

UNIVERSIDAD SAN FRANCISCO DE QUITO USFQ

Colegio de Ciencias e Ingenierías

**Automatización Inteligente en la Renta de Departamentos
Vacacionales: Desarrollo e Implementación de un Modelo de
Chatbot inteligente re-entrenado para la Gestión de Consultas**

Mateo Nicolás Benalcázar Medina
Ingeniería Industrial

Trabajo de fin de carrera presentado como requisito
para la obtención del título de
INGENIERO INDUSTRIAL

Quito, 8 de mayo de 2025

UNIVERSIDAD SAN FRANCISCO DE QUITO USFQ

Colegio de Ciencias e Ingenierías

HOJA DE CALIFICACIÓN DE TRABAJO DE FIN DE CARRERA

**Automatización Inteligente en la Renta de Departamentos
Vacacionales: Desarrollo e Implementación de un Modelo de Chatbot
inteligente re-entrenado para la Gestión de Consultas**

Mateo Nicolás Benalcázar Medina

Kenya Velasco Tapia, MS.

Quito, 8 de mayo de 2025

© DERECHOS DE AUTOR

Por medio del presente documento certifico que he leído todas las Políticas y Manuales de la Universidad San Francisco de Quito USFQ, incluyendo la Política de Propiedad Intelectual USFQ, y estoy de acuerdo con su contenido, por lo que los derechos de propiedad intelectual del presente trabajo quedan sujetos a lo dispuesto en esas Políticas.

Asimismo, autorizo a la USFQ para que realice la digitalización y publicación de este trabajo en el repositorio virtual, de conformidad a lo dispuesto en la Ley Orgánica de Educación Superior del Ecuador.

Nombres y apellidos: Mateo Nicolás Benalcázar Medina

Código: 00212605

Cédula de identidad: 1719955971

Lugar y fecha: Quito, 8 de mayo de 2025

ACLARACIÓN PARA PUBLICACIÓN

Nota: El presente trabajo, en su totalidad o cualquiera de sus partes, no debe ser considerado como una publicación, incluso a pesar de estar disponible sin restricciones a través de un repositorio institucional. Esta declaración se alinea con las prácticas y recomendaciones presentadas por el Committee on Publication Ethics COPE descritas por Barbour et al. (2017) Discussion document on best practice for issues around theses publishing, disponible en <http://bit.ly/COPETheses>.

UNPUBLISHED DOCUMENT

Note: The following capstone project is available through Universidad San Francisco de Quito USFQ institutional repository. Nonetheless, this project – in whole or in part – should not be considered a publication. This statement follows the recommendations presented by the Committee on Publication Ethics COPE described by Barbour et al. (2017) Discussion document on best practice for issues around theses publishing available on <http://bit.ly/COPETheses>.

RESUMEN

Este proyecto desarrolla un sistema de atención al cliente basado en inteligencia artificial, aplicado a un negocio real de renta de departamentos vacacionales. El objetivo principal fue automatizar la atención a consultas comunes de usuarios, optimizando tiempos de respuesta y reduciendo la carga operativa del administrador. Se utilizó un enfoque basado en RAG (Retrieval Augmented Generation) como base, sobre el cual se aplicaron técnicas de refinamiento de modelos de lenguaje como Few-shot learning, Fine-tuning y una combinación de ambos métodos. Los modelos fueron evaluados utilizando un set de 120 preguntas diseñadas a partir de escenarios reales, aplicando escalas de calificación tipo Likert y pruebas estadísticas como Friedman y Nemenyi para comparar resultados. El modelo combinado (Fine-tuned + Few-shot) demostró ser el de mejor desempeño, logrando un equilibrio entre precisión técnica y naturalidad conversacional. Si bien los resultados fueron positivos, el proyecto presenta limitaciones relacionadas con el entorno de prueba, el tamaño de muestra y la futura escalabilidad. Este estudio demuestra que, con el refinamiento adecuado, un chatbot basado en LLMs puede integrarse exitosamente a operaciones reales, contribuyendo a mejorar tanto la eficiencia interna como la experiencia de atención al cliente.

Palabras clave: chatbot, inteligencia artificial, RAG, fine-tuning, few-shot learning, automatización de atención.

ABSTRACT

This project develops an AI-based customer service system applied to a real vacation rental business. The main objective was to automate the handling of common user inquiries, optimizing response times and reducing the administrator's operational workload. The approach was based on Retrieval Augmented Generation (RAG), enhanced through language model refinement techniques such as Few-shot learning, Fine-tuning, and a combination of both methods. The models were evaluated using a set of 120 questions designed from real-life scenarios, applying Likert-type rating scales and statistical tests such as Friedman and Nemenyi for performance comparison. The combined model (Fine-tuned + Few-shot) showed the best overall performance, achieving a balance between technical accuracy and conversational naturalness. Although the results were positive, the project presents limitations related to the testing environment, sample size, and future scalability. This study demonstrates that, with proper refinement, an LLM-based chatbot can be successfully integrated into real business operations, contributing to improvements in both internal efficiency and customer experience.

Keywords: chatbot, artificial intelligence, RAG, fine-tuning, few-shot learning, customer service automation.

TABLA DE CONTENIDO

1. Introducción.....	11
1.1. Pregunta de investigación	12
1.2. Objetivo general.....	12
1.3. Objetivos específicos	12
1.4. Limitaciones de investigación.....	13
2. Revisión Literaria	13
2.1. Introducción	13
2.2. Chatbots y su Impacto en la Renta de Departamentos Vacacionales	13
2.3. Optimización mediante RAG, Fine-Tuning y Prompt Engineering	14
2.4. Embeddings en la recuperación de información	15
2.5. Beneficios y Desafíos en la Implementación de Chatbots.....	15
3. Metodología.....	16
3.1. Entendimiento del negocio	17
3.2. Entendimiento de los datos	17
3.3. Ejecución – Sprints	17
3.4. Despliegue.....	18
4. Desarrollo	18
4.1. Entendimiento del negocio	18
4.1.1. Diagrama de flujo del proceso	19
4.2. Entendimiento de los datos	20
4.2.1. Tiempo promedio de respuesta	20
4.2.2. Tiempo promedio en WhatsApp.....	20
4.2.3. Tasa de conversión.....	21
4.2.4. Consultas principales	21
4.2.5. Reglas del negocio	22
4.3. Sprints	22
4.3.1. Construcción de base de datos	22
4.3.2. Selección de tecnología de LLM (Inteligencia Artificial)	22
4.3.3. Investigación de estrategias RAG	23
5. Implementación y prueba de métodos de refinamiento.....	25
5.1. Modelo Base (RAG)	25
5.2. Modelo Few-shot	26
5.3. Modelo Fine-tuned.....	27

5.4.	Modelo combinado (Few-shot + Fine-tuned)	29
6.	Despliegue y pruebas	30
6.1.	Entorno de pruebas	30
6.2.	Método de evaluación	31
6.3.	Procedimiento	33
7.	Resultados.....	34
7.1.	Resultados Generales de evaluación.....	34
7.2.	Comparación entre modelos por categorías de pregunta	35
7.3.	Modelo de mejor desempeño	37
7.4.	Limitaciones.....	38
8.	Conclusiones.....	39
9.	Recomendaciones	41
10.	Referencias	43
11.	Anexos	45

ÍNDICE DE TABLAS

Tabla 1: Criterios para calificaciones del set de evaluación.	32
Tabla 2: Calificación promedio de modelos.	34
Tabla 3: Comparación pareada en evaluación general.	35
Tabla 4: Resultados de prueba de Friedman por categoría de preguntas.....	35
Tabla 5: Comparación pareada en evaluación de categoría consultas de SQL.....	36
Tabla 6: Comparación pareada en evaluación de categoría de consultas Ambiguas.....	36

ÍNDICE DE ANEXOS

Anexo 1: Tamaño de muestra tiempo promedio de respuesta.	45
Anexo 2: Diagramas de caja de tiempo promedio de respuesta con valores extremos y sin valores extremos.	45
Anexo 3: Tamaño de muestra de tiempo de operador en WhatsApp.	45
Anexo 4: Tamaño de muestra de tasa de conversión.	45
Anexo 5: Tamaño de muestra de proporción de mensajes por categoría.	45
Anexo 6: Pareto de requerimientos más importantes de respuesta de chatbot.	46
Anexo 7: Flujo de modelo Base.	46
Anexo 8: Código de modelo Base.	47
Anexo 9: Ejemplos de funcionamiento y respuesta de modelo Base.	48
Anexo 10: Flujo de modelo Few-shot.	49
Anexo 11: Fracción de código de modelo Few-shot (Ejemplos en prompt).	49
Anexo 12: Ejemplos de funcionamiento y respuestas de modelo Few-shot.	50
Anexo 13: Flujo de modelo Fine-tuned.	51
Anexo 14: Fracción de código de modelo Fine-tuned.	52
Anexo 15: Ejemplos de funcionamiento y respuestas de modelo Fine-tuned.	54
Anexo 16: Flujo de modelo combinado. (Mismo que el Fine-tuned).	55
Anexo 17: Fracción de código de modelo combinado.	55
Anexo 18: Ejemplos de funcionamiento y respuestas de modelo combinado.	57
Anexo 19: Ejemplos de preguntas de cada categoría.	58
Anexo 20: Archivo de texto guardado para preguntas frecuentes.	59
Anexo 21: ERD de base de datos de SQL.	59
Anexo 22: Diagrama de resultados prueba de Friedman en evaluación general.	60

1. Introducción

En la actualidad, la inteligencia artificial (IA) y la automatización desempeñan un papel fundamental en la mayoría de las industrias. La capacidad de las empresas para adaptarse e implementar estas tecnologías se ha convertido en un factor clave para optimizar recursos como costos y tiempo, además de permitir la automatización de tareas que antes parecían imposibles. Como lo señala Gordon Graell (2023), las tecnologías basadas en IA “potencian e incrementan el desarrollo tecnológico allí en donde se aplican” y transforman el entorno productivo empresarial (p. 3). Una de estas herramientas es el chatbot, cuyo uso en el servicio al cliente digital ha evolucionado significativamente. Según Sofiyah et al. (2024), el uso de chatbots mejora la calidad del servicio gracias a su capacidad de dar respuestas coherentes y en tiempo real, lo que contribuye a la satisfacción y lealtad del cliente (p. 46). Inicialmente, los chatbots estaban limitados a responder preguntas predefinidas; sin embargo, los avances en modelos de lenguaje grande (LLM, por sus siglas en inglés) han permitido desarrollar chatbots con inteligencia artificial capaces de mantener conversaciones naturales con los usuarios, mejorando así la experiencia y la calidad del servicio. Tal como explican Trad y Chehab (2024), los LLM permiten generar respuestas más contextuales y precisas gracias al uso de técnicas como el fine-tuning o el prompt engineering, lo cual representa una evolución frente a los sistemas basados en reglas (p. 368).

Este proyecto propone la implementación de un chatbot basado en LLM en un negocio de renta de departamentos vacacionales administrado por una sola persona. La necesidad surge del alto volumen de consultas recibidas diariamente a través de su principal canal de ventas: WhatsApp. Actualmente, la gestión manual de estas interacciones genera tiempos de respuesta inconsistentes, afectando la eficiencia del servicio. Además, al tratarse de un negocio secundario, el administrador debe destinar una cantidad considerable de tiempo para responder preguntas repetitivas.

Al analizar el proceso, se observa que la mayoría de las consultas siguen un patrón predecible, lo que sugiere que la automatización mediante un chatbot entrenado podría mejorar la gestión de reservas y la atención al cliente. Por ello, este proyecto busca desarrollar un chatbot que implemente Retrieval-Augmented Generation (RAG, generación aumentada por recuperación) y otras técnicas de refinamiento de modelos para responder automáticamente consultas y gestionar reservas. La hipótesis de trabajo plantea que esta solución no solo optimizará la eficiencia operativa, sino que también incrementará la tasa de conversión de clientes.

1.1. Pregunta de investigación

¿Cómo puede un chatbot basado en inteligencia artificial mejorar la eficiencia y conversión de clientes en la renta de departamentos vacacionales mediante la automatización de la atención al cliente?

1.2. Objetivo general

Desarrollar un chatbot con inteligencia artificial que automatice la atención al cliente en la renta de departamentos vacacionales, proporcionando respuestas precisas y en tiempo real.

