minimal costs. On the other hand, locally executed open-source models (LLaMA 3 8B and Mistral 7B) offer advantages in autonomy and zero direct costs, but have clear limitations in accuracy, stability, and speed. In terms of prompting techniques, the most effective were structured format and examples, which allowed for better guidance of model output and improved accuracy and subjective perception.

Finally, the system was implemented using the Gemini 2.0 Flash model in conjunction with the structured format prompting technique, a combination that offered an excellent balance between accuracy, speed, and cost. It is concluded that it is possible to reliably automate complex operational processes using LLMs, provided that a modular architecture, well-selected models, and adequate prompt engineering are in place. This experience paves the way for wider adoption of artificial intelligence in everyday industrial processes.

Índice de Contenido

Índice de Tablas

Índice de Figuras

CAPÍTULO 1

1 Introducción

Este capítulo establece el contexto general del estudio, que se centra en el uso de modelos de lenguaje de gran tamaño en la implementación de un proceso de automatización. Se presentan los antecedentes que motivan la investigación, incluyendo el interés por aplicar inteligencia artificial en la optimización de tareas en sistemas productivos. Además, se definen los objetivos del estudio y se describe la estructura general de este documento, como punto de partida para su posterior desarrollo.

1.1 Contexto general de Andritz Chile

Andritz Chile Ltda. es una filial del grupo austríaco ANDRITZ, una empresa multinacional que ofrece soluciones de ingeniería para diversas industrias como la minería, energía, tratamiento de aguas, pulpa y papel, biomasa y siderurgia. Su presencia mundial y experiencia técnica han convertido a la compañía en un actor clave en la provisión de equipos, plantas completas y servicios especializados, como mantenimiento predictivo y optimización de procesos industriales (ANDRITZ, 2025).

En Chile, una de las unidades de negocio más relevantes es la venta de repuestos y servicios asociados para la industria de la pulpa y el papel. Según el informe financiero de ANDRITZ (2024), esta industria representa el 34% de los ingresos del grupo a nivel global, siendo el área de ventas de piezas responsable de más del 60% de los ingresos nacionales. Dada esta relevancia, es fundamental que los procesos internos vinculados a la atención de clientes y gestión de solicitudes funcionen con eficiencia, precisión y trazabilidad.

1.2 Descripción del problema operativo

El proceso de recepción y organización de solicitudes de clientes presenta diversas ineficiencias operativas. Las solicitudes ingresan por múltiples canales, principalmente, correos electrónicos y plataformas de licitación, generando un volumen elevado y desordenado de mensajes. Cada analista del equipo de ventas es responsable de una categoría específica de productos. Sin embargo, una sola solicitud puede contener múltiples ítems de diferentes categorías, lo que obliga a que varios analistas revisen el mismo correo.

Esta situación deriva en una revisión manual y repetitiva, con duplicación de esfuerzos, alta carga operativa, errores en la asignación de responsabilidades y demoras en la respuesta al cliente. El proceso, además de ser propenso a fallos humanos, carece de trazabilidad automatizada y depende en gran medida del criterio de cada analista.

1.3 Justificación del proyecto

En un entorno donde la velocidad de respuesta, la trazabilidad de información con el cliente y la organización del trabajo impactan en la competitividad, es muy importante optimizar este proceso. Automatizar la clasificación, distribución y orden de solicitudes no solo reduce tiempos y errores, sino que también libera recursos humanos para tareas de mayor valor para la empresa, como la elaboración de cotizaciones o el contacto directo con clientes.

Asimismo, el uso de tecnologías de inteligencia artificial permite interpretar y clasificar solicitudes complejas de forma precisa y escalable. Incorporar herramientas como Power Automate y Python permite articular flujos automatizados, integrando sistemas que ya son utilizados por la empresa, como Outlook y SharePoint.

1.4 Objetivos

1.4.1 Objetivo general

Desarrollar un sistema automatizado basado en modelos de lenguaje para optimizar la clasificación y organización de solicitudes de clientes en el área de ventas de repuestos de la empresa, reduciendo los tiempos de revisión y asignación.

1.4.2 Objetivos específicos

- Modelar el proceso actual de recepción, distribución y organización de solicitudes de clientes en el área de ventas.

- Investigar y analizar modelos de lenguaje adecuados para la automatización de la clasificación y asignación de solicitudes de clientes.

- Incorporar técnicas de ingeniería de prompt[1] en el diseño y evaluación de los modelos, para encontrar el modelo con mejor desempeño según los requerimientos de la empresa.

- Desarrollar e implementar un prototipo de sistema automatizado que utilice modelos de lenguaje para analizar, categorizar y organizar solicitudes de clientes de manera eficiente.

- Evaluar el desempeño del sistema propuesto mediante pruebas con datos reales, comparando tiempos de procesamiento y precisión de la extracción de información.

- Proponer recomendaciones para la posterior escalabilidad y mejora continua del sistema automatizado dentro del área de ventas.

1.5 Alcances y limitaciones

1.5.1 Alcances

Este proyecto considera como alcance el procesamiento de solicitudes que ingresan en idioma español a través de correo electrónico, que contienen archivos adjuntos en formato .docx o .pdf. El sistema permite procesar automáticamente el contenido textual de los correos y estos documentos, identificar si corresponden a solicitudes de cotización, extraer la información relevante, organizarla y dejarla a disposición del vendedor en un solo lugar. El flujo contempla el almacenamiento momentáneo de los archivos en OneDrive, su posterior almacenamiento definitivo en SharePoint y la visualización estructurada de la información en listas para facilitar su revisión.

1.5.2 Limitaciones

Entre las principales limitaciones del proyecto se encuentra que no contempla la generación automática de cotizaciones. Además, la ejecución del script[2] de procesamiento se realiza de forma local mediante una tarea programada. La comparación de modelos de lenguaje se basa en un conjunto limitado de documentos históricos, lo que acota la generalización de los resultados. Finalmente, por

[1] El prompt es una instrucción textual que se le entrega al modelo de lenguaje para guiar la salida de su respuesta

[2] Un script es un conjunto de instrucciones escritas en un lenguaje de programación que una computadora puede ejecutar automáticamente, generalmente sin intervención del usuario.

concepto de la confidencialidad de la información de la empresa, no es posible acceder a las APIs del portal SAP Ariba, donde se almacenan muchas de las solicitudes y documentos adjuntos provenientes de procesos de licitación.

1.6 Metodología general del trabajo

El desarrollo del proyecto se basa en una metodología aplicada y comparativa. Primero, se diseña e implementa un flujo automatizado para capturar correos entrantes y almacenar la información básica y adjuntos. Luego, el adjunto es revisado por un modelo de lenguaje para que retorne de forma ordenada la información solicitada a través de un prompt, para que finalmente se guarde un nuevo documento con la información requerida. El proceso detallado de la solución automatizada, junto con su arquitectura técnica e integración entre herramientas, se presenta en el Capítulo 4.

Se prueban diferentes modelos utilizando distintas técnicas de prompting para analizar cuál entrega los mejores resultados. Finalmente, se evalúa el desempeño del sistema en condiciones reales, midiendo tiempos, costos, exactitud de la extracción, consistencia y percepción de usuarios.

1.7 Estructura del documento

Este documento se encuentra estructurado en ocho capítulos. El Capítulo 1 corresponde a la introducción del proyecto, se presentan los antecedentes, justificación, objetivos, alcances, limitaciones y metodología general. El Capítulo 2 desarrolla el marco teórico donde se fundamenta el uso de inteligencia artificial, modelos de lenguaje de gran tamaño (LLMs) y técnicas de prompting, así también se muestran ejemplos de aplicaciones reales en empresas. El Capítulo 3 evidencia el diagnóstico del proceso actual en el área de ventas de Andritz Chile, donde se identifican los principales problemas que justifican la intervención. El Capítulo 4 describe la estrategia de aplicación y metodología del sistema automatizado propuesto, incluyendo la evaluación de modelos de lenguaje, técnicas de prompting y herramientas tecnológicas utilizadas. El Capítulo 5 detalla la implementación práctica del prototipo automatizado, donde se describe cada etapa del flujo a través de una solicitud real. El Capítulo 6 expone los resultados obtenidos en la evaluación experimental, comparando el desempeño de los distintos modelos y técnicas con métricas objetivas y subjetivas. El Capítulo 7 presenta la discusión de los resultados obtenidos, realizando comparaciones entre los modelos y técnicas estudiadas. Finalmente, el Capítulo 8 expone las conclusiones generales del estudio y las recomendaciones para futuras mejoras.

CAPÍTULO 2

2 Marco teórico

Esta sección busca fundamentar conceptualmente el uso de modelos de lenguaje en entornos industriales, abordando su funcionamiento, evolución y aplicaciones.

Este capítulo constituye la base conceptual necesaria para comprender el enfoque metodológico y justificar la elección tecnológica utilizada en el desarrollo de la solución propuesta.

2.1 Inteligencia Artificial

La inteligencia artificial (IA) tiene sus raíces muchos años atrás. Desde los años 40, los relatos visionarios de Asimov (1942), sobre robots con leyes establecidas de comportamiento o el trabajo pionero de Alan Turing con "The Bombe", su máquina descifradora de códigos para los ingleses en la Segunda Guerra Mundial. Además de los fundamentos teóricos y éticos de la IA que fueron establecidos mucho antes de su formalización.

Fue en 1956, con la conferencia de Dartmouth liderada por McCarthy et al. (1955), donde se oficializó el término "inteligencia artificial", marcando el inicio de una disciplina científica que ha experimentado avances deslumbrantes. Además, periodos en los que se ha cuestionado su rendimiento y elevados gastos en su investigación, hasta convertirse en una tecnología transformadora en la actualidad (Haenlein, 2019).

Según Vallejo Echavarría (2024), la IA se puede definir como la encargada de desarrollar sistemas capaces de realizar tareas que normalmente requieren la inteligencia humana. Actualmente, se está utilizando cada vez más, en diferentes áreas, para automatizar tareas repetitivas, permitiendo procesar y organizar mucha información a la vez, realizando clasificaciones y distribuyendo información de manera más eficiente que los métodos manuales tradicionales.

2.2 Procesamiento del Lenguaje Natural

El procesamiento de lenguaje natural (NLP en su acrónimo en inglés) es un campo de la inteligencia artificial que busca permitir la interacción entre sistemas computacionales y el lenguaje humano, con el objetivo principal de que las máquinas logren simular el razonamiento humano (Khurana, 2017).

Jurafsky (2021) menciona que existen tareas que permiten transformar texto sin estructura, de manera útil para sistemas inteligentes, como:

- **Tokenización y segmentación:** Se divide un texto en unidades básicas llamadas tokens (palabras, signos, números) y en oraciones.

- **Clasificación de texto:** Tarea de asignar una o más etiquetas a un documento o fragmento.

- **Extracción de entidades nombradas:** Se identifica y etiqueta a entidades específicas en el texto, como empresas, nombre de personas, fechas o productos.

- **Análisis sintáctico y semántico:** Busca entender la estructura gramatical y el significado de una oración.

- **Resumen automático y respuesta a preguntas:** Condensa textos extensos o genera respuestas específicas a preguntas formuladas en lenguaje natural.

Considerando el ámbito empresarial, el NLP ha cobrado relevancia por su capacidad para automatizar tareas que necesiten comprensión del lenguaje, tales como la clasificación de correos electrónicos, análisis de opiniones de clientes, generación de respuestas automáticas, categorización de solicitudes, entre otras. Estas acciones son muy valiosas en contextos donde el volumen de datos tipo texto es alto y se requiere una gestión eficiente de la información.

Un ejemplo concreto de aplicación se encuentra en RExtractor, un sistema diseñado para extraer y clasificar automáticamente entidades y relaciones en documentos legales mediante árboles de dependencias lingüísticas, demostrando la viabilidad de estas tecnologías en entornos reales con documentos no estructurados (Hladká, 2015). Además, estudios como el de Chiticariu (2010) evidencian cómo los modelos basados en NLP pueden adaptarse eficazmente a diferentes dominios para tareas como la categorización automatizada de solicitudes y la extracción de información.

2.3 Modelos de Lenguaje de Gran Tamaño

Los Modelos de Lenguaje de Gran Tamaño (LLM en su acrónimo en inglés) son sistemas de IA generativa diseñados para que puedan comprender y comunicarse en lenguaje humano natural, lo que

les posibilita realizar tareas como clasificación, resumen, generación de texto y análisis. Estas herramientas permiten interpretar y clasificar contenido no estructurado con alta precisión (Larson, 2023).

Los modelos de lenguaje han evolucionado rápidamente gracias al desarrollo de arquitecturas avanzadas como los Transformers. Vaswani et al. (2017) explican que estos han permitido capturar relaciones contextuales entre palabras de manera más eficaz que otros modelos previos, redefiniendo la generación y comprensión del lenguaje natural. A diferencia de los enfoques anteriores basados en redes neuronales recurrentes, que sufrían problemas como dificultad para procesar secuencias largas, los Transformers introducen un mecanismo que permite procesar todas las palabras de una secuencia de forma simultánea. Este mecanismo se conoce como *self-attention*, y permite que cada palabra de la secuencia evalúe su relación con las demás, asignando pesos a cada una en función de su relevancia contextual. Este proceso se realiza en paralelo, mejorando significativamente la eficiencia y la capacidad de modelar dependencias a largo plazo.

Según Onat Topal et al. (2021), el éxito de esta arquitectura la ha convertido en la base para una nueva generación de modelos de lenguaje preentrenados, muy efectivos como BERT, que procesa el contexto de las palabras en ambas direcciones, permitiendo representar una palabra en función tanto de su contexto anterior como posterior. BERT fue introducido por Devlin (2019), como un modelo bidireccional basado en Transformers para tareas de comprensión del lenguaje natural. Otros modelos como RoBERTa, una optimización de BERT desarrollada por Meta, han demostrado que extender el entrenamiento puede mejorar en gran medida el rendimiento del modelo original, o también GPT, un modelo autorregresivo orientado a la generación de texto que predice la siguiente palabra en una secuencia (Onat Topal M, 2021).

El uso de LLMs permite automatizar tareas como la clasificación de textos o solicitudes con una precisión superior a los métodos tradicionales, reduciendo el esfuerzo manual y mejorando los tiempos de respuesta. Esta superioridad ha sido ampliamente documentada, modelos como BERT y RoBERTa superaron sistemáticamente a algoritmos clásicos como SVM o redes neuronales estándar en experimentos de clasificación de texto, alcanzando niveles de exactitud significativamente más altos en distintas categorías (Petridis, 2024).

2.4 Limitaciones y desafíos de los LLMs en las empresas

Si bien los LLMs han demostrado un enorme potencial para transformar procesos empresariales mediante la automatización de tareas, su implementación no está exenta de limitaciones que deben ser considerados.

- **Costos computacionales y requerimientos técnicos**

El entrenamiento y la ejecución de LLMs requieren una infraestructura computacional de alto rendimiento, incluyendo GPU especializadas y gran capacidad de almacenamiento, lo que conlleva altos costos operacionales y barreras de acceso para pequeñas y medianas empresas (OpenAI, 2023).

- **Riesgos de precisión y alucinación**

A pesar de su capacidad para generar texto coherente, los LLMs pueden presentar alucinaciones, esto es cuando el modelo entrega información que no corresponde con los datos reales, respuestas aparentemente muy buenas pero incorrectas o inventadas, lo que representa un riesgo en aplicaciones empresariales. Este comportamiento es crítico en contextos donde las decisiones se basan en información generada automáticamente (Cao, 2023).

- **Consideraciones éticas y de privacidad**

El uso de LLMs en entornos empresariales plantea desafíos relacionados con la protección de datos sensibles, confidencialidad y cumplimiento de normativas. Además, los modelos pueden reproducir sesgos presentes en los datos de entrenamiento, lo que puede derivar en decisiones discriminatorias que no se controlan adecuadamente (Bender, 2021).

2.5 Tokenización y segmentación

Una de las actividades básicas que utilizan los LLMs es la tokenización y segmentación, que constituyen una de las etapas iniciales y fundamentales en cualquier proceso de análisis de lenguaje natural. Según Jurafsky (2021), la tokenización se refiere a la tarea de dividir un texto en unidades mínimas significativas denominadas tokens. Un token puede definirse como una secuencia de caracteres que constituye una unidad de análisis, por lo general una palabra, pero también pueden ser signos de puntuación, números o símbolos especiales. Por ejemplo, en la frase “El equipo está

trabajando.", los tokens serían: "El", "equipo", "está", "trabajando", ".". La determinación de la cantidad de tokens se lleva a cabo contabilizando cuántas de estas unidades existen en el texto, en este caso es de cinco tokens. Es muy importante que esta contabilización se realice bien, ya que muchos LLMs tienen límites del número de tokens procesables por cada consulta.

La tokenización afecta directamente la calidad de todas las tareas posteriores, un error en esta etapa puede alterar la comprensión sintáctica o semántica del texto. Por su parte, la segmentación divide el texto en unidades mayores, como oraciones, lo que requiere identificar correctamente los límites gramaticales y puntuación. Estas dos tareas forman la base estructural sobre la cual se construyen modelos más complejos de procesamiento del lenguaje.

2.6 Técnicas de Prompting

El prompting es una técnica para interactuar con modelos de lenguaje, que consiste en formular entradas o instrucciones específicas que guían la respuesta del modelo hacia el objetivo buscado. La efectividad del modelo depende en gran medida de cómo se estructure y diseñe el prompt. Según Anthropic (2024), las mejores prácticas en ingeniería de prompts incluyen:

- **Claridad y contexto:** Los prompts deben ser claros, específicos y proporcionar suficiente contexto para que el modelo entienda la tarea. La inclusión de instrucciones detalladas ayuda a evitar ambigüedades y mejora la precisión de la respuesta.
- **Uso de roles o identidades:** Asignar un rol explícito al modelo, como "Eres un experto en repuestos industriales", permite que las respuestas se ajusten mejor al dominio y contexto esperado.
- **Instrucciones paso a paso o Chain of Thought:** Pedirle al modelo que razone o explique su pensamiento de manera secuencial, puede aumentar la calidad y coherencia de las respuestas, especialmente en tareas complejas.
- **Formato estructurado:** Incorporar etiquetas de formato, puede ayudar a que el modelo organice y entregue la información de forma ordenada.
- **Ejemplos y demostraciones (few-shot):** Proveer ejemplos concretos en el prompt puede guiar al modelo hacia el tipo de respuesta esperada, mejorando la consistencia y relevancia.