1.3. Objetivos específicos

1. Integrar el chatbot con una base de datos y aplicar RAG para mejorar la calidad de las respuestas.
2. Identificar la técnica de refinamiento más efectiva para optimizar la precisión y relevancia de las respuestas del chatbot.
3. Reducir el tiempo de respuesta a consultas y aumentar la tasa de conversión de clientes.
4. Minimizar el tiempo que el operario dedica a la atención de consultas repetitivas.

5. Diseñar un chatbot capaz de simular una conversación natural para mejorar la experiencia del usuario.

1.4. Limitaciones de investigación

Este proyecto se enfoca exclusivamente en la gestión de reservas y la automatización de respuestas a preguntas frecuentes. No abordará aspectos relacionados con la gestión financiera, pagos o facturación, los cuales seguirán realizándose a través de la plataforma externa utilizada por el negocio. También cabe recalcar que la efectividad del chatbot dependerá de la calidad y cantidad de datos disponibles para su entrenamiento, lo que puede influir en la precisión de sus respuestas.

2. Revisión Literaria

2.1. Introducción

El uso de chatbots en la automatización de la atención al cliente ha demostrado ser una estrategia eficiente en diversas industrias, incluyendo el sector turístico y de servicios. Según Sofiyah et al. (2024), “la implementación de un chatbot de inteligencia artificial basado en WhatsApp [...] tiene el potencial de aumentar la eficiencia y la calidad del servicio” (p. 45). Además, Valderrama y Aguilar-Alonso (2023) destacan que los chatbots basados en procesamiento de lenguaje natural permiten ofrecer respuestas rápidas y precisas, lo cual incrementa la satisfacción del cliente. En este contexto, la aplicación de modelos de lenguaje basados en inteligencia artificial (IA) no solo ha optimizado los tiempos de respuesta, sino que también ha mejorado la experiencia del usuario y ha contribuido a aumentar la conversión de clientes potenciales en reservas efectivas (Pionce Arteaga et al., 2022, p. 109).

2.2. Chatbots y su Impacto en la Renta de Departamentos Vacacionales

Los chatbots han evolucionado significativamente y han demostrado su impacto en la eficiencia operativa de empresas que requieren de atención al cliente continua. Gordon Graell

y Roberto Daniel (2023) destacan la capacidad de los chatbots para transformar la gestión de la información en los negocios, lo que facilita la automatización de tareas repetitivas y la personalización de la experiencia del cliente.

En el contexto de la renta de departamentos vacacionales, los chatbots permiten gestionar consultas sobre disponibilidad, precios y condiciones de alquiler, así como agilizar procesos de reserva y atención al cliente. De acuerdo con Zhang et al. (2024), los chatbots en el sector turístico han demostrado mejorar la satisfacción del cliente al proporcionar interacciones más ágiles y personalizadas, mitigando posibles violaciones de expectativas durante el servicio. Esta capacidad resulta especialmente relevante en negocios donde la rapidez y la calidad de respuesta inciden directamente en la conversión de reservas.

2.3. Optimización mediante RAG, Fine-Tuning y Prompt Engineering

La optimización de chatbots puede realizarse mediante enfoques como Retrieval-Augmented Generation (RAG), fine-tuning y prompt engineering. RAG combina la capacidad generativa de modelos de lenguaje con la recuperación de información en bases de datos externas, lo que permite generar respuestas más precisas y contextualizadas. Según Nankya y Oppenheimer (2025), el uso de RAG mejora la precisión en preguntas de dominio específico al incorporar información actualizada en tiempo real.

Por otro lado, Pornprasita y Tantithamthavorn (2024) analizaron la eficacia del fine-tuning y el prompt engineering en la automatización de revisiones de código, concluyendo que el fine-tuning permite un mejor desempeño en tareas específicas, aunque a mayor costo computacional. En el contexto de chatbots para renta vacacional, un enfoque híbrido podría optimizar la precisión en respuestas sin incurrir en altos costos de entrenamiento.

2.4. Embeddings en la recuperación de información

Los embeddings son representaciones numéricas de conceptos, codificadas como vectores de alta dimensión que capturan relaciones semánticas entre textos. Esta representación permite que frases o documentos puedan ser comparados no solo por coincidencias literales, sino por su significado contextual. Según OpenAI (2024), los embeddings convierten el contenido en vectores numéricos que pueden ser evaluados entre sí para medir su similitud, facilitando aplicaciones como búsqueda semántica, agrupamiento de textos similares y detección de relaciones conceptuales. En el contexto de los sistemas basados en Retrieval-Augmented Generation (RAG), los embeddings son fundamentales para encontrar fragmentos de información relevantes en bases de datos, incluso cuando la consulta del usuario no coincide exactamente con el lenguaje almacenado. Esto mejora significativamente la calidad y precisión de las respuestas generadas por chatbots que integran modelos de lenguaje con recuperación de información.

2.5. Beneficios y Desafíos en la Implementación de Chatbots

El uso de chatbots en la atención al cliente ha demostrado mejorar la relación con los clientes y la eficiencia en los tiempos de respuesta (Sofiyah et al., 2024). Sin embargo, la implementación de chatbots también presenta desafíos como la seguridad y privacidad de los datos, aspectos clave en la gestión de reservas vacacionales (Segovia-García, 2024). Además, la necesidad de garantizar respuestas coherentes y alineadas con la información actualizada hace que la integración de RAG sea una solución viable para mejorar la confiabilidad del chatbot.

En conclusión, la integración de chatbots con modelos de lenguaje avanzado representa una oportunidad para optimizar la atención al cliente en la renta de departamentos vacacionales.

Sin embargo, la selección de estrategias de optimización, como RAG, fine-tuning o prompt engineering, es clave para lograr una solución eficiente y escalable.

3. Metodología

Para el desarrollo de este proyecto, se utilizó una combinación de dos metodologías: CRISP-DM y Scrumban, cada una elegida por su capacidad para abordar distintos aspectos del proceso.

CRISP-DM (Cross Industry Standard Process for Data Mining) es una metodología ampliamente utilizada en proyectos que requieren minería de datos. Su estructura iterativa permite un enfoque sistemático en la preparación y modelado de datos, asegurando que cada fase del desarrollo esté respaldada por análisis rigurosos y mejoras continuas (Durango Vanegas et al., 2023). Esta metodología fue clave en la optimización del procesamiento de datos y en la generación de modelos predictivos que mejoran la precisión de las respuestas del chatbot.

Por otro lado, Scrumban combina elementos de Scrum y Kanban, proporcionando una estructura ágil sin la necesidad de definir roles estrictos de desarrollo. Scrum aporta claridad en la planificación y gestión iterativa, mientras que Kanban facilita un flujo de trabajo visualmente organizado y adaptable a cambios en tiempo real (Alqudah & Razali, 2018). Esto permitió mantener la flexibilidad necesaria en el desarrollo del chatbot, optimizando tiempos de implementación y adaptabilidad a nuevas necesidades del proyecto.

La combinación de estas metodologías permitió un desarrollo eficiente y estructurado del chatbot, garantizando que cada iteración estuviera respaldada por datos relevantes y que el proceso de implementación fuera ágil y adaptable a los requerimientos específicos del sector de renta de departamentos vacacionales.

3.1. Entendimiento del negocio

El primer paso en el desarrollo del proyecto consiste en comprender a fondo el negocio, identificando los problemas existentes y analizando sus causas raíz. Para ello, es fundamental establecer una comunicación efectiva con el responsable del negocio para quien se desarrolla la solución, con el objetivo de comprender sus necesidades y los procesos que llevan a cabo.

Este análisis detallado permite garantizar que la solución propuesta realmente aporte mejoras significativas y se alinee con los requerimientos específicos del negocio. En esta fase, se documentaron los procesos clave, se identificaron los principales desafíos y se justificó la necesidad de una solución que optimice la operación y resuelva eficientemente los problemas detectados.

3.2. Entendimiento de los datos

Este paso complementa el entendimiento del negocio, pero con un enfoque exclusivo en los datos. Su objetivo es analizar profundamente los tipos de datos que se manejan para garantizar que, al automatizarlos, sigan proporcionando el mismo valor o incluso mejoren su funcionalidad dentro del negocio.

Durante esta fase, se realiza una validación estadística para cuantificar los problemas previamente identificados, permitiendo así una justificación basada en datos para la solución propuesta. Estas métricas también serán útiles para después comparar el rendimiento actual del negocio con uno futuro cuando se haya implementado el chatbot. Además, se analizan los mensajes y conversaciones que deben ser automatizados, priorizando aquellos mensajes que tienen mayor impacto operativo y en la experiencia del cliente.

3.3. Ejecución – Sprints

Los sprints constituyen la fase central de la ejecución del proyecto, permitiendo organizar y priorizar las tareas clave para su desarrollo. En esta etapa, se desglosan y estructuran todas las

necesidades del proyecto en ciclos de trabajo cortos, asegurando una ejecución eficiente y alineada con los requerimientos establecidos.

El enfoque iterativo de los sprints permite evaluar continuamente el progreso, realizar ajustes y garantizar que cada entrega cumpla con los objetivos definidos. Al final de cada sprint, se lleva a cabo una evaluación de los resultados obtenidos, especialmente en lo que respecta a los datos y modelos utilizados. Esto incluye la validación de modelos fine-tuned y la evaluación de los 4 modelos desarrollados.

3.4. Despliegue

En esta fase, el chatbot desarrollado fue probado en un entorno controlado que simulaba su uso real a través de WhatsApp. Aunque no se implementó directamente en la plataforma, se recrearon las condiciones de interacción con usuarios para evaluar su funcionamiento de manera precisa. El despliegue implicó no solo la validación operativa del chatbot, sino también una evaluación detallada de su rendimiento en este entorno simulado. Se analizaron aspectos clave como la precisión de las respuestas utilizando una escala de Likert para evaluar la calidad y relevancia de las conversaciones del chatbot.

Por último, se identifican posibles áreas de mejora con base en los datos recopilados, estableciendo un punto de partida para futuras optimizaciones y refinamientos del chatbot.

4. Desarrollo

4.1. Entendimiento del negocio

En esta sección se investigó y discutió en detalle las dificultades asociadas a la gestión de rentas en la playa. Un tema muy importante, es que hay un gran volumen de consultas, sobre todo en temporada alta de verano o cerca de feriados. Estas consultas son atendidas por una sola persona, lo que se traduce en mucho tiempo en pantalla respondiendo preguntas.

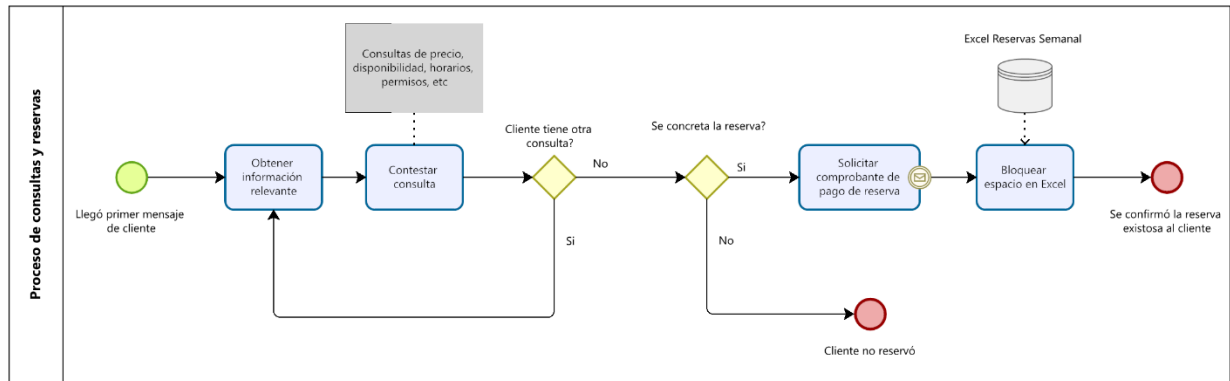
Además, es importante considerar que la mayoría de las consultas se quedan como simples preguntas y no se concretan en reservas. Esto demuestra como se implementa una gran cantidad de tiempo en una actividad, que, si bien es necesaria, en su mayor parte no representa ingresos ni suma valor, lo cual se puede considerar como un desperdicio.

También se observó que el tiempo de respuesta no es consistente. Si bien en una gran cantidad de ocasiones las respuestas eran rápidas, en menos de 5 minutos, también había varias ocasiones donde las respuestas duraban incluso varias horas, o donde no se contestó alguna consulta. Los motivos de esto son que justamente, el administrador de responder no se encarga únicamente de ese trabajo, y además no contesta a todas las horas.

4.1.1. Diagrama de flujo del proceso

El siguiente diagrama representa el proceso de consultas y reservas. Este proceso se da específicamente cuando una persona está interesada la renta de un departamento, por lo que se comunica de forma directa con el administrador. El operador envía información general del condominio, y posteriormente se produce una fase de intercambio constante de mensajes, dado que la persona interesada suele tener varias preguntas antes de decidirse por rentar o no rentar un departamento. El chatbot estará incluido en esta primera parte del proceso, ya que es donde hay mayor volumen de consultas y donde se dan constantemente los reflujos del proceso.

Figura 1: Diagrama de proceso de consultas y reservas.



4.2. Entendimiento de los datos

En esta sección se realizaron mediciones para entender la situación actual del negocio con métricas reales. También se investigó a detalle la información requerida para levantar los datos fundamentales para el desarrollo del chatbot.

4.2.1. Tiempo promedio de respuesta

Para calcular el tiempo promedio de respuesta se utilizó un tamaño de muestra de 51 datos (**Anexo 1**). El tiempo promedio de respuesta es de 53 minutos con una desviación estándar de 124 minutos. Dado que esta medición no refleja la realidad correctamente, se decidió hacer un análisis de diagrama de caja (**Anexo 2**). El análisis muestra que la mediana se ubica alrededor de los 5 minutos, sin embargo, hay muchos valores elevados hasta alrededor de los 40 minutos. Y, por último, existen con frecuencia valores extremos, que pueden demorar desde 100 hasta 500 minutos. Estos datos se dan principalmente cuando se envían mensajes en horas tardías y no se responden hasta el día siguiente. Esta métrica actual demuestra la necesidad de automatizar el sistema de consultas para poder ofrecer al cliente un tiempo consistente y corto de respuesta, lo cual no se da en la actualidad.

4.2.2. Tiempo promedio en WhatsApp

Para calcular el tiempo promedio diario de operador en WhatsApp se utilizó un tamaño de muestra de 16 datos (**Anexo 3**). En este se obtuvo un promedio de 238 minutos y una

desviación de 95 minutos. Este resultado es considerablemente alto dado que corresponde al tiempo diario de atención y refuerza la importancia de implementar un sistema automático de respuestas que brinde al operador mayor tiempo para realizar actividades de mayor valor.

4.2.3. Tasa de conversión

Para calcular la tasa de conversión se calculó la proporción de reservas en un mes con respecto al total de personas que hicieron una consulta. Se utilizó un tamaño de muestra de 203 conversaciones (**Anexo 4**). Se obtuvo que actualmente, se tiene una tasa de conversión de 19.03%. Esto puede interpretarse de varias formas que justifican la implementación del proyecto. En primer lugar, un 88,97% de las consultas no derivaron en reservas, representando una inversión considerable de tiempo sin retorno económico. Y segundo, los tiempos de respuesta elevados pueden afectar la tasa de conversión dado que un posible cliente encuentre otras opciones. Por lo tanto, se espera que, al disminuir el tiempo de respuesta, aumente la tasa de conversión al brindar al cliente un servicio rápido. Y también, se descarga al operador de una actividad que en mayor parte representa no conversiones.

4.2.4. Consultas principales

Como última métrica y para incluir en el Backlog del producto, se incluyó un estudio de las principales consultas que se reciben a WhatsApp en la parte del proceso que se quiere automatizar para priorizar los requerimientos que el chatbot debería manejar. Se utilizó un tamaño de muestra de 93 mensajes (**Anexo 5**) y se categorizó cada mensaje por la información que solicitaba. Con esto se realizó un diagrama de Pareto para poner la mayor cantidad del esfuerzo en el desarrollo de respuestas para las consultas que tengan mayor frecuencia. Como resultado, se obtuvo que las consultas sobre disponibilidad, precios y solicitudes de información adicional representan aproximadamente el 80% del total. (**Anexo 6**).

4.2.5. Reglas del negocio

Como última actividad, se evaluó como se manejan los precios y la disponibilidad de las reservas. Este es un paso importante para poder estructurar las tablas relacionales de la base de datos de forma correcta y que reflejen la realidad del negocio. Para esta parte se realizaron varias reuniones y el encargado del chat de WhatsApp explicó las partes esenciales que debería tener la base de datos para que funcione de manera correcta. Una vez finalizado esto, se procedió a realizar los sprints, donde justamente el primero, era acerca de la construcción de la base de datos.

4.3. Sprints

4.3.1. Construcción de base de datos

Para la construcción de la base de datos se utilizó SQL, implementando un esquema de tablas relacionales. Se diseñaron estructuras optimizadas para almacenar y gestionar eficientemente la información relevante para el chatbot, incluyendo información de cada departamento, información de disponibilidad por fechas, e información de los precios. Todas las tablas se definieron con claves primarias o foráneas para garantizar la integridad referencial y facilitar la recuperación de datos de manera eficiente.

4.3.2. Selección de tecnología de LLM (Inteligencia Artificial)

Para el desarrollo del chatbot, se optó por utilizar la API de OpenAI: GPT-4o-mini debido a su facilidad de integración, menor tiempo de implementación. Al estar alojado en la nube, la API de OpenAI permite una conexión directa con WhatsApp sin necesidad de configurar infraestructura adicional. Además, su modelo de precios es accesible, eliminando la necesidad de invertir en servidores o hardware especializado. Finalmente, la abundancia de documentación y tutoriales facilita la implementación y resolución de problemas, asegurando

una integración más ágil y eficiente. Si bien utilizar alternativas como Hugging Face Transformers no tiene costo por uso, puede tener costos ocultos como de infraestructura para servidores y para integración con la nube, además que carece de la cantidad de información que tiene OpenAI y su muy popular modelo.

4.3.3. Investigación de estrategias RAG

Para la implementación inicial del modelo (LLM) y la base de datos, se investigaron técnicas avanzadas como el RAG.

RAG es un enfoque que combina la capacidad generativa de los modelos de lenguaje con la recuperación de información relevante desde bases de datos externas, mejorando la precisión y actualidad de las respuestas generadas. Su funcionamiento se basa en dos fases principales. En primer lugar, el modelo recupera información relevante de una fuente externa, como una base de datos o documentos almacenados, utilizando técnicas de búsqueda semántica basadas en embeddings. Posteriormente, el modelo de lenguaje utiliza la información recuperada para generar una respuesta contextualizada y precisa.

Este método es particularmente útil para superar las limitaciones de los modelos de lenguaje que dependen exclusivamente de datos preentrenados, los cuales pueden estar desactualizados o carecer de información específica del dominio. Según Nankya y Oppenheimer (2025), la combinación de RAG con modelos de lenguaje preentrenados permite mejorar significativamente la precisión en preguntas de dominio específico, ya que el modelo puede acceder a datos externos en tiempo real en lugar de depender únicamente de su conocimiento interno. Además, el RAG proporciona una solución al no tener que entrenar un nuevo modelo cada vez que se actualicen datos, ya que el modelo solo accede a información, pero no la aprende.

Para la integración de RAG en el chatbot, se decidió utilizar LangChain, una biblioteca que facilita la creación de aplicaciones impulsadas por modelos de lenguaje. LangChain permite definir agentes, que son componentes diseñados para decidir qué acciones tomar en función de una consulta del usuario. Estos agentes siguen una serie de pasos que optimizan la generación de respuestas. Primero, el agente recibe la consulta del usuario y analiza su intención. Luego, selecciona las herramientas y fuentes de datos más adecuadas para obtener información relevante, accediendo a bases de datos, APIs o documentos externos. Una vez obtenida la información necesaria, el agente procesa los datos y genera una respuesta final, asegurando que el contenido sea preciso y contextualizado.

Según LangChain Documentation (2024), estos agentes pueden configurarse para interactuar con diversas herramientas, desde bases de datos SQL hasta APIs de recuperación de información, optimizando así la generación de respuestas precisas y adaptadas al usuario. En el proyecto, se configuró un agente de LangChain especializado en recuperar información desde una base de datos estructurada y generar respuestas adaptadas al contexto del usuario. Esta implementación permitió mejorar la precisión y relevancia de las respuestas del chatbot, asegurando que la información proporcionada estuviera siempre actualizada y alineada con los requerimientos específicos del negocio. También se implementó un agente con la idea de que se podría en un futuro agregar otro tipo de documentos para que se recupere información de allá, como podrían ser las preguntas frecuentes (Frequently-Asked Questions).

La combinación de RAG y agentes de LangChain ha resultado ser una solución eficiente para gestionar consultas complejas, optimizar la experiencia del usuario y proporcionar información relevante de manera dinámica y confiable.

5. Implementación y prueba de métodos de refinamiento

En esta sección se describe el proceso y las herramientas utilizadas para desarrollar los cuatro modelos de chatbot implementados en el proyecto. El trabajo inició con la construcción de un modelo base, que integró la técnica Retrieval-Augmented Generation (RAG) mediante un agente de LangChain, utilizando como fuentes de información la base de datos estructurada y un archivo de texto destinado a responder preguntas frecuentes. Posteriormente, se incorporaron diferentes métodos de refinamiento con el objetivo de evaluar su impacto en la calidad de las respuestas generadas. Las técnicas probadas fueron: Few-shot learning, Fine-tuning y una combinación de Few-shot learning con Fine-tuning.

5.1. Modelo Base (RAG)

El modelo base se implementó utilizando únicamente un agente de LangChain. Para su funcionamiento, se configuraron diversas “herramientas” a las cuales el agente podía acceder para recuperar información relevante de manera ordenada y coherente. La primera herramienta correspondía a la conexión con la base de datos SQL. Esta herramienta permite al agente conectarse directamente a la base de datos y recibir información del esquema — como los nombres de tablas y columnas disponibles— para que, con ese contexto, pueda generar consultas SQL pertinentes que respondan adecuadamente a las preguntas de los usuarios.

Otra herramienta configurada fue la conexión con una fuente de datos que contenía preguntas frecuentes y sus respectivas respuestas. Para gestionar esta información de forma eficiente, se emplearon embeddings (definidos en la sección de marco teórico), permitiendo al modelo recuperar fragmentos relevantes basados en la similitud semántica de las consultas. Así, la herramienta proporciona al agente los datos necesarios para construir respuestas contextualizadas a las preguntas de los usuarios.

Finalmente, se integró una herramienta para la configuración del prompt y la memoria conversacional. Esta herramienta cumple dos funciones principales: en primer lugar, proporciona instrucciones específicas al modelo sobre el tipo de respuestas que debe generar, el formato esperado y las categorías de preguntas que puede o no puede atender; en segundo lugar, permite almacenar el historial de la conversación, otorgando al agente la capacidad de mantener coherencia y continuidad lógica a lo largo del diálogo con el usuario.

En el **Anexo 7** se encuentra un diagrama detallado del flujo que utiliza el modelo para acceder a las herramientas y responder al usuario. Mientras que en los **Anexos 8 y 9** se muestra el código utilizado y ejemplos de las respuestas del modelo respectivamente. Este modelo es el base porque se utilizan las herramientas que se van a implementar en todos los modelos, solo que en los demás, también se agregan otras técnicas de refinamiento con la intención de mejorar la calidad de las respuestas del modelo.

5.2. Modelo Few-shot

El modelo Few-shot es muy similar al modelo base explicado en el punto anterior, utilizando las mismas herramientas y el mismo flujo de operación. La única diferencia radica en la configuración del prompt. La técnica de Few-shot busca mejorar la calidad de las respuestas brindando ejemplos explícitos dentro del prompt, mostrando al modelo cómo debería responder en situaciones específicas. En este caso, se observó que el modelo base presentaba dificultades en la precisión de las respuestas que requerían consultas a la base de datos SQL. Por tanto, la implementación del Few-shot (al igual que los siguientes métodos de refinamiento) se enfocó en mejorar la calidad de las consultas SQL y las respuestas derivadas.