2.7 Aplicaciones de modelos de lenguaje en la Industria

Los modelos de lenguaje, basados en NLP como en LLMs, han revolucionado los procesos actuales de las empresas gracias a su capacidad para automatizar tareas que antes requerían obligatoriamente la acción humana. En la industria, su implementación ha sido efectiva en áreas como atención al cliente, recursos humanos y gestión de documentos, facilitando la extracción de datos desde textos no estructurados (IBM, 2024).

Como menciona Taori (2022), un ejemplo es el sistema TaDaa propuesto por investigadores de Apple, que utiliza modelos de lenguaje basados en Transformers para automatizar el proceso de asignación de tickets de soporte a los resolutores adecuados dentro de una organización. El sistema opera asignando automáticamente a grupos funcionales, sugiriendo al mejor resolutor para cada solicitud y recuperando solicitudes similares resueltas anteriormente, permitiendo recuperar en tiempo real ejemplos pasados que pueden servir como referencia para resolver este nuevo ticket. Esto contribuye a reducir los tiempos de resolución, mejorar la asignación inicial y evitar el reenvío de solicitudes.

En base a lo revisado en la literatura, existe la posibilidad de utilizar modelos de lenguaje para automatizar tareas de clasificación, extracción y organización de información. El uso de estas herramientas no solo permite mejorar la eficiencia en el ingreso y orden de las solicitudes, sino que también abren la puerta a la integración de la inteligencia artificial con procesos humanos actuales, fomentando la transformación digital de las organizaciones y el uso de tecnología para disminuir el esfuerzo manual.

CAPÍTULO 3

3 Proceso de atención a clientes en Andritz Chile

Este capítulo presenta el proceso de atención a clientes en Andritz Chile, describiendo su funcionamiento actual, las principales problemáticas detectadas y la necesidad de avanzar hacia su automatización.

3.1 Actual flujo de recepción y manejo de solicitudes

El proceso de atención a clientes en Andritz Chile, como se muestra en la Figura 3.1, inicia con la recepción de solicitudes, las cuales pueden llegar a través de correos electrónicos directos o mediante plataformas de licitación abiertas. Estas plataformas invitan a múltiples proveedores a participar en cotizaciones, generando una gran cantidad de correos electrónicos simultáneos, muchas veces duplicados o con información redundante.

Cada solicitud puede contener desde un ítem puntual hasta listados extensos con más de 50 productos cuando es necesario para una PGP[3]. Estos ítems pertenecen a distintas categorías de Productos, las cuales están distribuidas entre analistas específicos del equipo de ventas, según su especialización. Dado que una sola solicitud puede involucrar múltiples categorías, es común que varios analistas deban revisar un mismo correo, lo que genera duplicidad de esfuerzos y ralentiza el proceso.

[3] PGP (Parada General Programada): Es la detención total o parcial planificada de una planta industrial, para ejecutar mantenimientos e inspecciones.

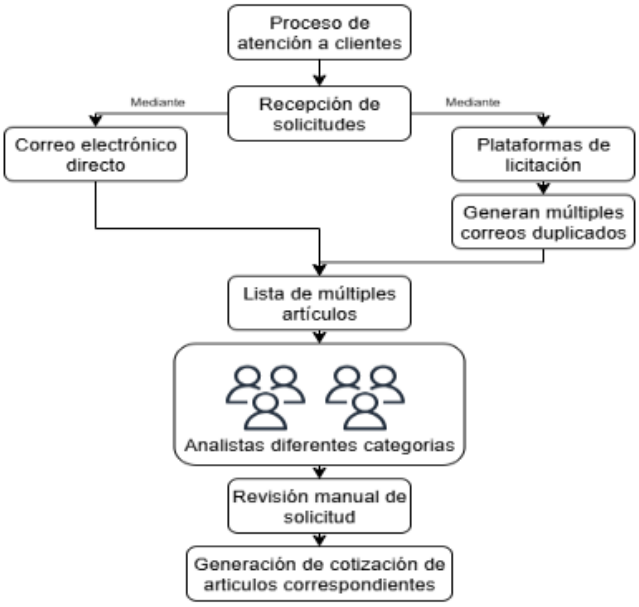


Figura 3.1: Diagrama de funcionamiento actual

Actualmente, no existe un sistema automatizado que organice, clasifique o derive las solicitudes de forma inteligente. Cada analista debe abrir manualmente cada correo, revisar el contenido y los archivos adjuntos, identificar si alguno de los ítems le corresponde, y en caso de que así sea, comenzar a elaborar la cotización. Esto implica una gestión fragmentada, descentralizada y altamente dependiente del juicio humano.

3.2 Identificación de puntos críticos

El flujo descrito presenta diversas problemáticas que afectan la eficiencia y la calidad del proceso. Entre los principales riesgos operacionales se identifican:

- **Alta carga operativa:** cada analista revisa decenas de correos, muchos de los cuales no son relevantes para su categoría de productos.
- **Duplicación de esfuerzos:** como no hay filtro automático, los mismos correos son abiertos por varios analistas, especialmente cuando contienen múltiples ítems.
- **Falta de trazabilidad y registro sistemático:** no hay una plataforma centralizada donde se almacenen las solicitudes, lo que impide un seguimiento estructurado.

- **Errores humanos en la interpretación de solicitudes:** debido al volumen de ítems solicitados, la alta cantidad de plantas donde solicitar los productos y los detalles técnicos de los ítems, pueden producirse omisiones, errores de categorización o demoras en la atención.

Estas ineficiencias no solo impactan la productividad del equipo de ventas, sino que también pueden afectar la experiencia del cliente, al generar demoras, respuestas incompletas y una falta de información oportuna.

3.3 Carga operativa

Durante el levantamiento de información con el equipo de ventas, se observa que un analista puede destinar cerca de tres horas diarias exclusivamente a la revisión de correos electrónicos. Este tiempo no incluye la elaboración de cotizaciones ni el contacto con proveedores o clientes, lo que indica una proporción significativa de tiempo invertido en tareas repetitivas y de bajo valor agregado.

3.4 Justificación técnica de automatización

Frente a este panorama, la implementación de un sistema automatizado que permita detectar, ordenar las solicitudes y además entregar información filtrada, se presenta como una solución técnica y estratégicamente necesaria. Automatizar este proceso permitiría:

- Reducir los tiempos de clasificación y orden de solicitudes.
- Evitar duplicación de esfuerzos, disminuyendo la cantidad de tiempo utilizado por cada analista en la revisión de cada correo.
- Mejorar la trazabilidad, al almacenar cada solicitud clasificada en listas estructuradas de SharePoint.
- Liberar tiempo del personal para tareas de mayor complejidad y valor técnico.
- Reducir información irrelevante, pues al utilizar modelos de lenguaje para interpretar lenguaje técnico y semiestructurado, es posible entregarle al analista la información necesaria respecto a esa solicitud ignorando información que no es de interés.

CAPÍTULO 4

4 Estrategia de aplicación y metodología

Este capítulo presenta la estrategia metodológica aplicada en el estudio, describiendo el diseño general del experimento, las variables consideradas y los criterios utilizados para evaluar el rendimiento de los modelos de lenguaje en tareas de extracción de información.

4.1 Diseño general del estudio

Esta memoria de título tiene un enfoque aplicado, cuantitativo y comparativo. Utilizando una metodología experimental, con la intención de evaluar el rendimiento de distintos modelos de lenguaje en tareas de extracción de información estructurada, desde documentos reales de la empresa.

El diseño del experimento considera la combinación de cuatro modelos de lenguaje y cuatro técnicas de prompting bajo las mismas condiciones de ejecución. El objetivo metodológico es comparar el desempeño de estas combinaciones según métricas objetivas, tales como, exactitud, tiempo de procesamiento y costo, y también subjetiva, como la percepción de exactitud, para determinar cuáles técnicas resultan más efectivas en entornos industriales automatizados.

4.2 Variables del estudio

La Tabla 4.1 resume las variables consideradas en el presente estudio. Las variables independientes, que corresponden a los modelos de lenguaje evaluados y las técnicas de prompting aplicadas, y las variables dependientes, representadas por las métricas utilizadas para analizar el rendimiento de los modelos.

Tabla 4.1: Variables a considerar en el estudio

Variables			
Independientes		**Dependientes**	
Modelos	GPT-4 Gemini 2.0 Flash Mistral 7B LLaMA 3 8B	**Métricas**	Exactitud por campo Percepción de exactitud Consistencia Tokens generados Costo monetario Tiempo de inferencia
Técnicas	Uso de Roles Chain of Thought Formato Estructurado Ejemplos (few-shot)		

4.3 Modelos de lenguaje evaluados

Con el objetivo de comparar el rendimiento de distintos enfoques de modelado en tareas de extracción de datos, se estudia el comportamiento de los cuatro modelos siguientes:

GPT - 4: Es un modelo de lenguaje avanzado desarrollado por OpenAI. Se destaca por su capacidad para comprender y generar texto muy coherente, contexto y precisión en una amplia variedad de tareas de lenguaje natural, incluyendo redacción, traducción, resumen y diálogo conversacional. GPT-4 también puede procesar entradas multimodales y ha sido entrenado con un enfoque en seguridad para minimizar respuestas inapropiadas o erróneas (OpenAI, 2023).

Gemini 2.0 Flash: Es un modelo de lenguaje desarrollado por Google. Se caracteriza por su alta velocidad de respuesta y bajo consumo de recursos, lo que lo hace ideal para tareas que requieren eficiencia en tiempo real, como la generación de texto y clasificación de contenido. Es más ligero que otros modelos de Google como Gemini 2.5 Pro, pero mantiene un rendimiento competitivo en tareas de procesamiento de lenguaje (Google DeepMind, 2024).

Mistral 7B: Es un modelo de lenguaje de código abierto[4] desarrollado por Mistral AI. Está diseñado para ofrecer un alto rendimiento en tareas de generación y comprensión de texto, combinado con

[4] El código abierto es el que su código fuente está disponible para que cualquiera lo utilice, modifique, mejore y distribuya.

eficiencia computacional. Su arquitectura permite un uso rápido y versátil, siendo ideal para aplicaciones que requieren modelos potentes, pero con un tamaño moderado (Mistral AI, 2023).

LLaMA 3 8B: Es un modelo de lenguaje desarrollado por Meta y se trata de un modelo avanzado de lenguaje natural que ofrece respuestas precisas y coherentes, optimizando la comprensión y razonamiento. Está diseñado para ser eficiente en recursos, facilitando su uso en aplicaciones reales como generación de texto y asistentes conversacionales.

Tanto LLaMA como Mistral pueden ejecutarse gratuitamente de forma local mediante plataformas como Ollama, que permiten utilizar el modelo sin complicaciones técnicas, facilitando la interacción mediante interfaces sencillas, para una experiencia rápida y privada (Meta, 2023). Además, está la posibilidad de realizar estas inferencias mediante herramientas como Groq u otras que sí tienen un costo utilizando estas plataformas.

Considerando que el uso de modelos de lenguaje implica costos variables según el proveedor y la modalidad de ejecución, es necesario comparar el valor por millón de tokens de entrada y salida, como se observa en la tabla 4.2. Esto permite dimensionar la viabilidad económica de cada alternativa.

Tabla 4.2: Costo de cada modelo por millón de tokens

	Costo 1 Millón de tokens (USD)	
Modelo	**Entrada**	**Salida**
GPT-4	30	60
GEMINI 2,0 Flash	0,1	0,4
LlaMA 3 8B4	0,05	0,08
MISTRAL 7B	0,05	0,25

4.4 Técnicas de prompting evaluadas

Para estudiar las distintas estrategias de formulación de instrucciones mediante la ingeniería de prompt, se aplican cuatro técnicas de prompting con cada modelo. Estas técnicas permiten analizar cómo varía la calidad de la extracción en función del tipo de guía proporcionada. A continuación, se presentan los cuatro prompts utilizados para el estudio:

Chain of Thought (CoT):

```
mi_prompt =

"""

“Lee el siguiente texto y piensa paso a paso.

Primero, identifica todos los elementos que aparecen.

Luego, para cada uno, extrae la siguiente información: nombre del elemento, número de parte, código de material, cantidad, planta, fabricante y características adicionales.

Entrega todo utilizando etiquetas XML válidas, una por elemento, sin explicaciones ni encabezados.”

"""
```

Formato Estructurado:

```
mi_prompt =

"""

“Extrae todos los elementos mencionados en el siguiente texto.

Para cada ítem, entrega un bloque con las siguientes etiquetas XML:

   • <nombre>

   • <número_parte>

   • <código_material>

   • <cantidad>

   • <planta>

   • <fabricante>

   • <característica_adicional>

Agrupa cada bloque dentro de una etiqueta <elemento>.

Devuelve únicamente una lista de elementos en XML válido, sin explicaciones ni encabezados.”

"""
```

Ejemplo (few-shot):

```
mi_prompt =

"""

"Extrae la información del texto entregado.

A continuación, se muestra un ejemplo de cómo debe ser la salida usando etiquetas XML:

<elemento>

  <nombre>BOQUILLA AHLSTROM</nombre>

  <número_parte>89414001</número_parte>

  <código_material>8900244-A3-006</código_material>

  <cantidad>2</cantidad>

  <planta>2002 Pulp Planta Pacífico</planta>

  <fabricante>AHLSTRO</fabricante>

  <característica_adicional>Equipo del que es parte: caldera recuperadora</característica_adicional>

</elemento>

Ahora, aplica el mismo formato XML al siguiente texto. Devuelve únicamente etiquetas XML válidas sin explicaciones ni encabezados."

"""
```

Uso de Roles:

```
mi_prompt =

"""

“Eres un experto en repuestos industriales con experiencia en análisis técnico.

Tu tarea es revisar el siguiente texto técnico y extraer información relevante sobre cada componente.

Para cada ítem, utilizando etiquetas XML dentro de un contenedor entrega los siguientes datos: Nombre del elemento, número de parte, código de material, cantidad, planta, fabricante, características adicionales.

No incluyas explicaciones ni encabezados. Devuelve únicamente un XML válido."

"""
```

4.5 Herramientas de automatización

Las herramientas de automatización son plataformas diseñadas para simplificar y acelerar la ejecución de tareas repetitivas y procesos complejos. Al implementar estas soluciones se puede mejorar la productividad y reducir la dependencia de la intervención manual.

Power Automate: Es una plataforma que permite crear flujos que reaccionan ante eventos como la llegada de correos y se utiliza para capturar estos correos, extraer contenido, almacenar, distribuir y activar otros procesos. Es una herramienta ampliamente utilizada para automatizar y optimizar procesos. Su principal objetivo es mejorar la productividad, reducir errores y ahorrar tiempo y costos, permitiendo gestionar tareas sin intervención manual. Entre sus ventajas se destacan la capacidad de integrarse con otras aplicaciones de Microsoft y automatizar flujos de trabajo complejos sin necesidad de programación avanzada (Kpaz, 2024).

Programador de tareas de Windows: Es una herramienta integrada que permite automatizar la ejecución de tareas y procesos en el sistema operativo según un horario definido o cuando se producen ciertos eventos específicos. Esta herramienta facilita la programación de acciones, mejorando la eficiencia y el mantenimiento del sistema.

Entre sus funcionalidades, el Programador de tareas puede iniciar programas, ejecutar scripts, enviar correos electrónicos y mostrar mensajes, todo ello sin intervención manual. Esto permite, por ejemplo, automatizar la ejecución de un script de Python para tareas como procesamiento de datos, copias de seguridad o mantenimiento programado (Microsoft, 2024).

4.6 Selección de documentos

Los documentos seleccionados para la realización de este estudio son documentos históricos utilizados por el área de ventas de la empresa, en este caso se utilizan cuatro documentos, en los cuales dos son en formato Word (.docx), como se observa en la Figura 4.1 y dos de tipo PDF (.pdf), como se muestra en la Figura 4.2. Los archivos en formato PDF presentan la cantidad de uno y cinco artículos solicitados, en cambio los archivos en formato Word, presentan solicitudes de uno y nueve ítems, con el fin de que exista una variabilidad en la cantidad de artículos solicitados y permitan evaluar de mejor manera la extracción de la información con los distintos modelos de lenguaje.

10 CAMISA DESGASTE SC108494K1/5 ANDRITZ	
CAMISA DESGASTE SC108494K1/5 ANDRITZ	
% IPI (Only for Brazilian suppliers)	
% PIS (Only for Brazilian suppliers)	
% COFINS (Only for providers in Brazil)	
% ICMS (Solo para proveedores de Brasil)	
% BASE DE CÁLCULO ICMS (Solo para proveedores de Brasil)	
Valor ICMS ST (Solo para proveedores de Brasil)	
VAT	
NCM (Only for Brazilian suppliers)	
Payment condition (Only for Brazilian suppliers)	
Material Code	000000000080070140 CAMISA DESGASTE SC108494K1/5 ANDRITZ
Plant	2003 Pulp Planta Sta. Fe
Precio	
Quantity	2 each
Discount Amount	
Surcharge Amount	
Subtotal	$0.00 CLP
Referencia	
¿Se cotiza exactamente lo mismo solicitado en el texto del material? (Si marca la opción "SI" y se entrega un material diferente o de otra marca está sujeto a devolución. Con Opción "NO" se debe adjuntar ficha técnica y el detalle de las diferencias)	

Figura 4.1: Ejemplo de información de productos en formato Word

Item Description	Unit	Quantity	Delivery Date	Your Quoted Price	Your Delivery Date
001 CPLG.SHFT JAW 55MM ANDRITZ 131713714 Arauco's material code 1644556	EA	1,00	20.07.2025	_____	_____

COUPLING SHAFT
TYPE:JAW
BORE:55MM
KEYWAY:*****
MATERIAL:STEEL
OUTER DIAMETER:132MM
SIZE:*****
LENGTH:*****
USAGE:TRANSPORTADOR DE CADENA DE ALIMENTACIÓN DE CHIPPER
MANUFACTURER:ANDRITZ
PART:131713714
MANUFACTURER:ANDRITZ
ITEM:703007357

Comprador: Mn.Bs- K.Medina

Figura 4.2: Ejemplo de información de productos en formato PDF

4.7 Arquitectura general de la solución

La solución propuesta se basa en una arquitectura modular e integrada, diseñada para automatizar el flujo de procesamiento de solicitudes y extracción de información desde documentos. La estructura principal incluye:

- **Power Automate**: Actúa como coordinador principal del sistema, encargándose de gestionar los eventos disparadores, transferir documentos entre plataformas y coordinar los distintos procesos.