Según Loukas et al. (2023), para lograr un mejor desempeño en escenarios de pocos ejemplos, se recomienda incluir en el prompt no solo la pregunta del usuario, sino también

una interpretación de la tarea requerida, la ejecución correcta de la consulta SQL, y la respuesta final que el modelo debería generar. Además, los autores sugieren incorporar entre tres y cinco ejemplos en el prompt para fomentar la generalización de los pasos lógicos, evitando que el modelo simplemente memorice los ejemplos presentados. Incluir más ejemplos de los necesarios puede inducir al modelo a copiar literalmente los patrones de los ejemplos, en lugar de aplicar los principios generales aprendidos a nuevas consultas.

En el **Anexo 10** se encuentra un diagrama detallado del flujo que utiliza el modelo para acceder a las herramientas y responder al usuario, mientras que en los **Anexos 11 y 12** se muestran el código utilizado y ejemplos de respuestas generadas, respectivamente. Los resultados preliminares indicaron una mejora en la precisión y relevancia de las respuestas generadas para consultas SQL.

5.3. Modelo Fine-tuned

Para el modelo Fine-tuned, se entrenó el LLM utilizando datos específicamente diseñados para consultas SQL. OpenAI menciona que, con tan solo 10 ejemplos de alta calidad, se pueden observar mejoras notables en el desempeño del modelo; sin embargo, sugiere utilizar entre 25 y 100 ejemplos para obtener resultados más consistentes. Es importante destacar que la calidad de los datos es fundamental, ya que errores en el conjunto de entrenamiento pueden derivar en un modelo defectuoso. Por ello, se generaron 100 ejemplos de entrenamiento, cada uno compuesto por un prompt de sistema simple, la pregunta del usuario y la consulta SQL correspondiente que permitiría recuperar la respuesta correcta de la base de datos. Todas las preguntas de entrenamiento fueron formuladas de manera clara y con el formato adecuado.

Durante el proceso, se entrenaron modelos utilizando 25, 50, 75 y 100 ejemplos, realizando posteriormente una evaluación preliminar mediante un set de 30 preguntas. Cada modelo fue evaluado en función de su precisión, calificando las respuestas como correctas o incorrectas.

Sorprendentemente, el modelo entrenado con apenas 25 ejemplos obtuvo el mejor desempeño, logrando una precisión del 86,7%. Por esta razón, este modelo fue seleccionado para integrar el flujo del sistema Fine-tuned.

En el caso de este modelo, el flujo de operación difiere notablemente respecto a los anteriores. No fue posible utilizar directamente el agente de LangChain, ya que los modelos Fine-tuned dejan de ser compatibles con dicho funcionamiento, particularmente cuando están especializados en tareas específicas como la generación de consultas SQL. Por lo tanto, se desarrolló un flujo personalizado a través de funciones programadas que replican las principales funcionalidades ofrecidas por LangChain.

En primer lugar, se implementó un clasificador de preguntas utilizando el mismo modelo LLM de OpenAI, instruido para categorizar las consultas en cuatro tipos: SQL, FAQ (Frequently Asked Questions), solicitudes de información adicional y preguntas fuera de alcance. Una vez clasificada, cada consulta se redirige a la función correspondiente.

Para consultas de información adicional, se envía un mensaje predeterminado con información general del condominio. En el caso de preguntas fuera de alcance (aquellas sin relación con el negocio), el modelo responde de manera cortés indicando que solo puede atender consultas relacionadas con los servicios ofrecidos.

Cuando la consulta corresponde a FAQ o SQL, el flujo se vuelve más específico. Para FAQs, el sistema accede a una fuente de datos basada en embeddings, recuperando una respuesta relevante si está disponible, o en su defecto, ofreciendo la opción de contactar al administrador. Esta funcionalidad reproduce el comportamiento del agente LangChain, pero mediante programación manual.

Para consultas SQL, la pregunta pasa primero por un modelo reformulador que mejora su estructura para facilitar una interpretación correcta. Este modelo también detecta preguntas ambiguas y solicita al usuario la información faltante antes de proceder. Una vez reformulada, la consulta se envía al modelo Fine-tuned, que genera la instrucción SQL correspondiente. La consulta se ejecuta en la base de datos, y el resultado es procesado por otro modelo LLM encargado de traducir la respuesta en lenguaje natural para el usuario final.

Finalmente, todo el intercambio se almacena en una memoria conversacional que permite al sistema mantener coherencia en los diálogos posteriores, preservando el contexto y la continuidad de la conversación.

En el **Anexo 13** se presenta un diagrama detallado del flujo utilizado para acceder a las funciones y responder al usuario, mientras que en los **Anexos 14 y 15** se muestran el código implementado y ejemplos de respuestas generadas, respectivamente. A pesar de su mayor complejidad de integración, el modelo Fine-tuned demostró una mejora notable en la calidad de las respuestas estructuradas, especialmente en consultas SQL complejas, en comparación con el modelo base.

5.4. Modelo combinado (Few-shot + Fine-tuned)

El modelo combinado implementa la misma arquitectura general que el modelo Fine-tuned descrito en la sección anterior. La principal diferencia radica en que, además de utilizar el modelo entrenado para la generación de consultas SQL, se incorporaron ejemplos de Few-shot learning dentro del prompt de entrada.

Estos ejemplos incluían la pregunta del usuario, su interpretación lógica, la consulta SQL esperada y la respuesta generada. El objetivo de incorporar estos ejemplos fue reforzar la estructura lógica deseada en las respuestas producidas por el modelo Fine-tuned, buscando

optimizar aún más la precisión y la coherencia en la generación de consultas SQL. La selección de ejemplos se realizó siguiendo las mejores prácticas descritas en la sección 5.2, incorporando entre tres y cinco ejemplos cuidadosamente elegidos para evitar el sobreajuste y preservar la capacidad de generalización del modelo.

Toda la arquitectura base del flujo de atención al cliente se mantuvo intacta: clasificación de preguntas, reformulación de consultas ambiguas, generación de consultas SQL, ejecución en la base de datos, generación de respuestas en lenguaje natural y gestión de memoria conversacional.

En el **Anexo 16** se presenta un diagrama actualizado que ilustra el flujo de operación del modelo combinado, mientras que en los **Anexos 17 y 18** se incluyen fragmentos de código y ejemplos de respuestas generadas, respectivamente. En las pruebas preliminares, este modelo mostró una mejora adicional en la precisión de generación de consultas SQL respecto al modelo base.

6. Despliegue y pruebas

6.1. Entorno de pruebas

Debido a limitaciones de tiempo y al hecho de que el chatbot aún no se ha desplegado oficialmente en el canal de WhatsApp, se implementó un entorno de pruebas controlado para evaluar su desempeño.

El entorno de pruebas consistió en una simulación manual de interacciones reales, replicando conversaciones basadas en históricos de mensajes recibidos a través de WhatsApp en el negocio. Las preguntas fueron enviadas directamente al sistema utilizando scripts de prueba en Python, lo que permitió validar tanto la recuperación de información como la generación de respuestas en un contexto similar al operativo real.

Este enfoque permitió observar el comportamiento del chatbot en condiciones controladas, asegurando que los flujos de atención (clasificación de consultas, recuperación de información, generación de respuestas, y mantenimiento de contexto conversacional) funcionaran de manera integrada antes de proceder a su despliegue en un entorno de producción.

El entorno de pruebas también permitió recopilar respuestas de cada uno de los modelos desarrollados (Base, Few-shot, Fine-tuned y Combinado) bajo condiciones equivalentes, garantizando que la posterior evaluación de desempeño fuera consistente y comparativa.

6.2. Método de evaluación

Para la evaluación de cada modelo, se desarrolló un set de 120 preguntas basadas en escenarios reales de uso. Esta cantidad de preguntas fue definida considerando la recomendación de Amidei, Piwek y Willis (2020), quienes sugieren utilizar entre 50 y 200 preguntas en evaluaciones manuales de modelos de generación de lenguaje natural, cuando la clasificación no puede ser completamente automatizada.

La necesidad de una evaluación manual surge debido a que las respuestas generadas por los modelos no siempre pueden ser clasificadas de forma binaria (correcta o incorrecta), ya que deben ser interpretadas considerando tanto la pregunta original como el contexto de negocio en el cual se desarrolló el chatbot.

El set de preguntas fue dividido en cuatro categorías, de acuerdo con la naturaleza de la información requerida para responder correctamente:

- SQL: Consultas relacionadas a precios y disponibilidad de departamentos.
- FAQs: Preguntas sobre reglas, condiciones o servicios disponibles en el condominio.

- Ambiguas: Consultas que carecen de información suficiente para ser respondidas directamente (por ejemplo, falta de fechas o número de personas).
- Fuera de alcance: Preguntas que no guardan relación con el negocio o exceden las capacidades previstas del chatbot.

El objetivo de esta clasificación fue simular un amplio rango de situaciones posibles en la atención real al cliente, asegurando que los modelos fueran evaluados en distintos tipos de interacciones. Ejemplos representativos de cada categoría de preguntas pueden encontrarse en el **Anexo 19**.

Para calificar las respuestas generadas por los modelos, se utilizó una escala de Likert de 1 a 5, basada en la calidad, precisión y relevancia de las respuestas. Esta metodología de evaluación se encuentra respaldada por Zeng et al. (2024) y Amidei, Piwek y Willis (2020), quienes recomiendan el uso de escalas ordinales en evaluaciones de modelos de lenguaje, debido a la complejidad inherente en valorar respuestas semánticamente diversas.

Con el objetivo de reducir sesgos y mantener consistencia en la evaluación, se definieron claramente los siguientes criterios de calificación:

Tabla 1: Criterios para calificaciones del set de evaluación.

Calificación	Criterio
1	Respuesta con múltiples errores o completamente incorrecta
2	Respuesta con un error grave o dos errores menores
3	Respuesta con un error menor
4	Respuesta no esperada, pero válida y de buena calidad
5	Respuesta esperada, precisa y completa

Esta escala permitió capturar de forma más realista la calidad de las respuestas generadas, ofreciendo un marco sistemático para la comparación entre los diferentes modelos desarrollados.

6.3. Procedimiento

Cada uno de los modelos desarrollados —Modelo Base, Modelo Few-shot, Modelo Fine-tuned y Modelo combinado— fue evaluado de manera independiente utilizando el set de 120 preguntas descrito previamente.

El procedimiento de evaluación consistió en enviar las mismas preguntas a cada modelo, registrando las respuestas generadas en condiciones controladas. De esta forma, se garantizó que todos los modelos enfrentaran escenarios equivalentes y pudieran ser comparados objetivamente bajo los mismos criterios.

Posteriormente, cada respuesta fue calificada individualmente utilizando la escala de Likert previamente establecida. Este proceso fue realizado manualmente por el evaluador, quien analizó tanto la pregunta original como la respuesta generada para asignar una calificación de 1 a 5 en función de la calidad, precisión y relevancia.

Para analizar las diferencias de desempeño entre los modelos, se aplicó la prueba de Friedman, una prueba estadística no paramétrica adecuada para comparar tratamientos en muestras relacionadas y datos de naturaleza ordinal. La hipótesis nula planteada fue que no existen diferencias significativas entre los modelos evaluados, mientras que la hipótesis alternativa establecía que sí existen diferencias.

En los casos donde se identificaron diferencias significativas mediante la prueba de Friedman ($p < 0.05$), se procedió a realizar un análisis de comparaciones múltiples utilizando el Test de Nemenyi, con el fin de determinar entre qué modelos se presentaban las diferencias observadas.

Este enfoque permitió no solo evaluar la calidad individual de las respuestas, sino también comparar de manera estadísticamente válida el desempeño general de cada estrategia de refinamiento aplicada en el chatbot.

7. Resultados

7.1. Resultados Generales de evaluación

Se realizó la evaluación de los cuatro modelos desarrollados —Modelo Base, Modelo Few-shot, Modelo Fine-tuned y Modelo combinado— utilizando el set de 120 preguntas elaborado a partir de escenarios reales de uso.

Cada respuesta fue calificada utilizando la escala de Likert de 1 a 5, de acuerdo a criterios de calidad, precisión y relevancia previamente establecidos.

Los resultados promedio de calificación para cada modelo fueron los siguientes:

Tabla 2: Calificación promedio de modelos.

Modelo	Calificación
Base	3.59
Few-Shot	3.83
Fine-Tuning	4.24
Combinada	4.65

La aplicación de la prueba de Friedman para comparar los resultados entre los modelos arrojó un valor $p = 0.00$, indicando diferencias estadísticamente significativas en el desempeño de los modelos evaluados con un nivel de significancia de $\alpha = 0.05$. Este resultado sugiere que al menos uno de los modelos ofrece un desempeño significativamente mejor en la generación de respuestas que los demás.

Posteriormente, se aplicó la prueba de Nemenyi para identificar entre qué pares de modelos existían diferencias significativas en su desempeño general. A continuación, se muestran los resultados.

Tabla 3: Comparación pareada en evaluación general.

Pares	p-valor prueba de Nemenyi	Diferencias significativas
Base vs Few-shot	0.81	No
Base vs Fine-tuned	0.04	Si
Base vs Combinado	0.00	Si
Few-shot vs Fine-tuned	0.29	No
Few-shot vs Combinado	0.01	Sí
Fine-tuned vs Combinado	0.57	No

Estos resultados, interpretados junto a la prueba de Friedmann, demuestran que el modelo combinado obtiene una mejora significativa con respecto a los modelos Few-shot y Base con un nivel de significancia de $\alpha = 0.05$. Sin embargo, no hay suficiente evidencia de mejoras estadísticamente significativas con respecto al modelo Fine-tuned.

7.2. Comparación entre modelos por categorías de pregunta

Después, se evaluaron los modelos por categorías, primero observando con una prueba de Friedman si los modelos demostraban comportamientos significativamente diferentes entre sí. Los resultados se pueden observar a continuación.

Tabla 4: Resultados de prueba de Friedman por categoría de preguntas.

Categoría	P-valor de la Prueba de Friedman	Diferencias Significativas
Consultas SQL	0.00	Sí
FAQs	0.17	No
Consultas Ambiguas	0.00	Sí
Fuera de Alcance	0.06	No

Estos resultados muestran que hay diferencias significativas entre por lo menos 2 de los modelos en las categorías de Consultas de SQL y Consultas Ambiguas. Por lo tanto, se procedió a hacer una prueba pareada de Namenyi en esas categorías para identificar el mejor modelo para cada caso.

Tabla 5: Comparación pareada en evaluación de categoría consultas de SQL.

Pares	p-valor prueba de Namenyi	Diferencias significativas
Base vs Few-shot	0.32	No
Base vs Fine-tuned	0.06	No
Base vs Combinado	0.00	Si
Few-shot vs Fine-tuned	0.85	No
Few-shot vs Combinado	0.04	Sí
Fine-tuned vs Combinado	0.22	No

Estos resultados, interpretados junto a la prueba de Friedmann, demuestran que el modelo combinado obtiene una mejora significativa con respecto a los modelos Few-shot y Base con un nivel de significancia de $\alpha = 0.05$. Sin embargo, no hay suficiente evidencia de mejoras estadísticamente significativas con respecto al modelo Fine-tuned para preguntas que requieren el uso de SQL.

Tabla 6: Comparación pareada en evaluación de categoría de consultas Ambiguas.

Pares	p-valor prueba de Namenyi	Diferencias significativas
Base vs Few-shot	0.65	No
Base vs Fine-tuned	0.01	Si
Base vs Combinado	0.02	Si
Few-shot vs Fine-tuned	0.14	No
Few-shot vs Combinado	0.27	No

Fine-tuned vs Combinado	0.98	No
-------------------------	------	----

Estos resultados, interpretados junto a la prueba de Friedmann, demuestran que el modelo Finetuned y Combinado tienen mejoras significativas con respecto al modelo Base. Sin embargo, entre los modelos Few-shot, Fine-tuned y Combinado, no hay suficiente evidencia estadística de que un modelo sea mejor que el resto respondiendo preguntas ambiguas.

7.3. Modelo de mejor desempeño

De acuerdo con los resultados obtenidos en la evaluación general y en las comparaciones por tipo de pregunta, el modelo que combinó Fine-tuning y Few-shot learning demostró ser el de mejor desempeño global por promedio.

Este modelo no solo alcanzó la calificación promedio más alta en la evaluación general, sino que además mostró un desempeño significativamente superior en categorías críticas como Consultas SQL y Consultas Ambiguas, según lo confirmado por la prueba de Friedman ($p = 0.00$) y las comparaciones múltiples realizadas mediante el Test de Nemenyi.

El análisis post-hoc evidenció que el modelo combinado superó al modelo Base y al modelo Few-shot de manera estadísticamente significativa, mientras que las diferencias con el modelo Fine-tuned aislado no fueron significativas. Sin embargo, el modelo combinado mostró una tendencia consistente hacia una mayor precisión y naturalidad en las respuestas, especialmente en consultas de mayor complejidad.

La integración de ejemplos Few-shot dentro del flujo Fine-tuned permitió reforzar la estructura lógica esperada en las consultas SQL, mientras que se mantenía un nivel adecuado de flexibilidad conversacional, equilibrando así control y adaptabilidad en la interacción con los usuarios.

En base a estos resultados, se concluye que la estrategia de refinamiento combinada representa la mejor alternativa para optimizar la atención automatizada en el contexto de la renta de departamentos vacacionales, maximizando tanto la precisión de las respuestas como la experiencia del usuario.

7.4. Limitaciones

A pesar de los resultados positivos obtenidos, el proyecto presenta algunas limitaciones que deben ser consideradas para futuras implementaciones y evaluaciones más amplias:

- Evaluación simulada: Las pruebas de desempeño del chatbot se realizaron en un entorno de pruebas controlado, simulando conversaciones basadas en históricos de consultas reales, pero no en un entorno de producción activo. Esto puede limitar la representación exacta de las condiciones de uso reales, donde las interacciones de los usuarios pueden ser más variables y dinámicas.
- Tamaño de muestra limitado: La evaluación se basó en un set de 120 preguntas, elaborado cuidadosamente para simular distintos escenarios. Sin embargo, una muestra mayor y variada podría permitir una caracterización aún más robusta del desempeño del chatbot en múltiples tipos de interacción.
- Evaluador único: La calificación de las respuestas generadas fue realizada por un solo evaluador, lo que introduce el riesgo de sesgo individual, a pesar de haberse definido criterios objetivos de calificación. Futuras evaluaciones podrían beneficiarse de contar con múltiples evaluadores independientes para garantizar aún mayor robustez.
- Dependencia de la calidad de los datos: El desempeño del modelo, especialmente del componente Fine-tuned, depende fuertemente de la calidad de los datos utilizados para el entrenamiento. Inconsistencias, errores o falta de cobertura en los datos podrían

impactar negativamente la calidad de las respuestas generadas. Y estos datos, al igual que la evaluación, fueron observadas y aprobadas por un solo evaluador.

- Escalabilidad y costos de operación: Aunque el sistema funcionó adecuadamente en pruebas controladas, no se evaluaron aún los costos de operación en producción real a gran escala, particularmente el costo de consumo de API de conexión a WhatsApp y de los servidores en la nube que serían necesarios para que el sistema funcione de forma constante.
- Limitaciones de tiempo: Las API de plataformas como WhatsApp utilizan un tiempo máximo de ejecución de funciones como la del chatbot, por lo que en un futuro sería necesario garantizar un flujo optimizado para garantizar una comunicación fluida entre el chatbot y el usuario. Este componente es de gran importancia para la implementación del chatbot en un contexto de producción real.

Estas limitaciones no invalidan los resultados obtenidos, pero proporcionan un marco para entender los alcances del proyecto y una base para definir líneas de mejora y trabajo futuro.

8. Conclusiones

El presente proyecto tuvo como objetivo general desarrollar un chatbot basado en inteligencia artificial que automatizara la atención al cliente en un negocio de renta de departamentos vacacionales, mejorando la eficiencia y calidad de las respuestas ofrecidas a los usuarios. A través del desarrollo de múltiples versiones del modelo y su evaluación formal, se pueden establecer las siguientes conclusiones:

- Automatización efectiva de la atención al cliente: Se logró implementar un sistema que integra generación aumentada por recuperación (RAG) y modelos de lenguaje refinados, capaz de atender consultas de forma precisa y en tiempo real, cumpliendo así con el objetivo general planteado.

- Integración exitosa de RAG con base de datos: El chatbot fue capaz de conectarse a una base de datos estructurada mediante técnicas RAG, recuperando información actualizada para responder preguntas de disponibilidad y precios de los departamentos, cumpliendo el primer objetivo específico.
- Optimización del desempeño mediante técnicas de refinamiento: Se probaron distintos métodos de mejora, incluyendo Few-shot learning, Fine-tuning, y una estrategia combinada. La evaluación estadística demostró que el modelo combinado (Fine-tuned + Few-shot) ofreció el mejor desempeño, cumpliendo el segundo objetivo específico de identificar la técnica de refinamiento más efectiva.
- Reducción del tiempo de respuesta y potencial aumento de conversión: La simulación de respuestas mostró tiempos de respuesta inmediatos y consistentes, contrastando con los tiempos promedio anteriores donde se evidenciaban demoras importantes. Esto respalda el tercer objetivo específico de reducir los tiempos de atención, contribuyendo potencialmente a mejorar la tasa de conversión de consultas a reservas.
- Minimización de la carga operativa del administrador: Con la implementación del chatbot, se automatizan las consultas repetitivas que previamente requerían una alta dedicación manual, cumpliendo el cuarto objetivo específico de optimizar el uso del tiempo del operador humano.
- Simulación de conversaciones naturales: Especialmente en la versión combinada, el chatbot fue capaz de mantener una interacción fluida y coherente, logrando no solo precisión técnica sino también una experiencia conversacional agradable para el usuario, en línea con el quinto objetivo específico.

Adicionalmente, los hallazgos subjetivos del proceso de evaluación reflejan que las técnicas de refinamiento no solo mejoran la exactitud de las respuestas, sino que también influyen en

el estilo de interacción del chatbot, un aspecto crítico para la aceptación por parte de los usuarios.

En conclusión, el proyecto demostró que es viable implementar un sistema inteligente de atención al cliente basado en modelos de lenguaje avanzados, adaptado a un negocio real de renta vacacional, logrando optimizar procesos operativos clave y mejorar la experiencia de usuario de manera significativa.

9. Recomendaciones

A partir de los resultados obtenidos en el presente proyecto y considerando las observaciones realizadas durante el proceso de desarrollo y pruebas, se plantean las siguientes recomendaciones para futuras etapas de implementación y optimización:

- **Despliegue en un entorno real de producción:** Se recomienda integrar el chatbot en el canal oficial de WhatsApp del negocio y realizar una prueba piloto durante un período prolongado (mínimo tres a seis meses), monitoreando indicadores clave como tiempo de respuesta, tasa de conversión de consultas en reservas y satisfacción del cliente.
- **Monitoreo continuo del desempeño:** Es importante establecer un sistema de seguimiento del desempeño del chatbot en producción, permitiendo identificar desviaciones en la calidad de las respuestas y ajustar los modelos o bases de datos conforme evolucione la operación.
- **Investigación sobre ajuste de hiperparámetros:** Para futuras mejoras, se sugiere explorar técnicas de ajuste fino de hiperparámetros en modelos Fine-tuned, como el tamaño del batch, tasa de aprendizaje o número de épocas, con el objetivo de optimizar aún más la precisión y la estabilidad de las respuestas.
- **Ampliación de la base de conocimiento:** Se recomienda incorporar fuentes de información adicionales, como políticas detalladas del condominio, recomendaciones

turísticas locales o guías de uso de las instalaciones, expandiendo así el dominio de respuestas del chatbot.

- **Análisis de costo-beneficio en escalabilidad:** En caso de incremento del volumen de consultas, se sugiere evaluar alternativas de despliegue que reduzcan costos, como el uso de modelos open-source ligeros en servidores propios o híbridos, en lugar de depender exclusivamente de APIs comerciales.
- **Considerar retroalimentación directa de usuarios:** Se recomienda implementar mecanismos para recoger retroalimentación de usuarios reales tras sus interacciones con el chatbot, como encuestas de satisfacción breves, para detectar áreas de mejora no evidentes en métricas técnicas.

Estas recomendaciones buscan garantizar la evolución continua del chatbot, su adaptabilidad a nuevas necesidades del negocio, y su contribución sostenida a la eficiencia operativa y la experiencia del cliente en el contexto de la renta de departamentos vacacionales.