- **SharePoint**: Funciona como repositorio central, permitiendo el almacenamiento estructurado de carpetas, documentos y registros en listas. Además, proporciona la interfaz de consulta y visualización para los usuarios.

- **Python**: Cumple el rol de procesador intermedio, ejecutando scripts para gestionar la integración con modelos de lenguaje y estructurar información en documento Excel.

- **Programador de tareas**: Automatiza la ejecución periódica de los scripts en Python, a partir de los archivos que llegan a la carpeta llamada ENTRADA almacenada en OneDrive.

- **OneDrive**: Se utiliza como puente de comunicación entre Power Automate, SharePoint y el entorno local del programador de tareas, permitiendo una sincronización de archivos.

- **Modelos de lenguaje**: La extracción de información se realiza mediante APIs de modelos Gemini y GPT, y de manera local con modelos Mistral y LLaMA, que permiten estructurar los datos de forma inteligente.

La Figura 4.3 presenta la arquitectura general teórica de la solución automatizada, la cual detalla la interacción entre los distintos módulos descritos anteriormente. El flujo contempla el paso de los documentos desde su recepción inicial hasta su procesamiento mediante modelos de lenguaje y su posterior almacenamiento estructurado para la fácil visualización y comprensión de los analistas. Esta representación gráfica permite visualizar de forma clara cómo se conectan las herramientas empleadas y cómo fluye la información entre ellas.

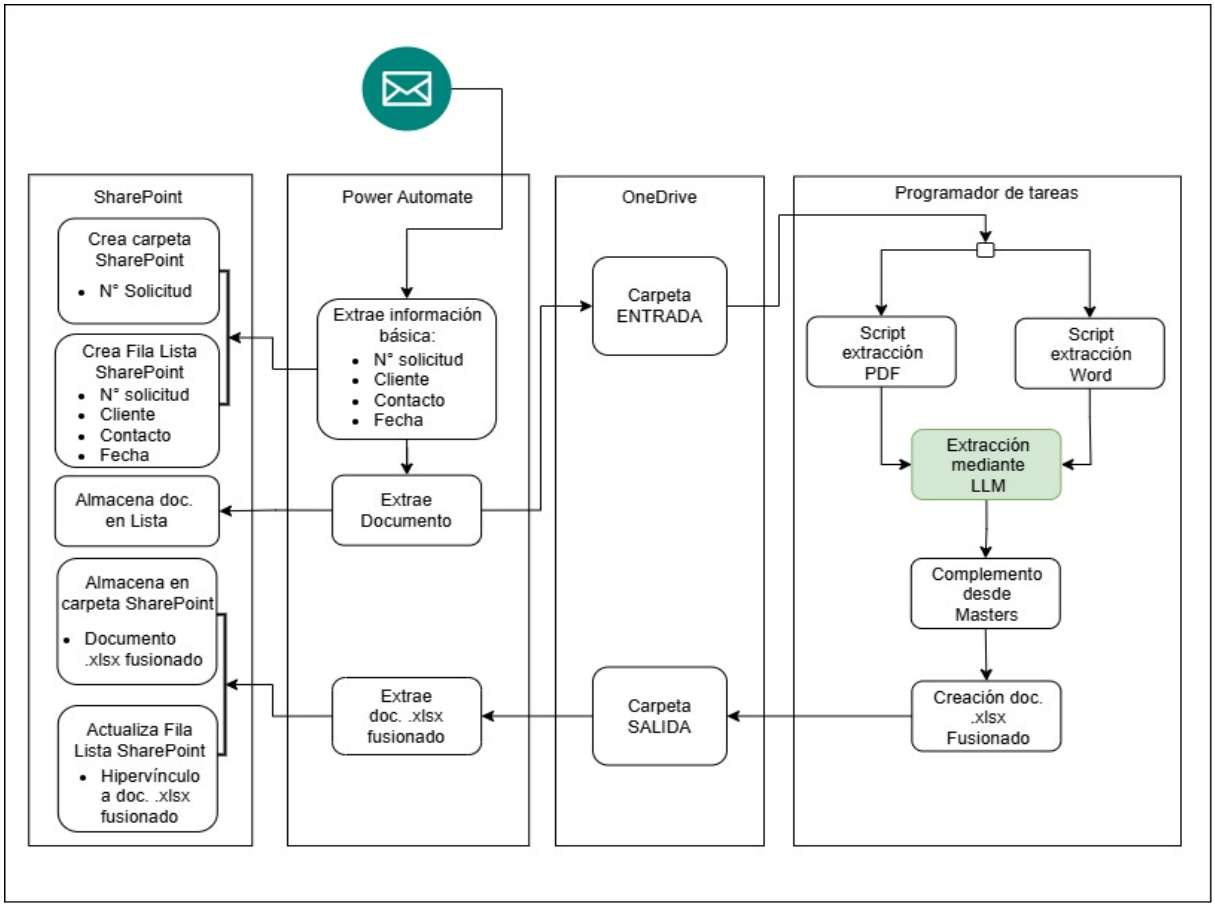


Figura 4.3: Diagrama de flujo del proceso automatizado

4.8 Flujo automatizado en Power Automate

Para automatizar la recepción y organización de solicitudes, se diseña un flujo en Power Automate que actúa como coordinador del proceso. Este flujo se activa de forma automática cuando se detecta la llegada de un nuevo correo electrónico con características predefinidas, permitiendo almacenar la información para su posterior procesamiento.

El flujo implementado se compone de las siguientes etapas:

1. **Detección de correo entrante:** El flujo se activa cuando se recibe un correo en Outlook, donde se lee el contenido del correo y se establece si el motivo de este es una solicitud de cliente, lo que permite filtrar los mensajes relevantes y excluir notificaciones o correos no relacionados.

2. **Extracción de datos del correo:** Se extrae el cuerpo del mensaje y los archivos adjuntos. Esta información contiene la solicitud técnica, listados de piezas y especificaciones.

3. **Almacenamiento en SharePoint:** Los archivos adjuntos son almacenados automáticamente en una carpeta específica de SharePoint creada con el nombre del número de solicitud. Además, se crea un nuevo ítem en una lista de seguimiento, con los siguientes campos:

    - Número de solicitud
    - Cliente
    - Fecha de recepción
    - Estado (por defecto: “Iniciar”)
    - Contacto Cliente
    - Hipervínculo al documento almacenado

4. **Preparación para el procesamiento local:** El archivo adjunto también es copiado a la carpeta “ENTRADA” de OneDrive, que está sincronizada con el equipo local del analista. Esto permite que el Programador de tareas de Windows detecte el nuevo archivo y active el script en Python.

5. **Actualización del ítem tras el procesamiento:** Una vez que el script genera el archivo Excel procesado en la carpeta “SALIDA” de OneDrive, un flujo de Power Automate lo detecta y actualiza el ítem correspondiente en la lista de SharePoint. Se agrega un hipervínculo al nuevo archivo Excel, lo que permite a los analistas acceder directamente al resumen extraído.

Este flujo automatizado permite centralizar y trazar el ciclo completo de procesamiento, desde la llegada de la solicitud hasta la disponibilidad del archivo estructurado, reduciendo la intervención manual y los tiempos de respuesta.

4.9 Ejecución del script Python

Se desarrolla un script en Python (Monitor.py) se puede visualizar en Anexo A.1, que se ejecuta automáticamente mediante el Programador de Tareas de Windows, como se describe en el diagrama de la Figura 4.4. Este script monitorea la carpeta “ENTRADA” de OneDrive cada cinco minutos, al detectar un nuevo archivo, espera que finalice su descarga. Posterior a eso, el script procesa con Arauco.py si el documento es PDF, y con CMPC.py si es Word (.docx). Para visualizar los scripts Arauco.py y CMPC.py consultar el Anexo A.2 y A.3, respectivamente.

Ambos scripts están diseñados para extraer información estructurada desde documentos mediante un modelo de lenguaje, a través de una API. La información extraída se almacena inicialmente en un archivo Excel momentáneo, como se muestra en la Tabla 4.3.

Posteriormente, como se muestra en la Tabla 4.4, esta información es complementada automáticamente con datos históricos provenientes de archivos maestros, incluyendo detalles como grupo de producto, vendedor asignado, origen, fecha de la última venta y el nombre del archivo maestro del cual se extrajo la información.

El procesamiento automático de los documentos se realiza utilizando bibliotecas como pdfplumber y python-docx para la lectura de texto, también se utiliza pandas y openpyxl para el manejo de datos y generación del archivo Excel final. Con todos estos datos integrados, se genera un archivo Excel final que se guarda en la carpeta “SALIDA”. Finalmente, el documento original procesado se traslada a la subcarpeta “PROCESADOS”, con el fin de evitar relecturas.

La automatización completa permite que la ejecución sea autónoma, confiable y sin necesidad de intervención manual.

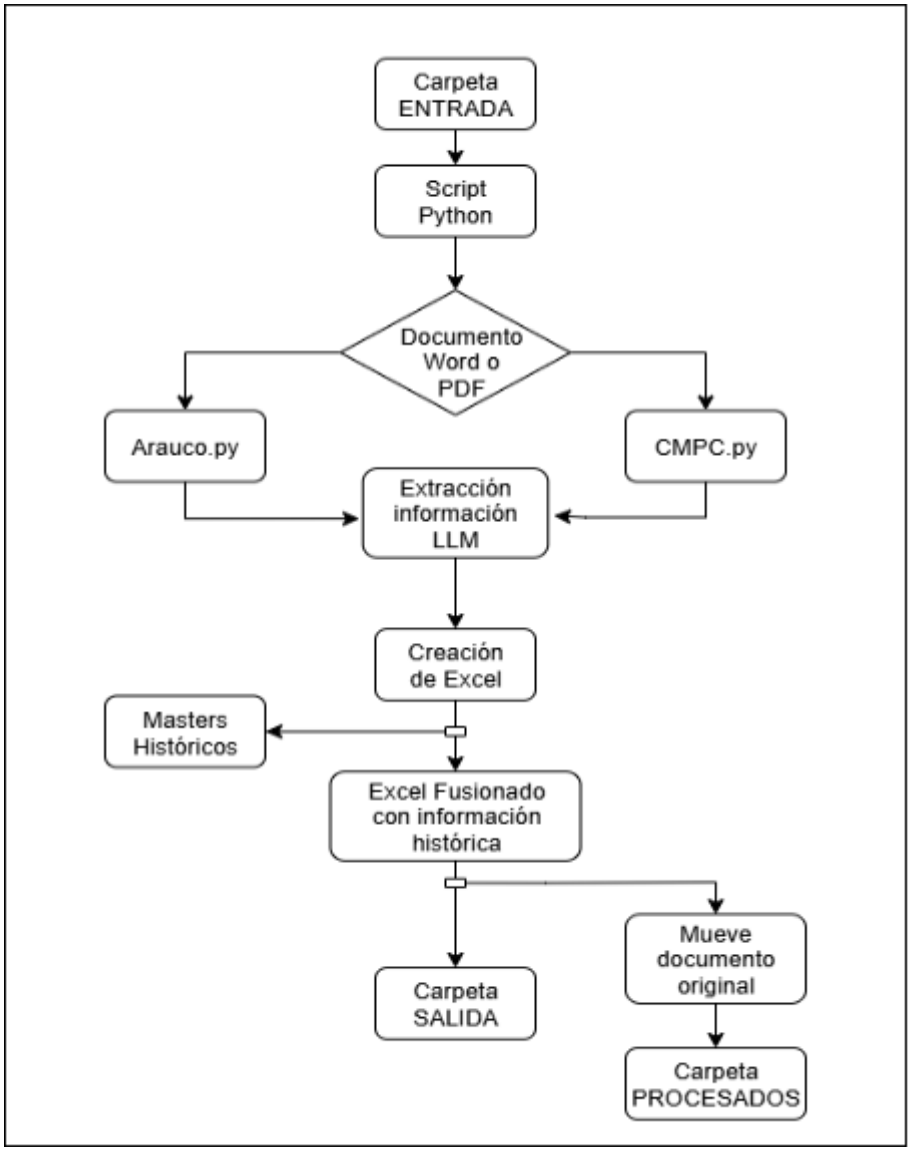


Figura 4.4: Diagrama de flujo de Script de Python

Tabla 4.3: Ejemplo información extraída desde documentos de solicitudes

Nombre del elemento	Número de parte	Código de material	Cantidad	Planta	Fabricante	Características adicionales
SELLO	131001167	12043616	4	2003 Pulp Planta Sta. Fe	ANDRITZ	800X840X18 SINGLE RADIAL SPLINT-RADIAL SHAFT VITON

Tabla 4.4: Ejemplo información extraída de archivos maestros

Product Group	Sent to	Origin	Date	Coincidencia	Archivo Maestro	Fila Excel Maestro	Fuente
PKF Fiberline	Ismo Hakkarainen	Savonlina	2025-03-07	Sí	MaestroAngela.xlsx	3621	Angela

4.10 Integración con modelos de lenguaje vía API

Una API se define como un contrato entre diferentes componentes de software, que permite que aplicaciones o sistemas interactúen entre sí sin necesidad de conocer los detalles internos de su implementación. Esto permite que desarrolladores externos accedan a funciones específicas de un sistema de forma controlada (Iqbal, 2023).

Iqbal menciona como ejemplo la aplicación en mapas turísticos, donde los desarrolladores integran Google Maps API junto con sensores de ubicación del dispositivo móvil para ofrecer funcionalidades útiles a los usuarios. Este ejemplo demuestra cómo las APIs permiten construir aplicaciones robustas sin tener que desarrollar cada componente desde cero.

En este trabajo, cada documento se analiza enviando una solicitud al LLM a través de una API o vía local, utilizando el script desarrollado en Python. Esta solicitud contiene un prompt diseñado específicamente para guiar la extracción de información relevante de forma precisa.

Para garantizar que la consistencia del modelo ante entradas idénticas se evalúe de forma válida, todas las solicitudes son procesadas con la temperatura configurada en 0. Esto implica que el modelo debe generar la misma salida siempre que el input sea idéntico, eliminando cualquier aleatoriedad asociada al proceso de generación. La respuesta esperada por ítem del modelo debe incluir los siguientes campos estructurados y debería ser similar a la Figura 4.5:

- Nombre del elemento
- Número de parte
- Código de material
- Cantidad

- Planta
- Fabricante
- Características adicionales

Una vez obtenida la respuesta, el script organiza los datos extraídos, integrándolos automáticamente en una tabla dentro del archivo Excel final, que posteriormente es exportado a OneDrive.

```
<elementos>
    <elemento>
        <nombre>CJTO ANILLO O ANDRITZ</nombre>
        <número_parte>131813294</número_parte>
        <código_material>000000000079173587</código_material>
        <cantidad>1 each</cantidad>
        <planta>1801 Cartulinas Maule</planta>
        <fabricante>ANDRITZ</fabricante>
        <característica_adicional>SET O-RING</característica_adicional>
    </elemento>
</elementos>
```

Figura 4.5: Respuesta esperadas del modelo

4.11 Criterios de evaluación

Para evaluar el rendimiento de los modelos de lenguaje en tareas de extracción de datos, se definieron seis métricas clave, descritas a continuación:

Exactitud: Esta métrica permite medir si el modelo logró extraer correctamente cada uno de los campos solicitados. Se calcula como el número de campos correctamente extraídos dividido por el total de campos esperados, expresado en porcentaje. Es una medida objetiva y cuantificable de la precisión del modelo.

Percepción de exactitud: Corresponde a una evaluación subjetiva realizada por el analista, utilizando una escala Likert, en la que se asigna un puntaje según la precisión percibida de la extracción. Se utiliza una escala de 5 puntos, donde 1 representa una extracción deficiente y 5 una extracción satisfactoria. Esta métrica complementa la evaluación objetiva con una valoración cualitativa.

Consistencia: Esta métrica evalúa el grado en que el modelo de lenguaje entrega resultados similares cuando se le presenta el mismo input en múltiples ejecuciones. Una alta consistencia indica que el modelo es estable bajo condiciones controladas. Para este estudio se fijó la temperatura = 0 en todos los modelos, lo cual elimina la aleatoriedad del proceso. Para cada combinación de modelo, documento y técnica de prompting, se realizan tres ejecuciones independientes con el mismo input. Se considera como respuesta consistente aquella que es idéntica o muy similar en estructura y contenido en las tres ejecuciones.

Tokens generados: Esta métrica estima el costo computacional y la eficiencia del modelo. Se considera la suma de tokens de entrada y de salida por cada solicitud procesada. Su análisis resulta útil para comparar el impacto de distintas técnicas de prompting en la carga del modelo y su consumo de recursos.

Costos monetarios: Esta métrica cuantifica el gasto económico asociado al uso del modelo de lenguaje. Se calcula a partir de la cantidad total de tokens generados considerando entrada y salida, y el valor por token establecido por el proveedor del modelo. Esto permite comparar la eficiencia económica entre modelos y técnicas de prompting, y evaluar su viabilidad en función del presupuesto.

Tiempo de inferencia: Mide el desempeño del modelo en función del tiempo que demora. Se calcula como el tiempo, en milisegundos, que transcurre desde el envío del prompt hasta la recepción completa de la respuesta. Esta métrica es clave para evaluar la viabilidad del modelo en aplicaciones en tiempo real o procesos automatizados.

CAPÍTULO 5

5 Prototipo de implementación

En este capítulo se describe el prototipo desarrollado para la implementación práctica del sistema automatizado. El propósito es detallar la integración de las distintas herramientas utilizadas y mostrar cómo interactúan entre sí, con una solicitud real.

5.1 Descripción general del sistema

La Figura 5.1 presenta el diagrama general simplificado del flujo automatizado implementado para procesar y clasificar las solicitudes de clientes en el área de ventas. Este sistema integra herramientas como Power Automate, Python, Programador de tareas, OneDrive, SharePoint y modelos de lenguaje, permitiendo automatizar el proceso desde la recepción de un correo electrónico hasta la entrega de un archivo estructurado para revisión por parte del analista. El diagrama resume los principales componentes involucrados y sus interacciones, proporcionando una visión clara del funcionamiento del prototipo a nivel general.

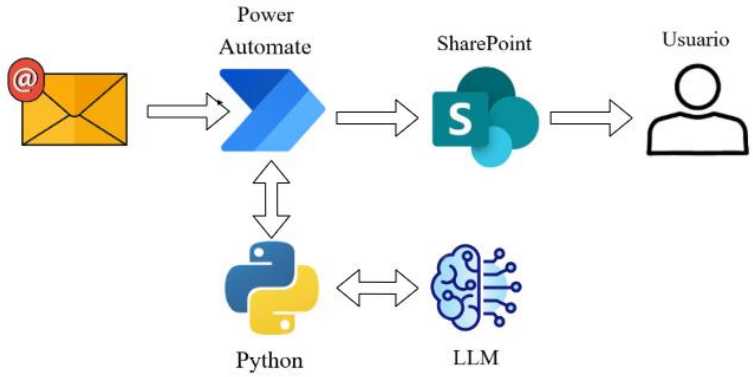


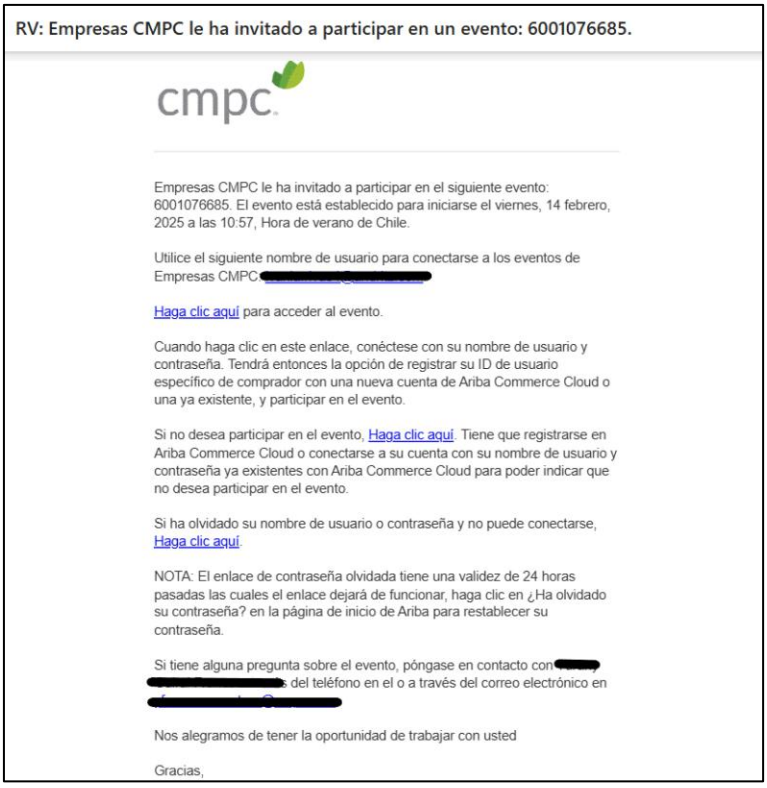
Figura 5.1: Diagrama general de funcionamiento del sistema automatizado.

5.2 Descripción detallada del flujo de trabajo

A continuación, se muestra la aplicación práctica del sistema automatizado a la llegada de la solicitud 6001076685.

Etapa 1: Recepción de solicitudes por correo

El proceso comienza con la llegada de un correo electrónico a la casilla para recepción de solicitudes, como ejemplo el que se muestra en la Figura 5.2, donde se ingresa la solicitud número 6001076685. El cuerpo de este correo contiene la información básica sobre el pedido como el número de solicitud, cuenta de correo electrónico a cuál llega, cliente y contacto cliente. Además de un archivo adjunto en el cual se incluyen los ítems a cotizar por parte del cliente.


RV: Empresas CMPC le ha invitado a participar en un evento: 6001076685.

cmpc

Empresas CMPC le ha invitado a participar en el siguiente evento: 6001076685. El evento está establecido para iniciarse el viernes, 14 febrero, 2025 a las 10:57, Hora de verano de Chile.

Utilice el siguiente nombre de usuario para conectarse a los eventos de Empresas CMPC: [redacted]

Haga clic aquí para acceder al evento.

Cuando haga clic en este enlace, conéctese con su nombre de usuario y contraseña. Tendrá entonces la opción de registrar su ID de usuario específico de comprador con una nueva cuenta de Ariba Commerce Cloud o una ya existente, y participar en el evento.

Si no desea participar en el evento, Haga clic aquí. Tiene que registrarse en Ariba Commerce Cloud o conectarse a su cuenta con su nombre de usuario y contraseña ya existentes con Ariba Commerce Cloud para poder indicar que no desea participar en el evento.

Si ha olvidado su nombre de usuario o contraseña y no puede conectarse, Haga clic aquí.

NOTA: El enlace de contraseña olvidada tiene una validez de 24 horas pasadas las cuales el enlace dejará de funcionar, haga clic en ¿Ha olvidado su contraseña? en la página de inicio de Ariba para restablecer su contraseña.

Si tiene alguna pregunta sobre el evento, póngase en contacto con [redacted] del teléfono en el o a través del correo electrónico en [redacted]

Nos alegramos de tener la oportunidad de trabajar con usted

Gracias,

Figura 5.2: Ejemplo de llegada de solicitud

Etapa 2: Activación del flujo en Power Automate

El correo entrante activa automáticamente un flujo en Power Automate. Este flujo extrae el archivo adjunto, lo almacena en una carpeta que crea en SharePoint con el nombre del número de solicitud, en este caso “6001076685”, como se visualiza en la Figura 5.3; y simultáneamente los copia a una carpeta llamada “ENTRADA” en OneDrive, la cual está sincronizada con el equipo local y el

Programador de tareas donde se ejecutará el procesamiento. Adicionalmente con la información básica del correo, crea una fila en la lista de visualización del analista como se observa en la Figura 5.4.

Figura 5.3: Creación de carpeta por número de solicitud

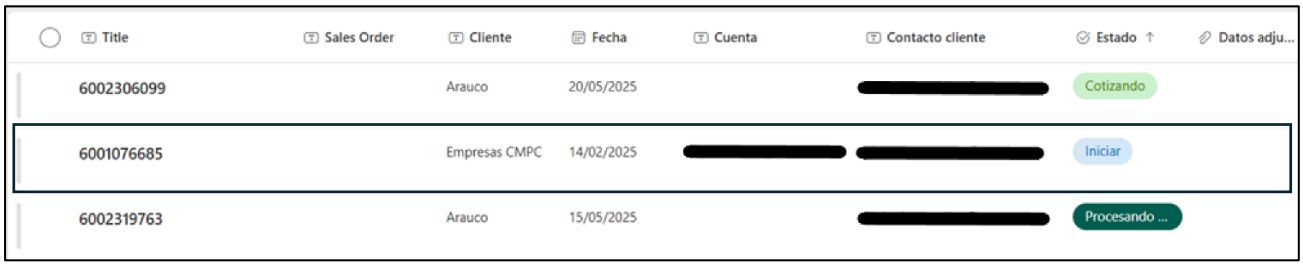


Figura 5.4: Creación de fila en lista de SharePoint de visualización

Etapa 3: Ejecución automática del script en Python

El Programador de Tareas de Windows monitorea la carpeta “ENTRADA” de OneDrive y al detectar un nuevo archivo, activa el script de procesamiento en Python llamado monitor.py que se encuentra en el Anexo A. Este script identifica el tipo de archivo (PDF o Word) y procede con el script de procesamiento correspondiente (CMPC.py o Arauco.py) usando el prompts establecido.

Etapa 4: Extracción de datos mediante modelos de lenguaje

El contenido del archivo es enviado al modelo de lenguaje mediante API. El modelo devuelve los datos clave solicitados: nombre del ítem, número de parte, código de material, planta, cantidad, fabricante y características adicionales, posteriormente Python los ordena en un documento Excel momentáneo, como se ve en la Figura 5.5.

Nombre del elemento	Número de parte	Código de material	Cantidad	Planta	Fabricante	Características adicionales
SELLO MODIFICADOR	131658144	12010506	6	2003 Pulp Planta Sta. Fe	ANDRITZ	EPDM 550MM
ANILLO	131700276	21043816	4	2003 Pulp Planta Sta. Fe	ANDRITZ	TIPO 32,2 X 3MM Q DIN 3771
CONJUNTO SELLO	TSNA 532 G	12786059	2	2003 Pulp Planta Sta. Fe	ANDRITZ	SET SELLOS 2 LABIOS
RESORTE	131002983	45003615	16	2003 Pulp Planta Sta. Fe	ANDRITZ	5X25X50 DIA INT 20+0,2MM SS2343
RUEDA	132300998	41480733	76	2003 Pulp Planta Sta. Fe	ANDRITZ	DIAMETRO 200MM
SELLO	131001167	12043616	4	2003 Pulp Planta Sta. Fe	ANDRITZ	800X840X18 SINGLE RADIAL SPLINT-RADIAL SHAFT VITON
ANILLO	N/A	12786060	2	2003 Pulp Planta Sta. Fe	ANDRITZ	DIAMETRO EXTERNO: 30MM MARCAS SUGERIDAS: FIVEFLEX
CAMISA	131017254	80030250	4	2003 Pulp Planta Sta. Fe	ANDRITZ	DESGASTE D= 190 SF401745K4 MATERIAL DLX2101
CAMISA DESGASTE	SC108494K1/5	80070140	2	2003 Pulp Planta Sta. Fe	ANDRITZ	N/A

Figura 5.5: Extracción de datos desde documento adjunto

Etapa 5: Complemento con información histórica

Los datos extraídos son comparados automáticamente en el mismo script con archivos maestros previamente cargados, desde los cuales se agregan campos adicionales como grupo de producto, última fecha de venta, vendedor asignado, origen y ubicación en el Excel maestro, mostrados en la Figura 5.6.

Product Group	Sent to	Origin	Date	Coincidencia	Archivo Maestro	Fila Excel Maestro	Fuente
PKF FIBERLINE	ISMO HAKKARAINEN	SAVONLINNA	2025-02-19 00:00:00	Sí	MaestroAngela.xlsx	3582	Angela
PKF FIBERLINE	ISMO HAKKARAINEN	SAVONLINNA	2025-02-19 00:00:00	Sí	MaestroAngela.xlsx	3582	Angela
PKF FIBERLINE	ISMO HAKKARAINEN	SAVONLINNA	2025-03-07 00:00:00	Sí	MaestroAngela.xlsx	3621	Angela
PKF FIBERLINE	ISMO HAKKARAINEN	SAVONLINNA	2025-03-07 00:00:00	Sí	MaestroAngela.xlsx	3621	Angela
PKF DRYING	FELIX FUCHS	GRAZ	2025-04-02 00:00:00	Sí	MaestroAngela.xlsx	3676	Angela
PKF FIBERLINE	ISMO HAKKARAINEN	SAVONLINNA	2025-03-07 00:00:00	Sí	MaestroAngela.xlsx	3621	Angela
PKF FIBERLINE	ISMO HAKKARAINEN	SAVONLINNA	2025-01-20 00:00:00	Sí	MaestroAngela.xlsx	3510	Angela
PKF FIBERLINE	ISMO HAKKARAINEN	SAVONLINNA	2025-03-03 00:00:00	Sí	MaestroAngela.xlsx	3607	Angela
PKR CHEM SYST	SEIJA MANNINEN	SAVONLINNA	2025-01-20 00:00:00	Sí	MaestroAngela.xlsx	3540	Angela

Figura 5.6: Complemento de datos extraídos desde archivos maestros

Etapa 6: Generación del archivo Excel final

Con la información combinada, se genera un archivo Excel estructurado, el cual se guarda automáticamente en la carpeta “SALIDA” de OneDrive. Este archivo contiene tanto los datos extraídos, como los campos complementados desde archivos maestros. Adicionalmente, para evitar relectura del archivo en la carpeta “ENTRADA”, después de ser procesado es removido y enviado a otra carpeta llamada “PROCESADOS”.

Etapa 7: Actualización en SharePoint

Una vez generado el Excel final, Power Automate detecta el archivo en la carpeta “SALIDA” y actualiza automáticamente el ítem correspondiente en la lista de seguimiento en SharePoint. Se agrega un hipervínculo al archivo generado, lo que permite a los analistas revisar los resultados directamente desde la plataforma.

Etapa 8: Revisión por parte del usuario

Finalmente, el analista de ventas accede a la lista de SharePoint, visualiza el registro de la solicitud o descarga el archivo Excel ya procesado y complementado, listo para comenzar la elaboración de la cotización, como se observa en la Figura 5.7. Este paso cierra el ciclo automatizado de procesamiento, reduciendo el tiempo de revisión y mejorando la trazabilidad.

Figura 5.7: Visualización de lista con hipervínculos

CAPÍTULO 6

6 Resultados

Esta sección presenta los resultados obtenidos a partir de la evaluación de distintos modelos de lenguaje aplicados a la tarea de extracción de información desde documentos utilizados por el área de ventas de Andritz Chile. Se analiza el desempeño de cada modelo bajo condiciones controladas, utilizando el mismo conjunto de documentos y evaluando métricas previamente definidas en el capítulo 4.

Los resultados son organizados y registrados en un archivo Excel, el cual se utiliza como base para construir visualizaciones y análisis comparativos. Para facilitar la interpretación y exploración de los datos, estos son analizados mediante Power BI, permitiendo generar gráficos y resúmenes visuales que destacan el desempeño de los modelos y técnicas evaluadas.

6.1 Resultados generales

En la Figura 6.1 se presenta el promedio de exactitud, percepción de exactitud y consistencia para cada combinación de modelo de lenguaje y técnica de prompting evaluada. El gráfico compara estos tres indicadores, permitiendo visualizar el rendimiento de cada configuración y detectar cuáles combinaciones ofrecen resultados más estables y precisos.

Se observa que técnicas como Ejemplos (few-shot), Formato Estructurado y Uso de Roles tienden a entregar resultados más consistentes y precisos, especialmente cuando se combinan con modelos de mayor capacidad. Estas combinaciones destacan por su equilibrio entre rendimiento técnico y percepción subjetiva. Por otro lado, la técnica Chain of Thought muestra un desempeño más irregular, con resultados bajos en ciertas combinaciones, especialmente con modelos de menor capacidad. Esto sugiere que no todas las técnicas se adaptan de igual manera a todos los modelos, y que su efectividad depende en gran medida de la interacción entre ambos factores.

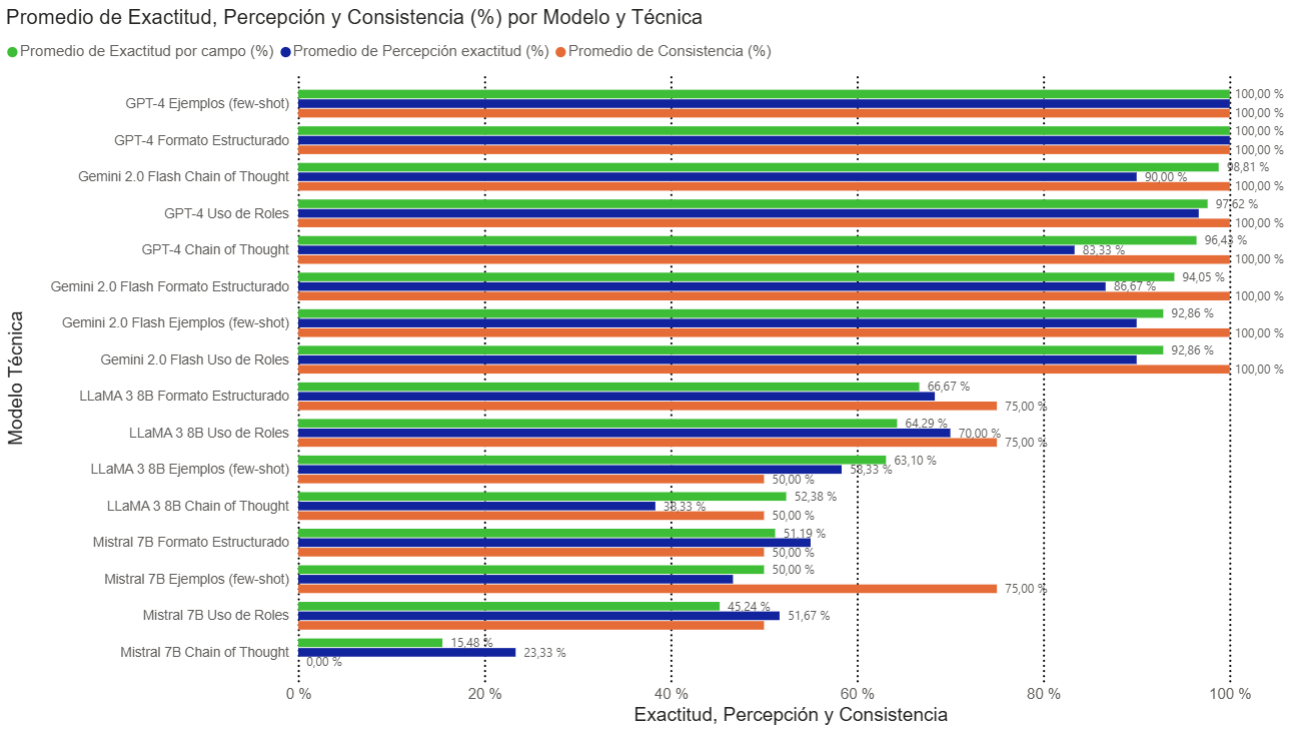


Figura 6.1: Desempeño promedio de modelos de lenguaje y técnica

6.2 Comparación por modelos de lenguaje

6.2.1 Desempeño de modelos

El primer análisis se centra en comparar el desempeño general de los cuatro modelos de lenguaje evaluados: GPT-4, Gemini 2.0 Flash, Mistral 7B y LLaMA 3 8B, considerando el promedio de sus resultados a través de las distintas técnicas de prompting. Esta comparación busca identificar cuál de ellos ofrece un mejor equilibrio entre precisión, eficiencia y costo en el contexto de extracción de información desde documentos semiestructurados.