10. Referencias

- Alqudah, M., & Razali, R. (2018). *An Empirical Study of Scrumban Formation based on the Selection of Scrum and Kanban Practices*.
- Crisafulli, B., & Singh, J. (2017). Service failures in e-retailing: Examining the effects of response time, compensation, and service criticality. *Computers in Human Behavior*, 77, 413–424. <https://doi.org/10.1016/j.chb.2017.07.013>
- Durango Vanegas, C. E., et al. (2023). *A Representation Based on Essence for the CRISP-DM Methodology*.
- Gordon Graell, R. D. (2023). *Chatbots e inteligencia artificial: aportes e innovaciones en sistemas de información empresarial*.
- Kraus, S., Oshrat, Y., Aumann, Y., Hollander, T., Maksimov, O., Ostroumov, A., & Shechtman, N. (2023). Customer service combining human operators and Virtual Agents: A call for multidisciplinary AI research. *Proceedings of the AAAI Conference on Artificial Intelligence*, 37(13), 15393–15401. <https://doi.org/10.1609/aaai.v37i13.26795>
- LangChain Documentation. (2024). *Agents in LangChain: How They Work and When to Use Them*. Disponible en: python.langchain.com.
- Loukas, G., Copara, J., & Lo, B. (2023). *Best practices for prompting language models: A guide to few-shot learning*. arXiv. <https://doi.org/10.48550/arXiv.2311.06102>
- Nankya, V., & Oppenheimer, M. (2025). *Comparing RAG and Fine-Tuned LLMs to ChatGPT*.
- Nassiri, K., & Akhloufi, M. (2022). Transformer models used for text-based question answering systems. Springer Nature. <https://doi.org/10.1007/s10489-022-04052-8>
- OpenAI. (2024). *Embeddings*. OpenAI. <https://platform.openai.com/docs/guides/embeddings>
- Pionce Arteaga, M. A., Caicedo Plúa, C. R., Delgado Lucas, H. B., & Murillo Quimiz, L. R. (2022). Chatbots para ventas y atención al cliente. *Journal TechInnovation*, 1(1), 107–116. <https://doi.org/10.47230/Journal.TechInnovation.v1.n1.2022.107-116>
- Pornprasita, C., & Tantithamthavorn, C. (2024). *Fine-Tuning and Prompt Engineering for Large Language Models*.
- Sai Mounika. (2024). Exploring AI-Driven Customer Service: Evolution, Architectures, opportunities, challenges and future directions. *International Journal For Multidisciplinary Research*, 6(3). <https://doi.org/10.36948/ijfmr.2024.v06i03.22283>
- Segovia-García, N. (2024). *Optimización de la atención estudiantil mediante chatbots de IA*.

- Sofiyah, F. R., et al. (2024). *The chatbot artificial intelligence as the alternative customer services strategic*.
- Trad, F., & Chehab, A. (2024). Prompt Engineering or Fine-Tuning? A Case Study on Phishing Detection with Large Language Models. *Machine Learning & Knowledge Extraction*, 6(1), 367–384. <https://doi.org/10.3390/make6010018>
- Valderrama, J., & Aguilar-Alonso, I. (2023). Creation of a chatbot based on natural language processing for WhatsApp. *Electrical and Electronics Engineering: An International Journal (ELELIJ)*, 12(2/3/4), 11–20. <https://doi.org/10.14810/elelij.2023.12402>
- Valdivieso Taborga, C. E., Valdivieso Castellón, R., & Valdivieso Taborga, O. Á. (2011). *Determinación del tamaño muestral mediante el uso de árboles de decisión*. *Investigación & Desarrollo*, 11(1), 53-80. <https://www.researchgate.net/publication/315138192>
- Wang, G., & Oliver, M. (2024). Crafting Efficient Fine-Tuning Strategies for Large Language Models. *Semantic Scholar*. <https://doi.org/https://doi.org/10.48550/arXiv.2407.13906>
- Wei, L., Jiang, Z., Huang, W., & Sun, L. (2023). InstructionGPT-4: A 200-Instruction Paradigm for Fine-Tuning MiniGPT-4. Qing Yuan Research Institute, Shanghai Jiao Tong University.
- Zhang, J., Chen, Q., Lu, J., Wang, X., Liu, L., & Feng, Y. (2024). Can chatbots care? The impact of chatbots' emotional expression on customer satisfaction in tourism services. *Tourism Management*, 99, 104852. <https://doi.org/10.1016/j.tourman.2023.104852>

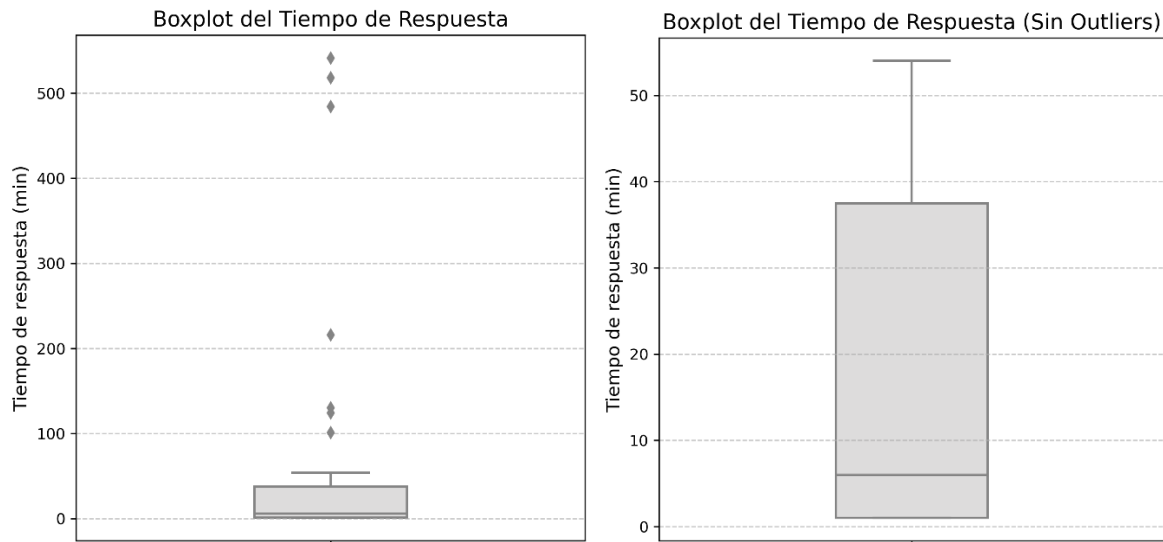
11. Anexos

Anexo 1: Tamaño de muestra tiempo promedio de respuesta. (Valdivieso Taborga et al., 2011)

$$n_0 = \left(z_{\frac{\alpha}{2}} \frac{\sigma}{e} \right)^2$$

$$n_0 = \left(1.645 \cdot \frac{129.44}{30} \right)^2 \approx 51$$

Anexo 2: Diagramas de caja de tiempo promedio de respuesta con valores extremos y sin valores extremos.



Anexo 3: Tamaño de muestra de tiempo de operador en WhatsApp. (Valdivieso Taborga et al., 2011)

$$n_0 = \left(z_{\frac{\alpha}{2}} \frac{\sigma}{e} \right)^2$$

$$n_0 = \left(1.645 \cdot \frac{95.09}{40} \right)^2 \approx 16$$

Anexo 4: Tamaño de muestra de tasa de conversión. (Valdivieso Taborga et al., 2011)

$$n_0 = \frac{P(1-P)Z_{\frac{\alpha}{2}}^2}{e^2}$$

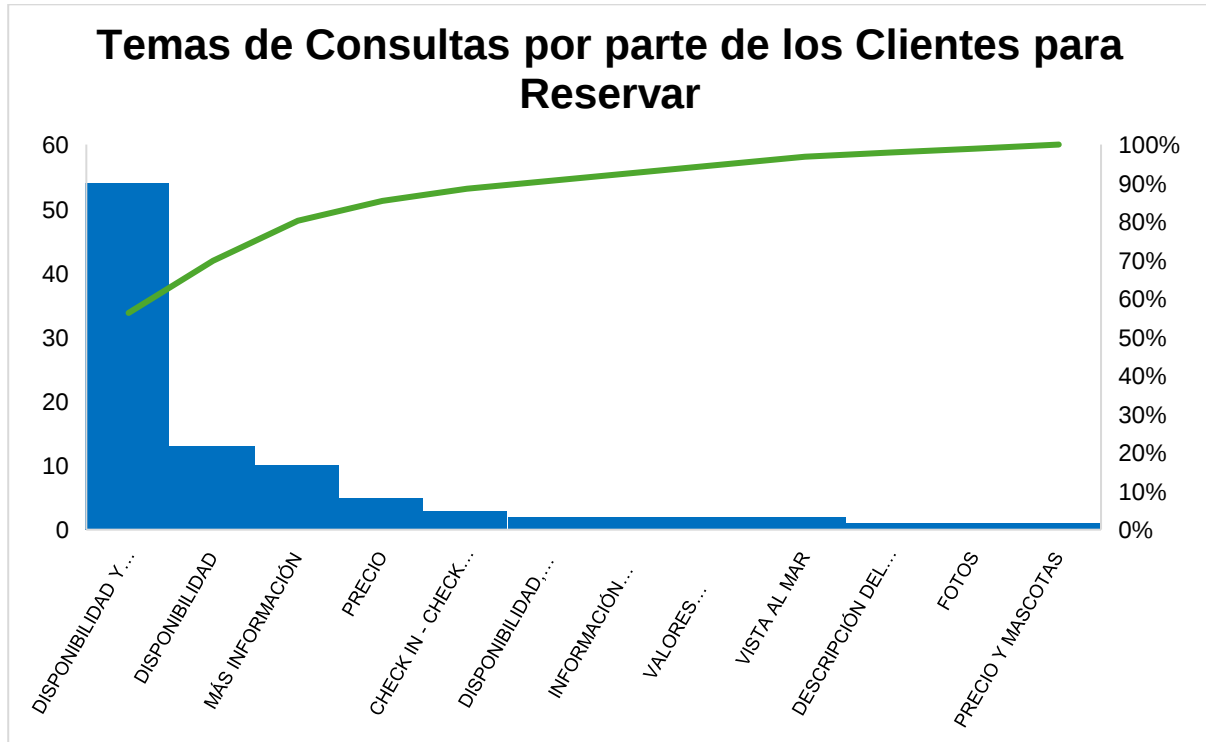
$$n_0 = \frac{0.25(1-0.25)(1.645)^2}{(0.05)^2} \approx 203$$

Anexo 5: Tamaño de muestra de proporción de mensajes por categoría. (Valdivieso Taborga et al., 2011)

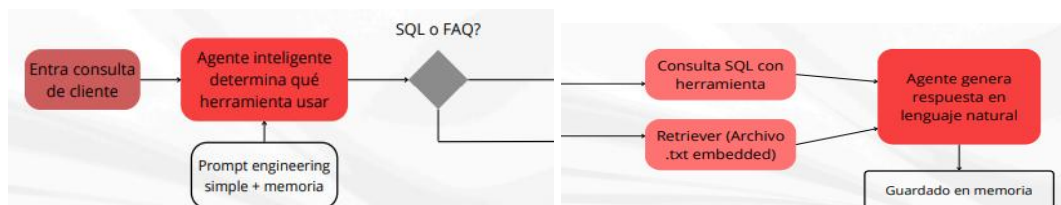
$$n_0 = \frac{P(1-P)Z_{\frac{\alpha}{2}}^2}{e^2}$$

$$n_0 = \frac{0.5(1-0.5)1.96^2}{0.10^2} \approx 96$$

Anexo 6: Pareto de requerimientos más importantes de respuesta de chatbot.



Anexo 7: Flujo de modelo Base.