Como se observa en los resultados de la Tabla 6.1, GPT-4 tiene el rendimiento más alto en exactitud (98,51 %) y percepción de exactitud (4,75 de 5), junto con una consistencia perfecta (100 %). Además, este modelo tiene un costo promedio por instancia de $0,10293 y un tiempo de respuesta de aproximadamente 19 segundos.

Gemini 2.0 Flash, por su parte, tiene una exactitud promedio de 94,64 %, percepción subjetiva de 4.46/5, consistencia estructural del 100%, con el menor tiempo de inferencia (4.613 ms) y un costo promedio por instancia de $0,00047.

LLaMA alcanza un 61,61 % de exactitud, con una percepción subjetiva de 2.94/5 y consistencia de 62,5 %. Es el modelo con mayor tiempo de inferencia promedio (156760 ms), pero el costo promedio más bajo por instancia (0.00015 USD). En tanto, Mistral muestra una exactitud global (40,48 %), tiempos de respuesta de 110 segundos promedio, una exactitud percibida de 2.21 de 5, costo monetario de 0.00021 USD promedio por instancia y la más baja consistencia (43.75%), aunque es en promedio el modelo que menos tokens genera por instancia.

Estos dos últimos modelos pueden utilizarse gratuitamente mediante la plataforma Ollama de forma local, sin generar costos económicos directos. Sin embargo, si se accede a estos mismos modelos a través de plataformas en línea como Groq, Fireworks o Replicate, sí implican un costo asociado por token procesado con los valores antes mostrados en la Tabla 4.2. En la Tabla 6.1 se muestran los valores considerando el uso de estas plataformas.

Tabla 6.1: Resultado métricas de evaluación por modelo de lenguaje

Modelo	Exactitud	Percepción de exactitud (1-5)	Consistencia	Tokens	Costo (USD)	Tiempo de inferencia (ms)
GPT-4	98,51%	4,75	100,00%	2795,63	0,10293	18802,88
Gemini 2.0 Flash	94,64%	4,46	100,00%	2784,08	0,00047	4613,77
LLaMA 3 8B	61,61%	2,94	62,50%	2683,75	0,00015	156760,42
Mistral 7B	40,48%	2,21	43,75%	2575,67	0,00021	110932,98

La Figura 6.2 presenta de forma gráfica los resultados descritos anteriormente, permitiendo visualizar y comparar el desempeño de cada modelo en términos de exactitud, percepción de exactitud y consistencia. Por su parte, la Figura 6.3 muestra la relación entre la exactitud promedio por campo y el tiempo de inferencia promedio de cada modelo, lo que permite analizar la eficiencia entre precisión y velocidad de procesamiento. En la Figura 6.4 se observa la comparación entre la cantidad promedio

de tokens generados y el tiempo de inferencia promedio, lo que entrega información sobre la eficiencia computacional frente a la carga de procesamiento.

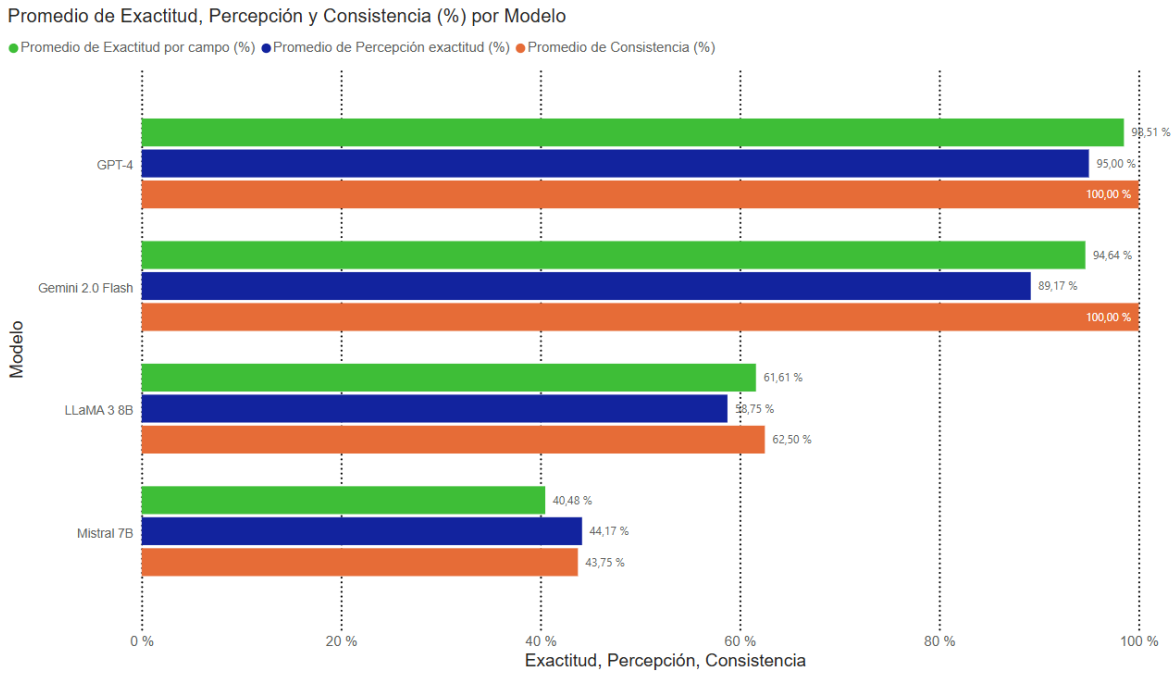


Figura 6.2: Desempeño promedio de modelos de lenguaje

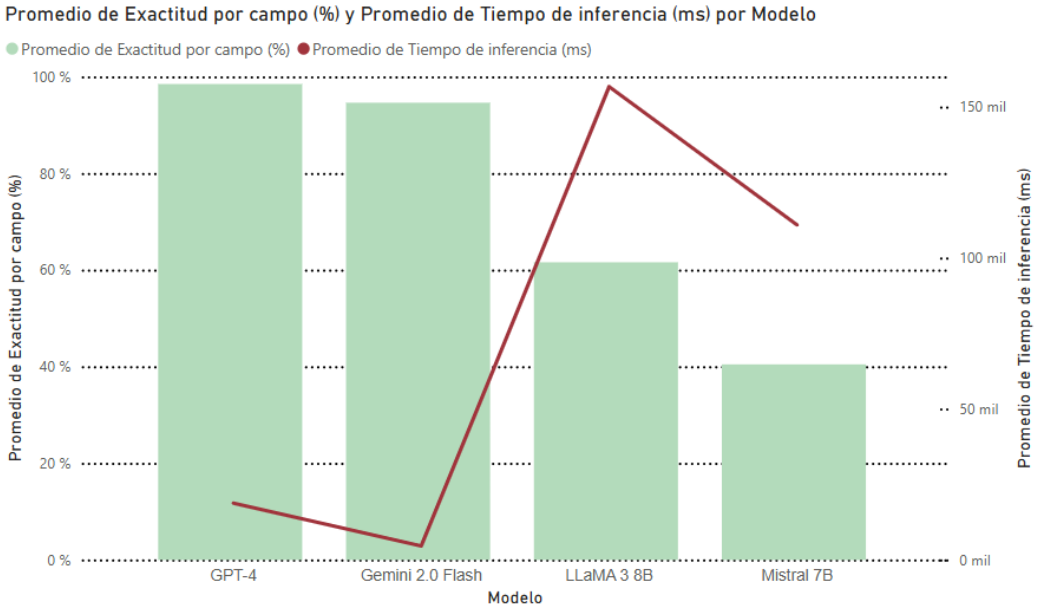


Figura 6.3: Comparación de exactitud y tiempo de inferencia promedio por modelo

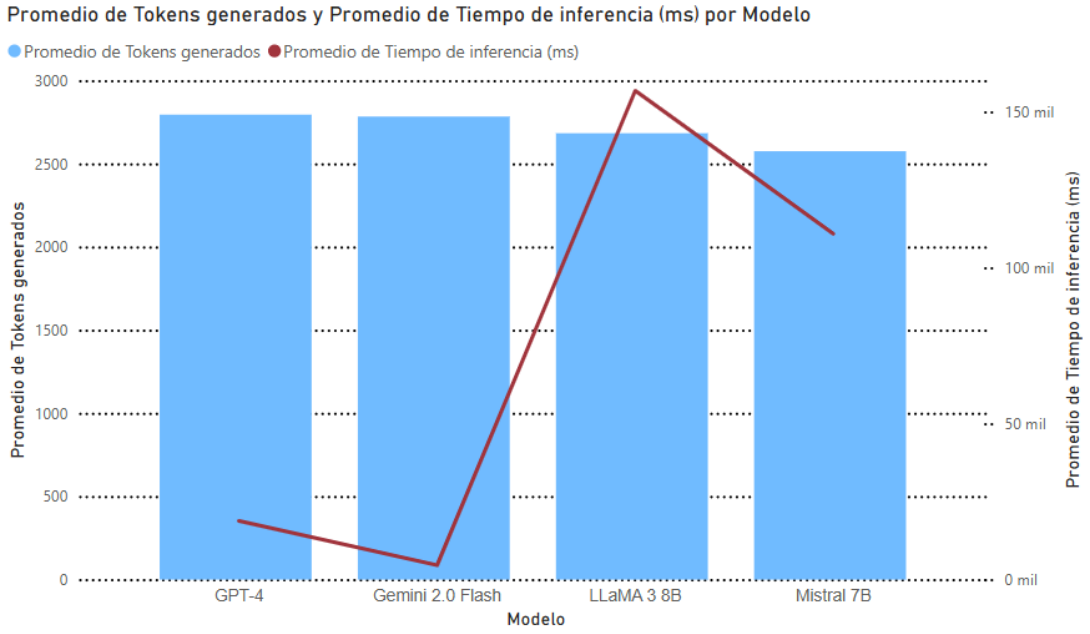


Figura 6.4: Comparación de tokens generados y tiempo de inferencia promedio por modelo

6.2.2 Comparación de costos por modelo

Esta sección presenta los valores promedio de costo monetario por instancia asociados al uso de cada modelo de lenguaje, según lo resumido en la Tabla 6.2. El modelo GPT-4 registra el valor más alto con un promedio de $0,10293 por instancia, mientras que Gemini 2.0 Flash muestra un costo promedio de $0,00047. Los modelos LLaMA 3 8B y Mistral 7B, al ser ejecutados localmente a través de Ollama, no generan un costo económico directo por inferencia. Sin embargo, para efectos comparativos, se consideraron valores de referencia basados en plataformas públicas como Groq para LLaMA 3 8B y Replicate para Mistral 7B, con costos promedios por instancias de $0,00015 y $0,00021, respectivamente.

La Figura 6.5 presenta la estimación del costo total acumulado para 1.000 instancias procesadas por cada modelo. Esta visualización permite dimensionar las diferencias en términos de costos monetarios ante escenarios de uso extensivo. Los resultados muestran que GPT-4 supera los 100 dólares, y los

modelos restantes, no superan los $0,50, lo que entrega información importante para decisiones futuras relacionadas con eficiencia económica y escalabilidad.

Tabla 6.2: Promedio de costo monetario por consulta

Modelo	Promedio de Costo monetario (USD)
GPT-4	0,10293
Gemini 2.0 Flash	0,00047
Mistral 7B	0,00021
LLaMA 3 8B	0,00015

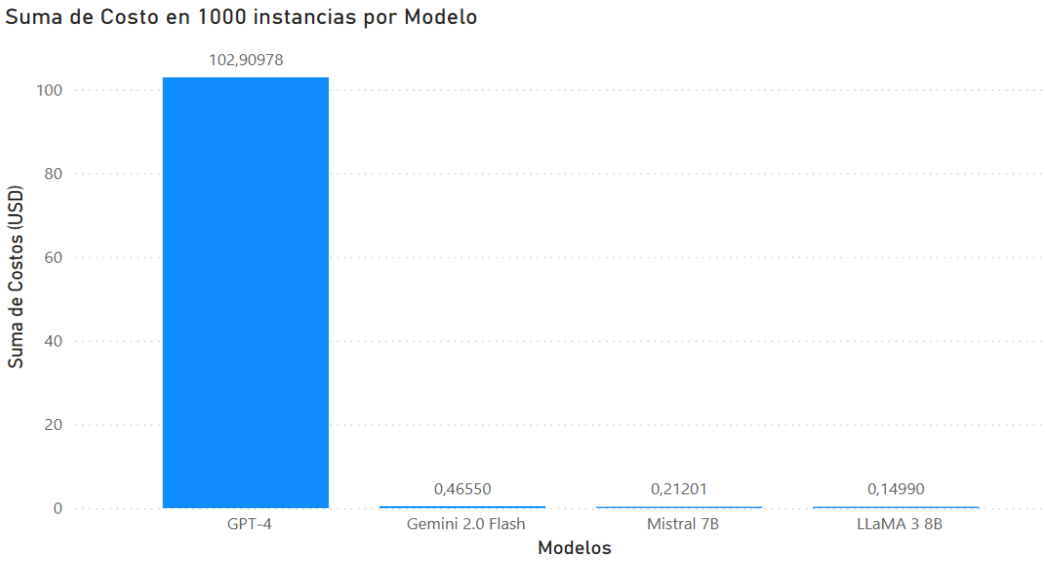


Figura 6.5: Comparación de costos de 1000 instancias por modelo

6.3 Comparación por técnica de prompting

Como se muestra en la Tabla 6.3, las técnicas de Formato Estructurado, Ejemplos (few-shot) y Uso de Roles alcanzan valores similares en cuanto a exactitud, con un rango que oscila entre el 75 % y el 78 %. En cuanto a la percepción de exactitud, las valoraciones promedio se mantienen cercanas a 3,8 en una escala Likert. La consistencia también muestra un comportamiento estable en estas técnicas, con un 81,25 % en todos los casos. En la Figura 6.6 se observan visualmente estos resultados.

Respecto al uso de recursos como se observa en la Figuras 6.7, el número de tokens generados se mantiene relativamente constante entre las tres técnicas mencionadas, al igual que el costo en dólares. Por otro lado, los tiempos de inferencia, presentan algunas diferencias, destacando que la técnica con menor tiempo promedio es el Uso de Roles (54.617 ms).

La técnica Chain of Thought presenta un comportamiento diferente. Se observan valores más bajos en exactitud (65,77 %), percepción de exactitud (2,94) y consistencia (62,50 %). Además, esta técnica genera la mayor cantidad de tokens (2.869) como se observa en la Figura 6.8, tiene el mayor costo por inferencia y registra el tiempo de respuesta promedio más alto (108.155 ms).

Tabla 6.3: Resultado métricas de evaluación por técnica

Técnica	Exactitud	Percepción de exactitud	Consistencia	Tokens	Costo (USD)	Tiempo de inferencia (ms)
Formato Estructurado	77,98%	3,88	81,25%	2604,88	0,02555	55730,4
Uso de Roles	75,00%	3,85	81,25%	2636,13	0,02571	54617,75
Ejemplos	76,49%	3,69	81,25%	2728,67	0,02506	72606,77
Chain of Thought	65,77%	2,94	62,50%	2869,46	0,02743	108155,13

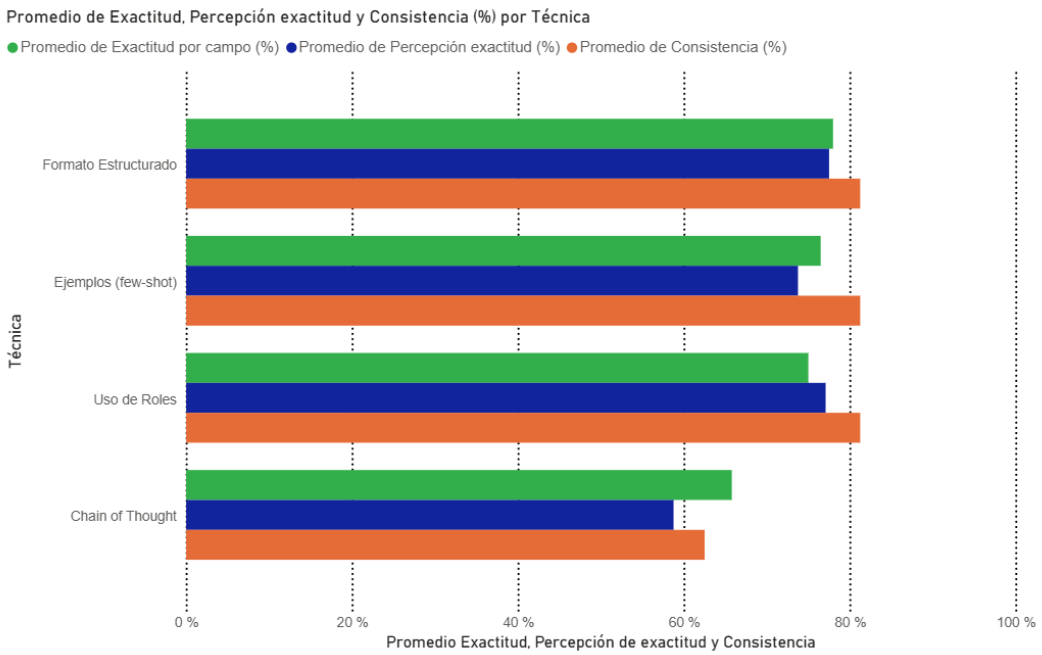


Figura 6.6: Comparación de desempeño promedio de técnica de prompting

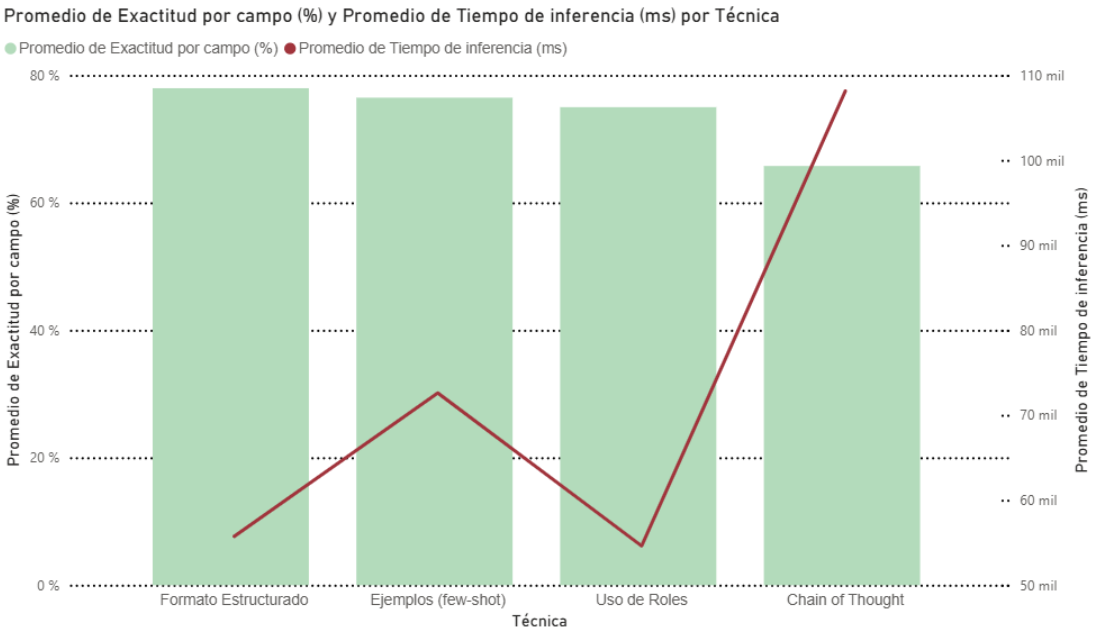


Figura 6.7: Comparación de exactitud y tiempo de inferencia promedio por técnica

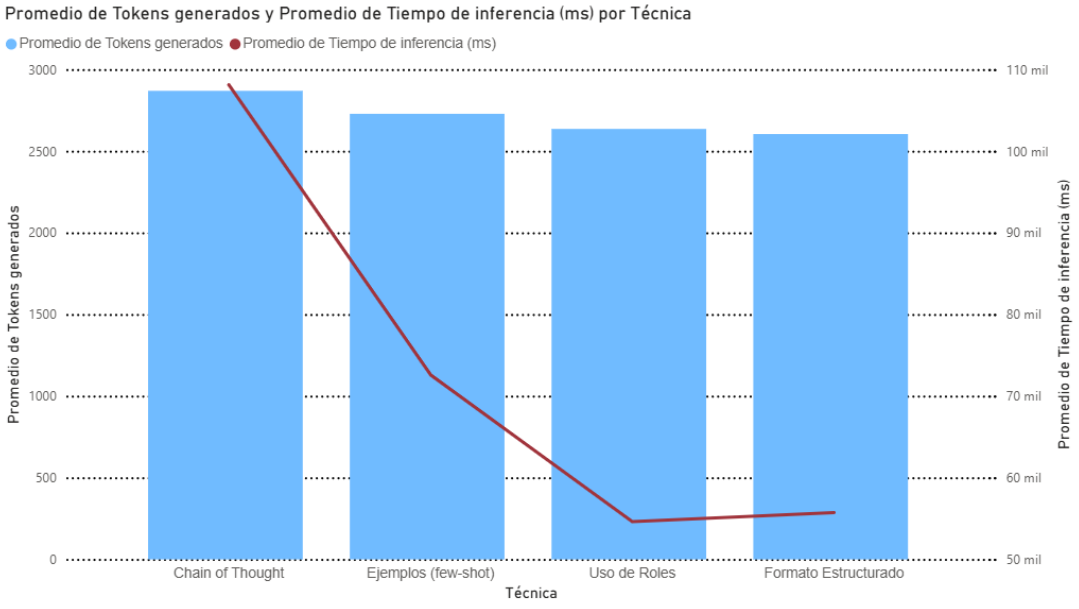


Figura 6.8: Comparación de tokens generados y tiempo de inferencia promedio por técnica

6.4 Tokens generados vs tiempo de inferencia

El análisis de la relación entre tokens generados y tiempo de inferencia, representado mediante el gráfico de la Figura 6.9, permite observar que los modelos que operan localmente requieren mayor tiempo de procesamiento, incluso con un volumen moderado de tokens. Mistral 7B y LLaMA 3 8B muestran tiempos de inferencia considerablemente altos en comparación con GPT-4 y Gemini en todas las técnicas estudiadas.

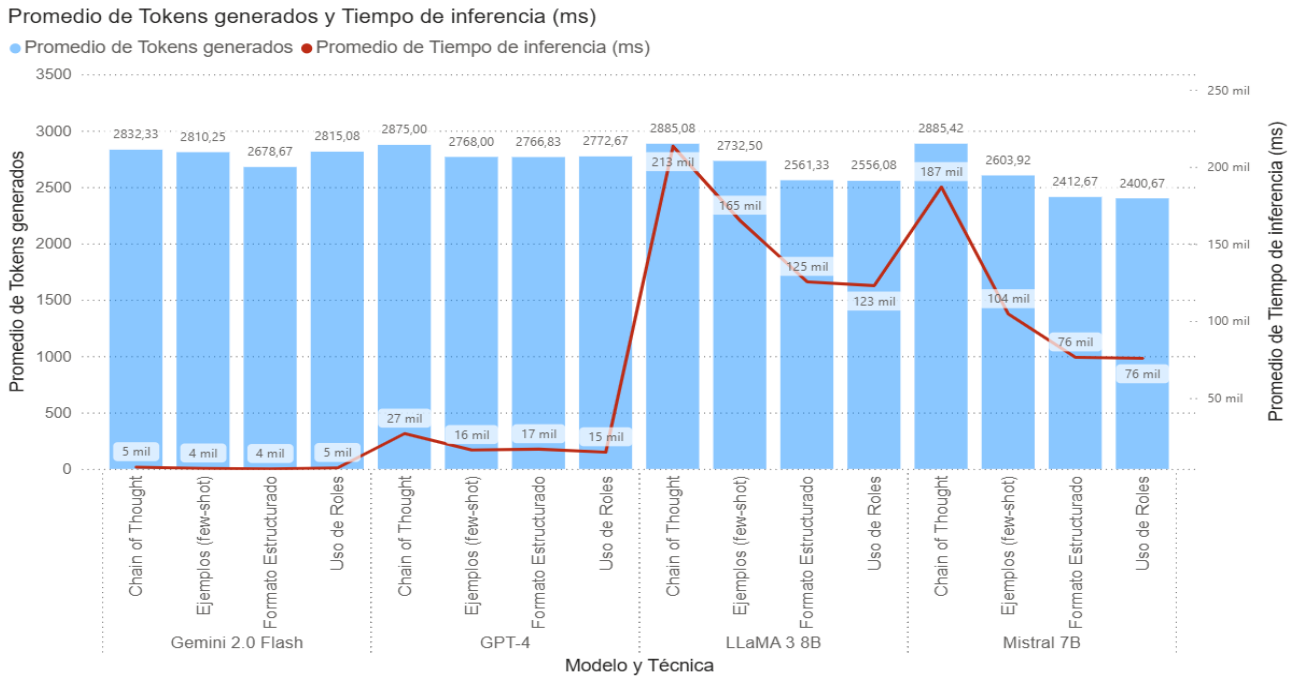


Figura 6.9: Comparación del promedio de tokens generados y tiempo de inferencia por modelo y técnica

6.5 Mejores resultados por métrica

La Tabla 6.4 presenta los resultados promedio en cada métrica evaluada respecto a cada modelo y técnica. Mientras, la Tabla 6.5 resume la combinación específica de modelo de lenguaje y técnica de prompting que alcanza los mejores resultados en cada métrica. Esto permite identificar buenas configuraciones en términos de precisión, eficiencia y costos.

En cuanto a la exactitud, se observa que GPT-4, utilizando las técnicas de Formato Estructurado y Ejemplo few-shot), logra un rendimiento perfecto, alcanzando un 100 % de campos correctamente

extraídos. Respecto a la percepción de exactitud, la mejor evaluación subjetiva fue nuevamente para GPT-4, con una puntuación máxima de 5/5 en las mismas técnicas anteriormente mencionadas. Esto indica no solo un cumplimiento objetivo de los campos, sino también una salida clara, precisa y confiable desde la perspectiva del analista. La consistencia también fue liderada por GPT-4 en conjunto con Gemini 2.0 Flash, ambos con un 100 % de estabilidad en sus respuestas frente a entradas iguales. Todas las técnicas de prompting favorecen este comportamiento para estos modelos.

En términos de tokens generados, el modelo que registra la menor cantidad es Mistral 7B, específicamente con la técnica de Uso de Roles alcanzando un promedio de 2400,67 tokens por instancia. Esto lo posiciona como el modelo más liviano en cuanto a consumo de recursos, claro que menor cantidad de tokens no quiere decir que sea la mejor respuesta.

En cuanto al costo monetario, los modelos LLaMA 3 y Mistral 7B destacan por su ventaja competitiva, pues al ser ejecutados localmente, su costo por inferencia es nulo (0 USD), lo que los hace atractivos desde el punto de vista presupuestario. Sin embargo, teniendo en cuenta el uso computacional, se estiman los valores como si fuera ejecutado por plataformas como Groq y Replicate y aun así, el modelo LlaMA 3 8B es el que tiene el menor costo por instancia promedio en técnicas de Uso de Roles y Formato Estructurado. Finalmente, el menor tiempo de inferencia lo obtiene Gemini 2.0 Flash, utilizando la técnica de Formato Estructurado, alcanzando un promedio de 4065,92 milisegundos.

Estos resultados permiten identificar buenas configuraciones según el objetivo de la aplicación: ya sea maximizar la precisión, minimizar los costos o reducir el tiempo de respuesta.

Tabla 6.4: Resultados promedio por modelo, técnica y métrica

Modelo/Técnica	Exactitud	Percepción	Consistencia	Tokens	Costo (USD)	Tiempo de inferencia (ms)
Chain of Thought						
Gemini 2.0 Flash	98,81%	4,5	100,00%	2832,33	0,0005	5242,58
GPT-4	96,43%	4,17	100,00%	2875	0,10875	27001,17
LLaMA 3 8B	52,38%	1,92	50,00%	2885,08	0,00017	213422,67
Mistral 7B	15,48%	1,17	0,00%	2885,42	0,0003	186954,08
Ejemplos (few-shot)						
Gemini 2.0 Flash	92,86%	4,5	100,00%	2810,25	0,00046	4403,5
GPT-4	100,00%	5	100,00%	2768	0,09942	16363,58
LLaMA 3 8B	63,10%	2,92	50,00%	2732,5	0,00015	165163,5
Mistral 7B	50,00%	2,33	75,00%	2603,92	0,00021	104496,5
Formato Estructurado						
Gemini 2.0 Flash	94,05%	4,33	100,00%	2678,67	0,00043	4065,92
GPT-4	100,00%	5	100,00%	2766,83	0,10148	16915,42
LLaMA 3 8B	66,67%	3,42	75,00%	2561,33	0,00014	125479
Mistral 7B	51,19%	2,75	50,00%	2412,67	0,00017	76461,25
Uso de Roles						
Gemini 2.0 Flash	92,86%	4,5	100,00%	2815,08	0,00048	4743,08
GPT-4	97,62%	4,83	100,00%	2772,67	0,10206	14931,33
LLaMA 3 8B	64,29%	3,5	75,00%	2556,08	0,00014	122976,5
Mistral 7B	45,24%	2,58	50,00%	2400,67	0,00017	75820,08

Tabla 6.5: Mejores resultados por métrica

Métrica	Mejor Modelo	Mejor Técnica	Valor Alcanzado
Exactitud por campo	• GPT-4	• Formato Estructurado • Ejemplo (few-shot)	100%
Percepción de exactitud	• GPT-4	• Formato Estructurado • Ejemplo (few-shot)	5/5
Consistencia	• GPT-4 • Gemini 2,0 Flash	• Formato Estructurado • Ejemplo (few-shot) • Chain of Thought • Uso de Roles	100%
Tokens generados	• Mistral 7B	•Uso de Roles	2400,67 tokens
Costo (USD)	• LLaMA 3 8B	• Formato Estructurado • Uso de Roles	0,00014 USD
Tiempo de inferencia	•Gemini 2,0 Flash	• Formato Estructurado	4065,92 milisegundos

6.6 Ejemplos cualitativos

Además de los resultados cuantitativos, se presentan ejemplos ilustrativos de las respuestas generadas por los modelos evaluados. Las Figuras 6.10 y 6.11 muestran salidas producidas por los modelos GPT y Gemini utilizando prompts con ejemplos. En ambos casos, las respuestas siguen un formato XML con todos los campos esperados presentes. Por otro lado, la Figura 6.12 muestra una respuesta generada por LLaMA 3, en la cual se observa la ausencia de etiquetas. En la Figura 6.13, correspondiente a Mistral 7B con técnica Chain of Thought, la salida contiene una estructura más narrativa y menos delimitada.

```
--- EJECUCIÓN 3 ---
<elementos>
    <elemento>
        <nombre>ARAND.PLANA M12 2,5MM 316 140HV A4 125</nombre>
        <número_parte>132284802</número_parte>
        <código_material>1720432</código_material>
        <cantidad>16,00</cantidad>
        <planta>Celulosa Planta Arauco Línea 3</planta>
        <fabricante>Andritz</fabricante>
        <característica_adicional>ARANDELA PLANA, A4 3506-1, DIN 125, ISO 7089</característica_adicional>
    </elemento>
    <elemento>
        <nombre>PERNO MÁQ HEX M12 1,75MM 65MM 316 A4-80</nombre>
        <número_parte>131950185</número_parte>
        <código_material>1760574</código_material>
        <cantidad>8,00</cantidad>
        <planta>Celulosa Planta Arauco Línea 3</planta>
        <fabricante>Andritz</fabricante>
        <característica_adicional>HEXAGONAL, AISI 316, ISO 4014</característica_adicional>
    </elemento>
    <elemento>
        <nombre>PERNO MÁQ HEX M16 2MM 90MM 304 A2-70 931</nombre>
        <número_parte>100002752</número_parte>
        <código_material>1772464</código_material>
        <cantidad>1,00</cantidad>
        <planta>Celulosa Planta Arauco Línea 3</planta>
        <fabricante>Andritz</fabricante>
        <característica_adicional>HEXAGONAL, AISI 304, DIN 13, ISO 4014, DIN 931</característica_adicional>
    </elemento>
```

Figura 6.10: Ejemplo salida modelo GPT-4

```
--- EJECUCIÓN 3 ---
```xml
<elemento>
  <nombre>ARAND.PLANA M12 2,5MM 316 140HV A4 125</nombre>
  <número_parte/>
  <código_material>1720432</código_material>
  <cantidad>16,00</cantidad>
  <planta>Arauco</planta>
  <fabricante/>
  <característica_adicional>ANDRITZ 132284802#</característica_adicional>
</elemento>
<elemento>
  <nombre>PERNO MÁQ HEX M12 1,75MM 65MM 316 A4-80</nombre>
  <número_parte>131950185</número_parte>
  <código_material>1760574</código_material>
  <cantidad>8,00</cantidad>
  <planta>Arauco</planta>
  <fabricante>ANDRITZ</fabricante>
  <característica_adicional/>
</elemento>
<elemento>
  <nombre>PERNO MÁQ HEX M16 2MM 90MM 304 A2-70 931</nombre>
  <número_parte>100002752</número_parte>
  <código_material>1772464</código_material>
  <cantidad>1,00</cantidad>
  <planta>Arauco</planta>
  <fabricante>ANDRITZ</fabricante>
  <característica_adicional>ISO 3506</característica_adicional>
</elemento>
```

Figura 6.11: Ejemplo salida modelo Gemini 2.0 Flash
```

```
--- EJECUCIÓN 2 ---
La información que se puede extraer del texto es la siguiente:

**Pedido 1**

* Material Code: 000000000012345678
* Plant: 2003 Pulp Planta Sta. Fe
* Price: [vacío]
* Quantity: 4 each
* Discount Amount: [vacío]
* Surcharge Amount: [vacío]
* Subtotal: $0.00 CLP
* Reference: [vacío]
* Requested Delivery Date: Sun, 15 Jun, 2025

**Información del material**

* Material Code: 000000000012345678
* Fabricante: ANDRITZ
* Nombre del elemento: Extrae la información del texto entregado.
* Características adicionales: D= 190 SF401745K4 MATERIAL DLX2101

**Pedido 2**

* Material Code: 000000000080030250
* Plant: 2003 Pulp Planta Sta. Fe
* Price: [vacío]
* Quantity: 4 each
* Discount Amount: [vacío]
* Surcharge Amount: [vacío]
* Subtotal: $0.00 CLP
* Reference: [vacío]
* Requested Delivery Date: Sun, 15 Jun, 2025
```

Figura 6.12: Ejemplo salida modelo LLaMA 3 8B

```
--- EJECUCIÓN 2 ---
 Este texto contiene una solicitud de cotización para productos en particular, como pernos y tuercas hechas por la empresa "ANDRITZ". Cada producto se de
scribe con su tipo, diámetro, rosca, longitud, material, especificación, grado de calidad y norma de fabricación. Además, hay un código de producto asign
ado a cada artículo en el sistema de inventario de la compañía. Al final del texto, también se menciona una fecha de envío para cada item.

   Es importante notar que no se especifica ninguna información sobre el precio unitario o total por cada producto. Esto sugiere que se requiere un preci
os personalizados basados en los acuerdos comerciales y volúmenes de compra que tenga la empresa solicitante con la firma proveedora.

   En general, esta solicitud de cotización es una manera eficiente para comunicar información detallada sobre los productos que se necesitan a un provee
dor. La respuesta a esta solicitud contendrá la oferta del precio y otros términos comerciales, como plazo de entrega y términos y condiciones generales.
```

Figura 6.13: Ejemplo salida modelo Mistral 7B

6.7 Resultados de la implementación práctica del prototipo

Además de los resultados obtenidos en la comparación experimental de modelos y técnicas, se evaluó el rendimiento del prototipo automatizado en un caso real, con el fin de contrastar directamente su desempeño frente al proceso manual realizado por un analista.