Anexo 8: Código de modelo Base.

```

from langchain.memory import ConversationBufferMemory
from langchain.agents import AgentExecutor, create_openai_tools_agent
from langchain_openai import ChatOpenAI
from langchain_core.prompts import ChatPromptTemplate, MessagesPlaceholder
from langchain_community.vectorstores import FAISS
from langchain.tools.retriever import create_retriever_tool
from langchain_openai import OpenAIEmbeddings
from datetime import datetime

# 1. Configurar modelo y memoria
llm = ChatOpenAI(model="gpt-4o-mini")
memory = ConversationBufferMemory(memory_key="chat_history", return_messages=True, input_key="input")

# 2. Tool de FAQs (usando tu archivo .txt)
loader = TextLoader(r"C:\Users\mateb\Documents\USFQ\Segundo 2024\TESIS\Modelo\FAQsRPA.txt")
docs = loader.load()
text_splitter = CharacterTextSplitter(chunk_size=300, separator="\n")
chunks = text_splitter.split_documents(docs)
vector_store = FAISS.from_documents(chunks, OpenAIEmbeddings())
faq_tool = create_retriever_tool(
    vector_store.as_retriever(),
    name="faqs_informacion_general",
    description="Responde preguntas frecuentes como horarios de check-in y políticas de mascotas o servicios del condominio"
)

# 3. Tool de Base de Datos
from langchain_community.agent_toolkits import SQLDatabaseToolkit
db = SQLDatabase.from_uri(r"sqlite:///C:\Users\mateb\Documents\USFQ\Segundo 2024\TESIS\Base de Datos\TestingDB.db")
sql_toolkit = SQLDatabaseToolkit(db=db, llm=llm)
sql_tools = sql_toolkit.get_tools() # Incluye sql_db_query, sql_db_list_tables, etc.
for tool in sql_tools:
    if tool.name == "sql_db_query":
        tool.description = "Consulta SQL directa a la base de datos para conocer precios, disponibilidad y propiedades."
    elif tool.name == "sql_db_schema":
        tool.description = "Muestra la estructura (columnas y tipos de datos) de las tablas de la base de datos."
    elif tool.name == "sql_db_list_tables":
        tool.description = "Lista todas las tablas disponibles en la base de datos."
    elif tool.name == "sql_db_query_checker":
        tool.description = "Revisa si una consulta SQL está bien estructurada antes de ejecutarla."

# 4. Prompt Engineering
# Obtener la fecha actual en español
fecha_actual = datetime.today().strftime("%d de %B de %Y")

system_prompt = f"""
Hoy es {fecha_actual}.

Eres un asistente inteligente que ayuda a las personas a consultar y reservar departamentos vacacionales en la playa en Resort Playa Azul. Responde exclu
Estás conectado a una base de datos con información sobre propiedades, precios y disponibilidad, y también tienes acceso a documentos con información gen

Tu objetivo es entender claramente qué necesita el usuario, buscar la información correcta y responder con claridad, amabilidad y precisión.

Cuando debas responder preguntas relacionadas con:
- **Fechas, precios, disponibilidad, capacidad o número de personas**: usa la base de datos (herramientas que comienzan con `sql_db_`).
- **Reglas generales, mascotas, check-in/check-out, servicios incluidos o normas del condominio**: usa las FAQs (herramienta `faq_info_general`).

### Sobre la base de datos:
Antes de responder preguntas que requieren acceso a la base de datos, revisa primero la estructura de las tablas usando la herramienta `sql_db_schema`, p
- La tabla `propiedades` contiene los departamentos disponibles, con datos como capacidad y número de dormitorios (`tipo_dormitorios`). Refierete a esto
- La tabla `disponibilidad` indica si una propiedad está reservada (`reservado = TRUE`) en una fecha específica.
- La tabla `precios` define el precio por día, dependiendo del día de la semana (`dia_semana`), la temporada (`id_temporada`) y el tipo de propiedad.
- La tabla `temporadas` indica si una fecha pertenece a un feriado o a una temporada especial.

Para calcular el precio total de una reserva:
1. Recorre cada día dentro del rango solicitado.
2. Determina qué día de la semana es (ej. lunes, viernes).
3. Verifica si ese día está dentro de una temporada.
4. Busca el precio correspondiente en la tabla `precios`.

Una propiedad está disponible solo si **no está reservada** en ninguna de las fechas del rango.

### Comportamiento esperado:
- Si el usuario menciona fechas como "el 27", asume que se refiere al 27 de marzo de 2025, a menos que se indique otra cosa. Lo mismo si se refiere al "f
- Si la pregunta es ambigua o le falta información, como fechas o cantidad de personas, **pide esa información claramente** antes de responder.
- Si el usuario mezcla temas (por ejemplo, disponibilidad y política de mascotas), **responde a ambos temas si puedes**.
- Si no sabes algo o no tienes acceso a esa información, **no inventes**: indica que no sabes o que puedes verificarlo con alguien.
- Usa un tono cálido, profesional y claro, como si estuvieras atendiendo a alguien que quiere reservar sus vacaciones.

Responde siempre en español neutro. Sé amable con los clientes y ayúdales para que logremos conversiones!

```

```

prompt = ChatPromptTemplate.from_messages([
    ("system", system_prompt),
    MessagesPlaceholder("chat_history"),
    ("human", "{input}"),
    MessagesPlaceholder("agent_scratchpad")
])

# 5. Crea y ejecuta el agente
agent = create_openai_tools_agent(llm, [faq_tool] + sql_tools, prompt)
agent_executor = AgentExecutor(
    agent=agent,
    tools=[faq_tool] + sql_tools,
    memory=memory,
    verbose=True # Desactiva en producción
)

```

Anexo 9: Ejemplos de funcionamiento y respuesta de modelo Base.

```

)

/*
3 rows from precios table:
tipo_dormitorios    tipo_temporada    dia_semana    precio
1      BAJA      LUNES      70.0
1      BAJA      MARTES      70.0
1      BAJA      MIERCOLES      70.0
*/

CREATE TABLE propiedades (
    id INTEGER,
    nombre TEXT NOT NULL,
    tipo_dormitorios INTEGER,
    capacidad INTEGER NOT NULL,
    PRIMARY KEY (id),
    CHECK (tipo_dormitorios IN (1, 2, 3))
)

```

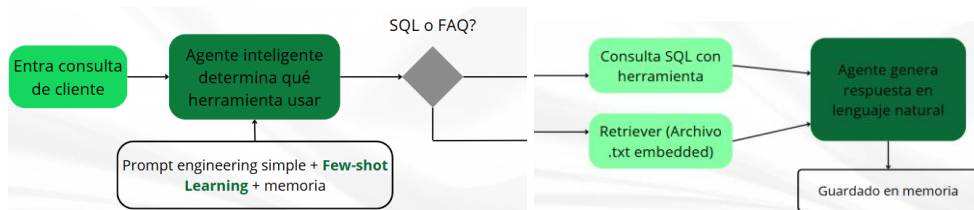
```
pregunta = "Cuanto cuesta un depar para 6 del 11 al 13?"
print(responder_chatbot_Base(pregunta))
```

Opciones de Departamentos

```
1. **Departamento 17018**
- **Tipo de dormitorios:** 2
- **Capacidad:** 6 personas
- **Precios:**
  - **11 de abril (Viernes):** $110.00
  - **12 de abril (Sábado):** $120.00
  - **13 de abril (Domingo):** $90.00
  - **Total:** $110 + $120 + $90 = $320.00

2. **Departamento 17038**
- **Tipo de dormitorios:** 3
- **Capacidad:** 8 personas
- **Precios:**
  - **11 de abril (Viernes):** $140.00
  - **12 de abril (Sábado):** $150.00
```

Anexo 10: Flujo de modelo Few-shot.



Anexo 11: Fracción de código de modelo Few-shot (Ejemplos en prompt).

```
Para calcular el precio total de una reserva:
1. Recorre cada día dentro del rango solicitado.
2. Determina qué día de la semana es (por ejemplo, lunes, viernes).
3. Consulta la tabla `temporadas` para saber a qué tipo de temporada pertenece cada fecha (priorizando feriado si aplica).
4. Busca el precio correspondiente en la tabla `precios` según `tipo_dormitorios`, `tipo_temporada` y `dia_semana`.

Una propiedad está disponible solo si **no está reservada en ninguna de las fechas** dentro del rango solicitado.

### Comportamiento esperado:
- Si el usuario menciona fechas como "el 27", asume que se refiere al 27 de marzo de 2025, a menos que se indique otra cosa. Lo mismo si se refiere al "f
- Si la pregunta es ambigua o le falta información, como fechas o cantidad de personas, **pide esa información claramente** antes de responder.
- Si el usuario te proporciona la cantidad de personas y la fecha o fechas, deberías tener suficiente información para responder. Si tienes esta informac
- Si te preguntan disponibilidad para más de 8 personas, puedes ofrecer 2 departamentos. Por ejemplo, si alguien te pide para 12 personas, puedes ofrecer
- Si hay muchas opciones disponibles, **muestra solo las 2 primeras más económicas** y dile al usuario que puede pedirte ver más opciones si lo desea y *
- Si el usuario mezcla temas (por ejemplo, disponibilidad y política de mascotas), **responde a ambos temas si puedes**.
- Si no sabes algo o no tienes acceso a esa información, **no inventes**: indica que no sabes o que puedes verificarlo con alguien.
- Usa un tono cálido, profesional y claro, como si estuvieras atendiendo a alguien que quiere reservar sus vacaciones.

Responde siempre en español neutro. Sé amable con los clientes y ayúdales para que logremos conversiones!

### Ejemplos de cómo resolver solicitudes
A continuación, algunos ejemplos de cómo traducir las solicitudes del usuario a consultas SQL y respuestas correctas:

### EJEMPLO 1
Usuario: Quiero reservar un departamento para 5 personas, del 18 al 20 de abril.
```

```

JOIN fechas_completas f ON 1=1
JOIN precios p ON p.tipo_dormitorios = pr.tipo_dormitorios
                AND p.tipo_temporada = f.tipo_temporada
                AND p.dia_semana = f.dia_semana
WHERE pr.capacidad >= 5
)
SELECT
  id_propiedad,
  nombre,
  capacidad,
  SUM(precio) AS total,
  GROUP_CONCAT(precio || ' (' || fecha || ' - ' || dia_semana || ' - ' || tipo_temporada || ')', ', ') AS detalle
FROM precio_por_fecha
GROUP BY id_propiedad, nombre, capacidad
HAVING COUNT(*) = 2;

```

Respuesta:

Tenemos las siguientes opciones disponibles para 5 personas entre el 18 y 20 de abril:

- Departamento 1701B: \$360 por 2 noches + \$180 (viernes feriado) + \$180 (sábado feriado)
- Departamento 1703B: \$480 por 2 noches + \$240 (viernes feriado) + \$240 (sábado feriado)

Si deseas proceder con la reserva o tienes más preguntas, ¡avísame!

EJEMPLO 2:

Usuario: ¿Cuánto cuesta una noche el sábado 5 para 4 personas?

SQL:

```

-- Esta consulta busca propiedades con capacidad >= 4,
-- que estén disponibles el sábado 5 de abril de 2025,
-- y obtiene el precio por noche considerando si es feriado.
-- Se prioriza tipo_temporada 'FERIADO' si aplica.
WITH fechas AS (
  SELECT DATE('2025-04-05') AS fecha

```

SQL:

```

-- Esta consulta busca departamentos con capacidad >= 5
-- que estén disponibles el 18 y 19 de abril (2 noches),
-- y calcula el precio por noche considerando si esas fechas caen en feriado.
-- Se prioriza tipo_temporada 'FERIADO' si aplica.
WITH fechas AS (
  SELECT DATE('2025-04-18') AS fecha

```

```

  UNION ALL
  SELECT DATE('2025-04-19')
),

```

temporada_por_fecha AS (

```

  SELECT
    f.fecha,
    (
      SELECT tipo_temporada
      FROM temporadas t
      WHERE f.fecha BETWEEN t.fecha_inicio AND t.fecha_fin
      ORDER BY es_feriado DESC
      LIMIT 1
    ) AS tipo_temporada
  FROM fechas f
),

```

detalle_dias AS (

```

  SELECT
    f.fecha,
    CASE strftime('%w', f.fecha)
      WHEN '0' THEN 'DOMINGO'
      WHEN '1' THEN 'LUNES'
      WHEN '2' THEN 'MARTES'
      WHEN '3' THEN 'MIÉRCOLES'
      WHEN '4' THEN 'JUEVES'
      WHEN '5' THEN 'VIERNES'
      WHEN '6' THEN 'SÁBADO'
    END AS dia_semana

```

Anexo 12: Ejemplos de funcionamiento y respuestas de modelo Few-shot.