El tiempo de procesamiento de la solicitud 6001076685 es de 20,5 minutos, al ser procesada por el analista de manera manual para obtener la misma información entregada de manera automática, mientras que, al ser ejecutada mediante el prototipo automatizado, esta tarda 4,3 minutos, es decir el 21% del tiempo que utilizaría el analista. Así también, se debe tener en cuenta que el sistema automatizado comienza a trabajar al instante en que llega el correo con la solicitud, sin necesidad de esperar a que el analista revise la bandeja de entrada para comenzar con el proceso. Por tanto, en este punto, el tiempo de mano de obra ahorrado es la totalidad del tiempo que habría tardado el analista.

Finalmente, cabe destacar que, frente a un mayor volumen de ítems por solicitud, el sistema automatizado mantiene tiempos de procesamiento prácticamente constantes. En cambio, en el proceso manual cada ítem adicional aumenta proporcionalmente el tiempo requerido, lo que se traduce en un proceso menos eficiente.

CAPÍTULO 7

7 Discusión

Los resultados obtenidos permiten establecer diferencias claras entre los modelos de lenguaje evaluados. GPT-4 demuestra un desempeño superior en términos de precisión (98,51 %), percepción de exactitud (4,75/5) y consistencia (100 %), lo que lo posiciona como el modelo más robusto e ideal para tareas de extracción técnica estructurada. Su capacidad de interpretar correctamente estructuras complejas y mantener una salida estable hace que sea el más adecuado para entornos de estas características. No obstante, este rendimiento viene acompañado de un costo significativamente mayor, pues el valor promedio por instancia con GPT-4 fue de USD 0,10293, lo que representa un costo más de 219 veces superior al de Gemini 2.0 Flash, cuyo valor fue de USD 0,00047 promedio por instancia. Esta diferencia de costos es relevante al proyectar la implementación, donde la eficiencia económica se convierte en un factor determinante.

Gemini 2.0 Flash, por su parte, se posiciona como la mejor alternativa, al ofrecer un equilibrio muy competitivo entre precisión (94,64 %), tiempo de respuesta (4.613 ms) y bajo costo por uso, manteniendo además una consistencia perfecta (100 %). Es eficaz cuando se emplean técnicas estructuradas como prompts con etiquetas XML o ejemplos claros. Su rendimiento sostenido y su bajo impacto económico lo posicionan como una alternativa viable y escalable a la integración con otros sistemas empresariales. Si bien GPT-4 presenta una precisión levemente superior (98,51 %), la diferencia de 3,87 % en exactitud no parece justificar el enorme incremento en costos, considerando que una sola instancia con GPT-4 es aproximadamente un 22000% más costosa que con Gemini. Por tanto, en contextos donde se busca eficiencia sin comprometer la calidad, Gemini 2.0 Flash representa una opción altamente costo-efectiva.

En cuanto a los modelos ejecutados localmente, LLaMA 3 y Mistral 7B muestran limitaciones importantes en cuanto a exactitud y consistencia, aunque con la ventaja de operar sin costos por uso mediante vía local y con plena autonomía sobre los datos procesados. LLaMA 3 alcanzó una exactitud promedio del 61,61 %, mientras que Mistral 7B solo logró un 40,48 %, ambas muy por debajo de los valores que se obtienen con los modelos de paga, siendo además Mistral el modelo con peor desempeño en percepción de exactitud, y LLaMA 3 con el peor tiempo de inferencia promedio,

superando los 156.000 milisegundos. Adicionalmente, los resultados fueron más inestables y sensibles a la formulación del prompt, lo que restringe su aplicación en flujos automatizados sin ajustes adicionales.

Es importante señalar que, si bien los modelos locales pueden ejecutarse sin intermediarios, su funcionamiento depende de los recursos computacionales disponibles. A medida que se utilizan modelos de mayor tamaño o complejidad, como variantes superiores de LLaMA o Mistral con más parámetros, se requieren GPUs de alto rendimiento, mayor cantidad de memoria y una configuración técnica avanzada, lo que puede implicar una inversión considerable en infraestructura local.

El estudio muestra que el rendimiento de los modelos no depende exclusivamente de su arquitectura, sino también de su combinación con técnicas de prompting adecuadas y del entorno técnico en que se despliegan. El análisis evidencia que una estrategia de automatización efectiva requiere evaluar conjuntamente precisión, costos, velocidad, consistencia y capacidad técnica disponible.

Es importante destacar que todas las pruebas fueron realizadas con temperatura configurada en 0.0, asegurando un comportamiento determinista en los modelos de lenguaje. Esta configuración elimina la aleatoriedad asociada al proceso de generación, por lo que las diferencias observadas en las métricas de consistencia entre modelos no responden a variabilidad del azar, sino a limitaciones reales en su capacidad para producir respuestas estructuradas de manera estable. Este enfoque metodológico permite afirmar con mayor solidez que, modelos como GPT-4 o Gemini 2.0 Flash, presentan una superioridad técnica no solo en precisión, sino también en estabilidad, frente a modelos locales como Mistral 7B o LLaMA 3 8B.

7.1 Influencia de las técnicas de prompting

Las técnicas de prompting utilizadas en este estudio juegan un rol determinante en el rendimiento de los modelos evaluados. Los resultados evidencian que no basta con seleccionar un buen modelo, la calidad y estructura del prompt influyen directamente en la precisión, consistencia y utilidad de la respuesta.

Las estrategias más efectivas son aquellas que incluyen Formato Estructurado y Ejemplos concretos. Estas técnicas entregan una guía clara sobre el tipo de salida esperada, lo que facilita que modelos como GPT-4 y Gemini 2.0 Flash generen respuestas en formato XML correcto, con todos los campos

requeridos y en el orden adecuado. Además, estas técnicas presentan altos niveles de percepción subjetiva y consistencia, reduciendo errores de interpretación y facilitando el tratamiento posterior de las salidas en sistemas automatizados.

En cambio, técnicas como Chain of Thought, basadas en razonamiento paso a paso, resultan menos adecuadas para este tipo de tarea. Aunque pueden ser útiles en contextos donde se requiere razonamiento o reflexión, su aplicación en modelos como Mistral 7B o LLaMA 3 8B conduce a salidas extensas, poco estructuradas o con formato incorrecto. Esto se traduce en un mayor esfuerzo para su limpieza o integración, ya que el script no puede interpretar automáticamente y de buena manera si no encuentra etiquetas claras o una estructura fija para la extracción de datos, lo que compromete la eficiencia del flujo automatizado.

Asimismo, la técnica de Uso de Roles, que busca orientar el comportamiento del modelo asignándole una identidad o función, muestra un rendimiento medio-alto. Si bien ayuda a mejorar la calidad de las respuestas, no logra igualar los resultados obtenidos por los prompts que incluyen directamente la estructura XML deseada.

Cabe destacar que los modelos menos potentes son más sensibles a la técnica de prompting utilizada. En modelos como Mistral 7B, incluso técnicas bien formuladas generan salidas inconsistentes, lo que indica que la capacidad del modelo para interpretar y seguir instrucciones complejas también influye en la eficacia del prompt.

7.2 Alcance respecto a los objetivos del estudio

El estudio logra cumplir con sus objetivos principales: comparar modelos de lenguaje de última generación, evaluar distintas estrategias de prompting, y medir su desempeño en un proceso automatizado de extracción de datos en un contexto industrial real. Además, se diseña e implementa un flujo de trabajo funcional e integrado, combinando herramientas de automatización, procesamiento y visualización, lo que permite cubrir todas las etapas del proceso, desde la recepción del documento hasta la generación del archivo final.

Cabe señalar que los tiempos prácticos del prototipo reportados en la sección de resultados corresponden al modelo Gemini 2.0 Flash, ya que fue el elegido para la implementación. Este modelo

mostró un desempeño competitivo en las métricas comparativas y también demostró en la práctica una reducción evidente en los tiempos frente al proceso manual.

La evidencia empírica obtenida permite validar la viabilidad técnica de la automatización del proceso, demostrando que, al emplear modelos de lenguaje con suficiente capacidad y prompts adecuadamente estructurados, se puede alcanzar un alto nivel de precisión, consistencia y eficiencia en la extracción de información. El sistema automatizado es capaz de identificar ítems relevantes, completar todos los campos requeridos y generar salidas en formato XML y Excel sin intervención humana directa.

Además, el estudio permite identificar qué combinaciones de modelo y técnica son más efectivas, así como cuáles resultan insuficientes para una implementación práctica. Esta información es clave no solo para evaluar el desempeño desde un punto de vista técnico, sino también para orientar decisiones futuras respecto a la escalabilidad, sostenibilidad y costo de operación del sistema en un entorno como el de Andritz Chile.

CAPÍTULO 8

8 Conclusiones

Este proyecto de memoria de título permitió diseñar, implementar y evaluar un sistema automatizado para la clasificación, orden y distribución de solicitudes de clientes en el área de ventas de repuestos de Andritz Chile, integrando modelos de lenguaje y herramientas de automatización. A través de una arquitectura que combina Power Automate, SharePoint, Python, OneDrive y modelos como GPT-4, Gemini 2.0 Flash, LLaMA 3 8B y Mistral 7B, se logró estructurar un flujo funcional capaz de recibir documentos y extraer información clave automáticamente para los analistas, sin intervención manual.

Los resultados mostraron que es posible automatizar tareas complejas como la lectura e interpretación de solicitudes técnicas, siempre que se utilicen modelos adecuados y prompts bien diseñados. Se comprobó que no todos los modelos de lenguaje ofrecen el mismo rendimiento. Modelos comerciales como GPT-4 y Gemini 2.0 Flash demostraron una clara superioridad en precisión, velocidad y estabilidad. Sin embargo, esta mejora en el rendimiento viene acompañada de alzas notables en el costo, lo que abre el debate sobre la relación costo/beneficio de su uso.

En particular, Gemini 2.0 Flash destacó como una alternativa altamente costo-efectiva, alcanzando un rendimiento comparable al de GPT-4 pero con un costo notablemente inferior, lo que refuerza su potencial para implementaciones reales a gran escala.

8.1 Potencial del sistema propuesto

El sistema automatizado desarrollado en este estudio demostró que es viable aplicar LLMs para apoyar tareas operativas de lectura y análisis técnico. El prototipo permite reducir el tiempo destinado por los analistas a la revisión manual de correos, mejorar la trazabilidad de las solicitudes y disminuir la duplicación de esfuerzos.

Además, el sistema es escalable, ya que puede adaptarse a diferentes volúmenes de trabajo y ampliarse a nuevas categorías de productos o formatos de documentos. La integración de herramientas conocidas por la empresa como Outlook, SharePoint y OneDrive facilita su adopción por parte del personal, minimizando la resistencia al cambio y sin incurrir en gastos adicionales.

Este tipo de soluciones evidencia cómo los modelos de lenguaje pueden incorporarse en flujos operativos reales, aportando eficiencia sin necesidad de grandes cambios tecnológicos ni inversión adicional en infraestructura. Además, es posible utilizar plataformas que, a bajo costo, pueden ahorrar mucho tiempo en mano de obra.

8.2 Limitaciones del estudio

A pesar de los resultados positivos obtenidos, el estudio presenta una serie de limitaciones que se deben considerar al interpretar los hallazgos y pensar en su aplicación dentro de la empresa.

En primer lugar, esta evaluación se realizó utilizando un conjunto reducido de documentos históricos, considerando solo cuatro casos comunes del área de ventas de la empresa. Si bien, fueron seleccionados estratégicamente para incluir variabilidad en el formato y cantidad de ítems, el tamaño de la muestra puede no ser suficiente para alcanzar la robustez necesaria. Aumentar la cantidad de documentos y pruebas permitiría validar con mayor profundidad la estabilidad y adaptabilidad del sistema ante diferentes tipos de solicitudes.

En segundo lugar, los modelos locales (LLaMA y Mistral) fueron ejecutados en una infraestructura computacional restringida, lo que afectó los tiempos de cada instancia, sin reflejar el real rendimiento que podrían alcanzar bajo condiciones técnicas óptimas. Además, estos modelos de código abierto fueron utilizados con pesos preentrenados genéricos, sin aplicar adicionalmente procesos de *fine-tuning*[5] con datos propios de la empresa, lo que podría haber mejorado su precisión y adaptabilidad al contexto técnico de Andritz Chile.

Tercero, no se abordó la integración directa con plataformas empresariales como SAP Ariba o sistemas CRM, desde donde se generan muchas de las solicitudes reales. Esto impidió evaluar el comportamiento de la solución en un entorno productivo completamente automatizado e integrado, limitando su aplicación práctica inmediata.

[5] Fine-tuning o ajuste fino es una técnica de aprendizaje automático que consiste en tomar un modelo pre-entrenado y ajustarlo o aumentar su entrenamiento con un conjunto de datos específico.

Por otra parte, el uso de modelos de lenguaje trae consigo riesgos, como errores de interpretación, omisión de campos, generación de respuestas erróneas o desorden estructural. A pesar de configurar todos los modelos con temperatura igual a 0 para eliminar la aleatoriedad, se observó que algunos modelos fueron sensibles a la formulación del prompt, produciendo salidas incompletas o incorrectas. Este tipo de problemas puede comprometer la confiabilidad del sistema si no se implementan mecanismos de validación o supervisión humana.

Finalmente, la percepción subjetiva fue evaluada por una sola persona, lo que podría introducir sesgos. En futuras mediciones se debería considerar una evaluación de múltiples usuarios con distintos perfiles, para una mejor validación de la experiencia respecto de las salidas generadas por el sistema.

Estas limitaciones no invalidan los resultados obtenidos, pero dificultan su aplicabilidad inmediata y a la vez, abren oportunidades para futuras mejoras de escalabilidad, robustez técnica y validación en entornos reales de operación.

8.3 Futuros trabajos

Este proyecto deja abiertas varias oportunidades para seguir mejorando y ampliando el sistema desarrollado en la empresa. Una de las principales ideas es utilizar la técnica de *fine-tuning*, para entrenar los modelos de lenguaje con información histórica de la empresa, lo que posibilitaría que el modelo entienda mejor los documentos reales que maneja la organización, aumentando la precisión y reduciendo errores.

Otra mejora importante por realizar es conectar el sistema directamente con plataformas como SAP Ariba. Esto automatizaría completamente el proceso, desde la recepción del archivo hasta su procesamiento final, sin intervención manual.

Por último, se recomienda realizar una evaluación del impacto operativo mediante una implementación parcial o piloto del sistema en un entorno productivo real. Esta instancia permitiría medir con mayor fidelidad los efectos concretos sobre tiempos de procesamiento, reducción de carga manual y mejora en la trazabilidad.

9 Referencias

ANDRITZ. (2024). Informe financiero.

ANDRITZ. (2025). https://www.andritz.com/pulp-and-paper-en/locations/chile-ltda/andritz-chile-ltda-spn

Anthropic. (2024). Prompt engineering best practices. https://docs.anthropic.com/en/docs/build-with-claude/prompt-engineering

Asimov, I. (1942). Runaround. *En Astounding Science-Fiction, 29(1), 94–103.*

Bender, E. M.-M. (2021). On the Dangers of Stochastic Parrots: Can Language Models Be Too Big? Association for Computing Machinery. doi:https://doi.org/10.1145/3442188.3445922

Cao, Z. Y. (2023). Reduciendo las brechas de dominio en las representaciones contextuales para la traducción automática neuronal basada en vecinos más cercanos. *Actas de la 61.ª Reunión Anual de la Asociación de Lingüística Computacional.*

Chiticariu L., K. R. (2010). Domain adaptation of rule-based annotators for named-entity recognition tasks. *Proceedings of the 2010 Conference on Empirical Methods in Natural Language Processing* (págs. 1002–1012). Association for Computational Linguistics.

Devlin, J. C. (2019). BERT: Pre-training of deep bidirectional transformers for language understanding. *Proceedings of NAACL-HLT*. https://arxiv.org/abs/1810.04805

Google DeepMind. (2024). Introducing Gemini 2.0: our new AI model for the agentic era. https://blog.google/technology/google-deepmind/google-gemini-ai-update-december-2024/

Haenlein, M. &. (2019). A brief history of artificial intelligence: On the past, present, and future of artificial intelligence. *California Management Review*, 5–14. doi:https://doi.org/10.1177/0008125619864925

Hladká, B. H. (2015). RExtractor: a Robust Information Extractor. *Proceedings of the 2015 Conference of the North American Chapter of the Association for Computational Linguistics: Demonstrations* (págs. 37–41). Association for Computational Linguistics.

IBM. (2024). ¿Qué es el procesamiento de lenguaje natural (PLN)?. https://www.ibm.com/mx-es/think/topics/natural-language-processing

Iqbal, F. N. (2023). Una breve introducción a la interfaz de programación de aplicaciones (API). *Zenodo*. doi:https://doi.org/10.5281/zenodo.10198423

John McCarthy, M. L. (1955). A Proposal for the Dartmouth Summer Research Project on Artificial Intelligence. Hanover, NH: Dartmouth College.

Jurafsky D, M. J. (2021). Speech and Language Processing (3rd ed). Stanford University.

Khurana, K. K. (2017). Natural Language Processing: State of The Art, Current Trends and Challenges. https://arxiv.org/pdf/1708.05148

Kpaz. (2024). https://kpaz.la/2024/06/12/beneficios-de-microsoft-power-automate/

Larson, A. M. (2023). Modelo de lenguaje grande (LLM). *Enciclopedia de Ciencia de Salem Press*.

Meta. (2023). LLaMa 3: https://www.llama.com/models/llama-3/

Microsoft. (2024). Programador de tareas. https://learn.microsoft.com/es-es/windows/win32/taskschd/task-scheduler-start-page

Mistral AI. (2023). Announcing Mistral 7B: a powerful, open-weight dense language model.: https://mistral.ai/news/announcing-mistral-7b

Onat Topal M, A. B. (2021). Exploring Transformers in Natural Language Generation: GPT, BERT, and XLNet. https://arxiv.org/abs/2102.08036

OpenAI. (2023). Informe técnico de GPT-4. https://doi.org/10.48550/arXiv.2303.08774

Petridis, C. (2024). Clasificación de textos: redes neuronales VS aprendizaje automático. https://arxiv.org/pdf/2412.21022

Taori, R. G. (2022). TaDaa: Transformer-based Automatic Ticket Assignment Advisor. https://arxiv.org/abs/2207.11187

Vallejo Echavarría, J. C. (2024). Más allá de los datos: la revolución de la inteligencia artificial en las ciencias de la información. *Revista Interamericana de Bibliotecología, 47*(1). doi:https://doi.org/10.17533/udea.rib.v47n1e355849

Vaswani A, S. N. (2017). Attention Is All You Need. https://arxiv.org/pdf/1706.03762

Anexo

A.1 Script Monitor.py

```
import time
import os
import shutil
import subprocess
from watchdog.observers import Observer
from watchdog.events import FileSystemEventHandler
import sys

# RUTAS
entrada_dir = r'C:\Users\xlllx\OneDrive\ENTRADA'

salida_dir = r'C:\Users\xlllx\OneDrive\SALIDA'

procesados_dir = os.path.join(entrada_dir, 'PROCESADOS')

script_arauco = r'C:\Users\xlllx\Desktop\Gemini_prueba\Arauco.py'

script_cmpc = r'C:\Users\xlllx\Desktop\Gemini_prueba\CMPC.py'

# FUNCIONES
def esperar_descarga_completa(ruta, timeout=60):
    """Espera hasta que el archivo deje de modificarse."""
    tiempo_inicio = time.time()
    ultima_mod = None
    while time.time() - tiempo_inicio < timeout:
        if os.path.exists(ruta):
            mod_actual = os.path.getmtime(ruta)
            if ultima_mod is None:
                ultima_mod = mod_actual
            elif mod_actual == ultima_mod:
                return True
            else:
                ultima_mod = mod_actual
        time.sleep(1)
    return False
```

```
class ArchivoHandler(FileSystemEventHandler):
    def on_created(self, event):
        if event.is_directory:
            return

        ruta_archivo = event.src_path
        nombre_archivo = os.path.basename(ruta_archivo)
        print(f"\n Detectado nuevo archivo: {nombre_archivo}")

        if not esperar_descarga_completa(ruta_archivo):
            print(" Tiempo de espera agotado")
            return

        try:
            if nombre_archivo.lower().endswith('.pdf'):
                print(" Procesando con Arauco.py...")
                comando=f'python"{script_arauco}""{ruta_archivo}""{salida_dir}"'
            elif nombre_archivo.lower().endswith('.doc') or
nombre_archivo.lower().endswith('.docx'):
                print(" Procesando con CMPC.py...")
                comando = f'python "{script_cmpc}" "{ruta_archivo}" "{salida_dir}"'
            else:
                print(" Archivo no compatible. Se omite.")
                return

            resultado = subprocess.run(comando, shell=True, capture_output=True,
text=True)
            print(" Salida del script:\n", resultado.stdout)
            if resultado.stderr:
                print(" Errores:\n", resultado.stderr)

            if resultado.returncode != 0:
                print(f" El script falló con código {resultado.returncode}.")
                return
```

```
            # MOVER CARPETA PROCESADOS
            if not os.path.exists(procesados_dir):
                os.makedirs(procesados_dir)

            destino = os.path.join(procesados_dir, nombre_archivo)
            shutil.move(ruta_archivo, destino)
            print(f" Archivo movido a: {destino}")

        except Exception as e:
            print(f" Error al procesar el archivo {nombre_archivo}: {e}")

# MONITOREO

if __name__ == "__main__":
    print(f" Usando Python desde: {sys.executable}")
    observer = Observer()
    observer.schedule(ArchivoHandler(), path=entrada_dir, recursive=False)
    observer.start()
    print(f" Monitoreando carpeta: {entrada_dir}...\n")

    try:
        while True:
            time.sleep(1)
    except KeyboardInterrupt:
        observer.stop()
    observer.join()
```

A.2 Script CMPC.py

```
import google.generativeai as genai
from docx import Document
import os
import sys
import pandas as pd
import warnings
from openpyxl import load_workbook
from openpyxl.styles import Font, Alignment, Border, Side
from openpyxl.utils import get_column_letter
import re
from difflib import get_close_matches
import xml.etree.ElementTree as ET


# CONFIGURACIÓN API
API_KEY = "-----------------------------------"
MODEL_NAME = "models/gemini-2.0-flash"
genai.configure(api_key=API_KEY)

# RUTAS
ruta_docx = sys.argv[1]
ruta_salida = sys.argv[2]
nombre_base = os.path.splitext(os.path.basename(ruta_docx))[0]
ruta_salida_final = os.path.join(ruta_salida, f"{nombre_base}_fusion.xlsx")

# FUNCIONES
def leer_docx(ruta):
    doc = Document(ruta)
    texto = []
    for p in doc.paragraphs:
        if p.text.strip():
            texto.append(p.text.strip())
    for tabla in doc.tables:
        for fila in tabla.rows:
            celdas = [celda.text.strip() for celda in fila.cells]
            if any(celdas):
                texto.append(" | ".join(celdas))
    return "\n".join(texto)
```

```
def extraer_con_gemini(texto):
    prompt = f"""
Extrae todos los elementos mencionados en el siguiente texto.
Para cada ítem, entrega un bloque con las siguientes etiquetas XML:
• <nombre>
• <número_parte>
• <código_material>
• <cantidad>
• <planta>
• <fabricante>
• <una_característica_adicional>
Agrupa cada bloque dentro de una etiqueta <elemento>.
Devuelve únicamente una lista de elementos en XML válido, sin explicaciones ni encabezados.
Texto:
"""{texto}"""
"""
    modelo = genai.GenerativeModel(MODEL_NAME)
    respuesta = modelo.generate_content(prompt, generation_config={"temperature": 0})
    return respuesta.text
def convertir_xml_a_diccionarios(xml_texto):
    try:
        elementos = []
        root = ET.fromstring(f"<root>{xml_texto}</root>")
        for elem in root.findall("elemento"):
            item = {
                "Nombre del elemento": elem.findtext("nombre", default="").strip(),
                "Número de parte": elem.findtext("número_parte", default="").strip(),
                "Código de material": elem.findtext("código_material", default="").strip(),
                "Cantidad": elem.findtext("cantidad", default="").strip(),
                "Planta": elem.findtext("planta", default="").strip(),
                "Fabricante": elem.findtext("fabricante", default="").strip(),
                "Características adicionales": elem.findtext("característica_adicional",
default="").strip(),
            }
            elementos.append(item)
        return elementos
    except Exception as e:
        print("Error al convertir XML:", e)
        return []
```

```
def completar_campos_faltantes(datos):
    campos = ["Nombre del elemento", "Número de parte", "Código de material", "Cantidad",
"Planta", "Fabricante", "Características adicionales"]
    for item in datos:
        for campo in campos:
            if campo not in item:
                item[campo] = ""
    return datos


def normalizar(texto):
    return re.sub(r"\W+", "", str(texto)).strip().lstrip("0").upper()


# EJECUCIÓN PRINCIPAL
if __name__ == "__main__":
    print(f" Procesando archivo: {ruta_docx}")
    texto = leer_docx(ruta_docx)

    if not texto.strip():
        print(" El documento está vacío.")
        sys.exit(1)

    resultado = extraer_con_gemini(texto)
    print(" Resultado de Gemini generado.")

    datos = convertir_xml_a_diccionarios(resultado)
    if not datos:
        print(" Error: No se pudo procesar la respuesta XML.")
        print(resultado)
        sys.exit(1)

    datos = completar_campos_faltantes(datos)
    df_info = pd.DataFrame(datos)
    df_info['Número de parte'] = df_info['Número de parte'].apply(normalizar)
    df_info['Código de material'] = df_info['Código de material'].apply(normalizar)

    columnas_extraer = ['Product Group', 'Sent to', 'Origin', 'Date']
    for col in columnas_extraer:
        df_info[col] = ""
```

```
    df_info['Coincidencia'] = ""
    df_info['Archivo Maestro'] = ""
    df_info['Fila Excel Maestro'] = ""
    df_info['Fuente'] = ""

    ruta_maestros = r"C:\Users\xlllx\Desktop\Gemini_prueba"
    maestros = [
        {"ruta": os.path.join(ruta_maestros, "MaestroAngela.xlsx"), "fuente": "Angela"},
        {"ruta": os.path.join(ruta_maestros, "MaestroIvania.xlsx"), "fuente": "Ivania"},
        {"ruta": os.path.join(ruta_maestros, "MaestroPablo.xlsx"), "fuente": "Pablo"}
    ]
    for idx_info, fila_info in df_info.iterrows():
        parte = fila_info['Número de parte']
        codigo = fila_info['Código de material']
        coincidencia_encontrada = False

        for maestro in maestros:
            ruta_maestro = maestro["ruta"]
            fuente = maestro["fuente"]
            nombre_archivo = os.path.basename(ruta_maestro)

            if not os.path.exists(ruta_maestro):
                continue
            df_maestro = pd.read_excel(ruta_maestro)

            with warnings.catch_warnings():
                warnings.simplefilter("ignore", category=FutureWarning)
                df_maestro = df_maestro.astype(str)

            col_map = {}
            for ref in columnas_extraer:
                coincidencias = get_close_matches(ref, df_maestro.columns, n=1, cutoff=0.6)
                if coincidencias:
                    col_map[ref] = coincidencias[0]

            for idx_maestro in reversed(df_maestro.index):
                fila_maestro = df_maestro.loc[idx_maestro]
                celdas = [normalizar(v) for v in fila_maestro.values if pd.notnull(v)]
```

```
            if normalizar(parte) in celdas or normalizar(codigo) in celdas:
                df_info.at[idx_info, 'Coincidencia'] = 'Sí'
                df_info.at[idx_info, 'Archivo Maestro'] = nombre_archivo
                df_info.at[idx_info, 'Fila Excel Maestro'] = idx_maestro + 2
                df_info.at[idx_info, 'Fuente'] = fuente

                for ref_col, real_col in col_map.items():
                    df_info.at[idx_info, ref_col] = fila_maestro[real_col]

                coincidencia_encontrada = True
                break
    if coincidencia_encontrada:
        break

    if not coincidencia_encontrada:
        df_info.at[idx_info, 'Coincidencia'] = 'No'

df_info.to_excel(ruta_salida_final, index=False)

wb = load_workbook(ruta_salida_final)
ws = wb.active
thin_border = Border(left=Side(style='thin'), right=Side(style='thin'),
                     top=Side(style='thin'), bottom=Side(style='thin'))
center_align = Alignment(horizontal="center", vertical="center")

for col in range(1, ws.max_column + 1):
    cell = ws.cell(row=1, column=col)
    cell.font = Font(bold=True)
    cell.alignment = center_align
    cell.border = thin_border

for row in ws.iter_rows(min_row=2, max_row=ws.max_row, max_col=ws.max_column):
    for cell in row:
        cell.alignment = center_align
        cell.border = thin_border
```

```
    for col in ws.columns:
        max_length = max(len(str(cell.value)) if cell.value else 0 for cell in col)
        col_letter = get_column_letter(col[0].column)
        ws.column_dimensions[col_letter].width = max_length + 2

    wb.save(ruta_salida_final)
    print(f" Excel fusionado guardado en: {ruta_salida_final}")
```

A.3 Script Arauco.py

```
import google.generativeai as genai
import pdfplumber
import os
import sys
import pandas as pd
import warnings
from openpyxl import load_workbook
from openpyxl.styles import Font, Alignment, Border, Side
from openpyxl.utils import get_column_letter
import re

# CONFIGURACIÓN GENERAL

API_KEY = "---------------------------------"
MODEL_NAME = "models/gemini-2.0-flash"
genai.configure(api_key=API_KEY)

# RUTAS
ruta_pdf = sys.argv[1]  # Ruta del PDF
ruta_salida = sys.argv[2]  # Carpeta de salida
nombre_base = os.path.splitext(os.path.basename(ruta_pdf))[0]
ruta_salida_final = os.path.join(ruta_salida, f"{nombre_base}_fusion.xlsx")
```

```
# FUNCIONES
def normalizar(texto):
    return re.sub(r'\W+', '', str(texto)).strip().lstrip("0").upper()

def extraer_texto_desde_pdf(ruta):
    texto_total = ""
    with pdfplumber.open(ruta) as pdf:
        for pagina in pdf.pages:
            texto = pagina.extract_text()
            if texto:
                texto_total += "\n" + texto
    return texto_total

def extraer_bloque_items(texto):
    texto_limpio = texto.lower().replace(" ", "").replace("\n", "").replace("\r", "")
    inicios     =     ["itemdescription",     "itemdescripción",     "itemdescripciónart",
"itemdescripciónart."]
    for inicio in inicios:
        if inicio in texto_limpio:
            return texto
    raise ValueError("No se encontró ningún delimitador inicial válido.")

def extraer_con_gemini(texto):
    prompt = f"""
Extrae todos los elementos mencionados en el siguiente texto.
Para cada ítem, entrega un bloque con las siguientes etiquetas XML:
• <nombre>
• <número_parte>
• <código_material>
• <cantidad>
• <planta>
• <fabricante>
• <característica_adicional>
Agrupa cada bloque dentro de una etiqueta <elemento>.
Devuelve únicamente una lista de elementos en XML válido, sin explicaciones ni encabezados.
Texto:
"""{texto}"""
"""
```

```
    modelo = genai.GenerativeModel(MODEL_NAME)
    respuesta = modelo.generate_content(prompt, generation_config={"temperature": 0})
    return respuesta.text

def xml_a_dict_lista(xml_str):
    import xml.etree.ElementTree as ET
    try:
        root = ET.fromstring(f"<root>{xml_str}</root>")
        datos = []
        for elemento in root.findall("elemento"):
            datos.append({
                "Nombre del elemento": elemento.findtext("nombre", default=""),
                "Número de parte": elemento.findtext("número_parte", default=""),
                "Código de material": elemento.findtext("código_material", default=""),
                "Cantidad": elemento.findtext("cantidad", default=""),
                "Planta": elemento.findtext("planta", default=""),
                "Fabricante": elemento.findtext("fabricante", default=""),
                "Características                                                adicionales":
elemento.findtext("característica_adicional", default="")
            })
        return datos
    except Exception as e:
        print(f" Error al parsear XML: {e}")
        return []

def formatear_excel(ruta):
    wb = load_workbook(ruta)
    ws = wb.active
    thin_border = Border(left=Side(style='thin'), right=Side(style='thin'),
                         top=Side(style='thin'), bottom=Side(style='thin'))
    center_align = Alignment(horizontal="center", vertical="center")

    for col in range(1, ws.max_column + 1):
        cell = ws.cell(row=1, column=col)
        cell.font = Font(bold=True)
        cell.alignment = center_align
        cell.border = thin_border
```

```
    for row in ws.iter_rows(min_row=2, max_row=ws.max_row, max_col=ws.max_column):
        for cell in row:
            cell.alignment = center_align
            cell.border = thin_border

    for col in ws.columns:
        max_length = max(len(str(cell.value)) if cell.value else 0 for cell in col)
        col_letter = get_column_letter(col[0].column)
        ws.column_dimensions[col_letter].width = max_length + 2
    wb.save(ruta)

# EJECUCIÓN PRINCIPAL
if __name__ == "__main__":
    print(f"Procesando archivo PDF: {ruta_pdf}")
    texto = extraer_texto_desde_pdf(ruta_pdf)
    bloque = extraer_bloque_items(texto)

    resultado = extraer_con_gemini(bloque)
    print("Resultado de Gemini generado.")

    datos = xml_a_dict_lista(resultado)
    if not datos:
        print("No se pudo extraer información válida del XML.")
        sys.exit(1)
    df_info = pd.DataFrame(datos)
    df_info['Número de parte'] = df_info['Número de parte'].astype(str).str.strip().str.upper()
    df_info['Código de material'] = df_info['Código de material'].astype(str).str.strip().str.lstrip("0").str.upper()

    columnas_extraer = ['Product Group', 'Sent to', 'Origin', 'Date']
    for col in columnas_extraer:
        df_info[col] = ""

    df_info['Coincidencia'] = ""
    df_info['Archivo Maestro'] = ""
    df_info['Fila Excel Maestro'] = ""
    df_info['Fuente'] = ""
```

```
ruta_maestros = r"C:\Users\xlllx\Desktop\Gemini_prueba"
maestros = [
    {"ruta": os.path.join(ruta_maestros, "MaestroAngela.xlsx"), "fuente": "Angela"},
    {"ruta": os.path.join(ruta_maestros, "MaestroIvania.xlsx"), "fuente": "Ivania"},
    {"ruta": os.path.join(ruta_maestros, "MaestroPablo.xlsx"), "fuente": "Pablo"}
]

for idx_info, fila_info in df_info.iterrows():
    parte = fila_info['Número de parte']
    codigo = fila_info['Código de material']
    coincidencia_encontrada = False

    for maestro in maestros:
        ruta_maestro = maestro["ruta"]
        fuente = maestro["fuente"]
        nombre_archivo = os.path.basename(ruta_maestro)

        if not os.path.exists(ruta_maestro):
            continue

        df_maestro = pd.read_excel(ruta_maestro)
        with warnings.catch_warnings():
            warnings.simplefilter("ignore", category=FutureWarning)
            df_maestro = df_maestro.astype(str)

        columnas_validas = [col for col in columnas_extraer if col in df_maestro.columns]

        for idx_maestro in reversed(df_maestro.index):
            fila_maestro = df_maestro.loc[idx_maestro]
            celdas = [normalizar(v) for v in fila_maestro.values if pd.notnull(v)]

            if normalizar(parte) in celdas or normalizar(codigo) in celdas:
                df_info.at[idx_info, 'Coincidencia'] = 'Sí'
                df_info.at[idx_info, 'Archivo Maestro'] = nombre_archivo
                df_info.at[idx_info, 'Fila Excel Maestro'] = idx_maestro + 2
                df_info.at[idx_info, 'Fuente'] = fuente
```

```
                    for col in columnas_validas:
                        df_info.at[idx_info, col] = fila_maestro[col]

                    coincidencia_encontrada = True
                    break
        if coincidencia_encontrada:
            break
        if not coincidencia_encontrada:
            df_info.at[idx_info, 'Coincidencia'] = 'No'

    df_info.to_excel(ruta_salida_final, index=False)
    formatear_excel(ruta_salida_final)
    print(f"Excel fusionado guardado en: {ruta_salida_final}")
```