3 rows from precios table:

	tipo_dormitorios	tipo_temporada	dia_semana	precio
1	BAJA	LUNES	70.0	
1	BAJA	MARTES	70.0	
1	BAJA	MIÉRCOLES	70.0	

*/

```

Invoking: `sql_db_query` with `{'query': "WITH fechas AS (SELECT DATE('2025-04-28') AS fecha UNION ALL SELECT DATE('2025-04-29') UNION ALL SELECT DATE('2025-04-30')), temporada_por_fecha AS (SELECT f.fecha, (SELECT tipo_temporada FROM temporadas t WHERE f.fecha BETWEEN t.fecha_inicio AND t.fecha_fin ORDER BY es_feriado DESC LIMIT 1) AS tipo_temporada FROM fechas f), detalle_dias AS (SELECT f.fecha, CASE strftime('%w', f.fecha) WHEN '0' THEN 'DOMINGO' WHEN '1' THEN 'LUNES' WHEN '2' THEN 'MARTES' WHEN '3' THEN 'MIÉRCOLES' WHEN '4' THEN 'JUEVES' WHEN '5' THEN 'VIERNES' WHEN '6' THEN 'SÁBADO' END AS dia_semana FROM fechas f), fechas_completas AS (SELECT tf.fecha, tf.tipo_temporada, dd.dia_semana FROM temporada_por_fecha tf JOIN detalle_dias dd ON t.fecha = dd.fecha), precio_por_fecha AS (SELECT pr.id AS id_propiedad, pr.nombre, pr.capacidad, f.fecha, f.tipo_temporada, f.dia_semana, p.precio FROM propiedades pr JOIN disponibilidad d ON d.id_propiedad = pr.id AND d.fecha = f.fecha AND d.reservado = 0 JOIN fechas_completas f ON 1=1 JOIN precios p ON p.tipo_dormitorios = pr.tipo_dormitorios AND p.tipo_temporada = f.tipo_temporada AND p.dia_semana = f.dia_semana WHERE pr.capacidad >= 4) SELECT id_propiedad, nombre, capacidad, SUM(precio) AS total, GROUP_CONCAT(precio || ' (' || fecha || ' - ' || dia_semana || ' - ' || tipo_temporada || ')', ', ') AS detalle FROM precio_por_fecha GROUP BY id_propiedad, nombre, capacidad HAVING COUNT(*) = 3;"}

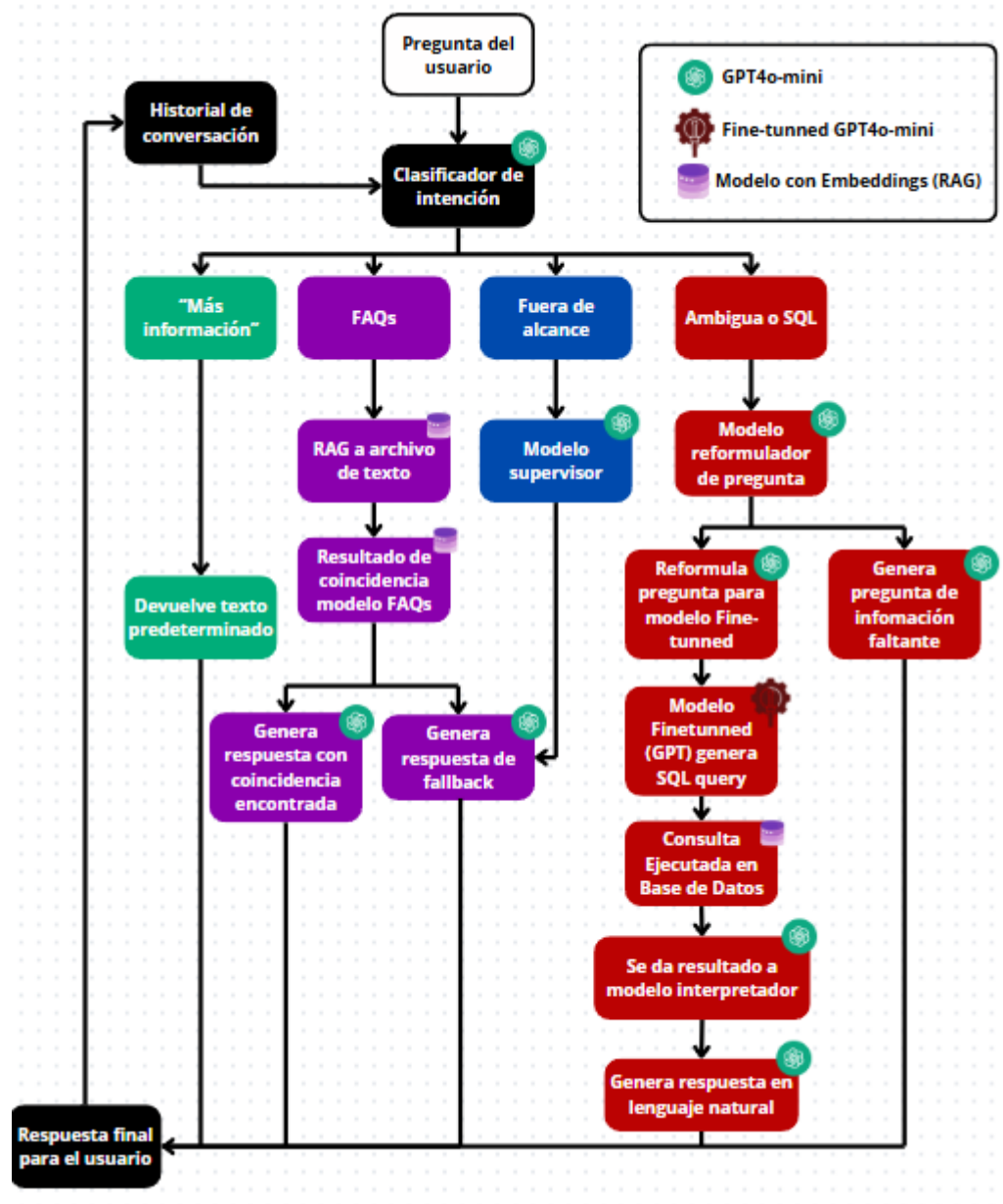
```

Para las fechas del 28 al 30 de abril de 2025, hay disponible un departamento que se ajusta a tu solicitud de 4 personas:

- **Departamento 1702B**:
- **Capacidad**: 4 personas
- **Precio total por 3 noches**: **\$210**
- \$70 (lunes)
- \$70 (martes)
- \$70 (miércoles)

Si deseas proceder con la reserva o si tienes más preguntas, ¡házmelo saber! 😊

Anexo 13: Flujo de modelo Fine-tuned.



Anexo 14: Fracción de código de modelo Fine-tuned.

```

from langchain_openai import ChatOpenAI
from langchain_core.prompts import ChatPromptTemplate
from langchain.sql_database import SQLiteDatabase
from datetime import datetime
from langchain.memory import ConversationBufferMemory

# MODELOS
llm_sql = ChatOpenAI(model="ft:gpt-4o-mini-2024-07-18:personal:ftmodel25:BHHDNheq", temperature=0)
llm_natural = ChatOpenAI(model="gpt-4o-mini", temperature=0.3)

# BASE DE DATOS
db = SQLiteDatabase.from_uri("sqlite:///C:/Users/mateb/Documents/USFQ/Segundo 2024/TESIS/Base de Datos/TestingDB.db")

# FUNCIÓN PRINCIPAL
def responder_con_sql(pregunta: str) -> str:
    try:
        # 1. Generar la consulta SQL desde la pregunta
        prompt_sql = ChatPromptTemplate.from_messages([
            ("system", ""Eres un experto en SQL. Cuando el usuario pregunte por disponibilidad o precios, genera exclusivamente una consulta SQL usando """),
            ("user", "{pregunta}")
        ])

        # Eres un asistente experto en reservas vacacionales. Cuando el usuario pregunte por fechas, precios o disponibilidad, debes generar una consulta SQL usando las tablas disponibles son:
        #- disponibilidad(id_propiedad, fecha, reservado)
        #- propiedades(id, nombre, tipo_dormitorios, capacidad)
        #- precios(tipo_dormitorios, tipo_temporada, dia_semana, precio)
        #- temporadas(id_nombre_fecha_inicio_fecha_fin_es_feriado_tipo_temporada). SOLO USA ESQ PARA HACER LA CONSULTA

    except Exception as e:
        return f"Error: {e}"

# 1. Cargar documentos de FAQs
loader = TextLoader("C:/Users/mateb/Documents/USFQ/Segundo 2024/TESIS/Modelo/FAQsRPA.txt")
docs = loader.load()

# 2. Partir en chunks
text_splitter = CharacterTextSplitter(chunk_size=300, separator="\n")
chunks = text_splitter.split_documents(docs)

# 3. Crear el vectorstore con embeddings
vector_store = FAISS.from_documents(chunks, OpenAIEmbeddings())

# 4. LLM para responder preguntas
llm_faq = ChatOpenAI(model="gpt-4o-mini", temperature=0.1)

from langchain.chains import RetrievalQA

# Crear el QA chain (una sola vez)
qa_chain = RetrievalQA.from_chain_type(
    llm=llm_faq,
    retriever=vector_store.as_retriever(search_kwargs={"k": 2}),
    return_source_documents=False
)

# Función que usa el RAG
def responder_con_faq(pregunta: str) -> str:
    try:
        respuesta = qa_chain.invoke(pregunta)["result"]
        return respuesta.strip()
    except Exception as e:
        return f"Error: {e}"

```

```

from datetime import datetime
from langchain_openai import ChatOpenAI
from langchain_core.prompts import ChatPromptTemplate

llm_reformulador = ChatOpenAI(model="gpt-4o-mini", temperature=0.2)

def formatear_pregunta_para_sql(pregunta_original: str, memoria) -> str:
    hoy = datetime.today().strftime("%A %d de %B de %Y") # Ej: "Miércoles 02 de abril de 2025"

    fechas_memoria = memoria.fechas or "(no se especificaron fechas)"
    personas_memoria = f"{memoria.personas} personas" if memoria.personas else "(sin número de personas)"

    prompt = ChatPromptTemplate.from_messages([
        ("system", f"""
Eres un asistente que debe reformular una pregunta sobre reservas de departamentos vacacionales.

Tu objetivo es crear una nueva pregunta, clara y completa, combinando lo que el usuario dijo originalmente con los datos adicionales disponibles.

Reglas:
- Hoy es {hoy}.
- Si el usuario mencionó cosas como "el sábado", "el 24", o "el fin de semana", conviértelo en una fecha completa (YYYY-MM-DD), asumiendo que se refiere a hoy.
- Usa el número de personas y fechas de la memoria si están disponibles.
- Si los datos de la memoria y la pregunta original ya incluyen todo, simplemente reformula todo en una oración bien redactada.
- Si la pregunta original no incluye el número de personas o las fechas, entonces genera una pregunta de seguimiento amable pidiendo esa información. Recuerda no inventar nada si no tienes datos. Pide lo que falta.

Resultado esperado: una pregunta clara y natural que pueda ser entendida por un modelo entrenado para generar consultas SQL basadas en texto.
"""),
        ("user", f"""
Pregunta original del usuario:
{pregunta_original}

Información adicional del contexto:
{memoria.fechas}
{memoria.personas}
""")
    ])

def responder_chatbot(mensaje_usuario, historial):
    historial.append({"role": "user", "content": mensaje_usuario})

    # Primero: clasificar la intención original (para capturar FAQs)
    intencion_inicial = clasificar_intencion(mensaje_usuario)
    #print("🤖 Clasificador (original):", intencion_inicial)

    if intencion_inicial == "faq":
        respuesta = responder_con_faq(mensaje_usuario)

    elif intencion_inicial == "fuera_de_alcance":
        respuesta = responder_con_modelo_supervisor(mensaje_usuario, intencion_inicial, historial)

    else:
        # Si es SQL o ambigua, reformular y volver a clasificar con contexto
        pregunta_final = reformular_con_historial(historial)
        #print("💡 Pregunta reformulada:", pregunta_final)

        intencion_final = clasificar_intencion(pregunta_final)
        #print("🤖 Clasificador (reformulada):", intencion_final)

        if intencion_final == "sql":
            respuesta = responder_con_sql(pregunta_final)
        else:
            respuesta = pregunta_final # Sigue siendo ambigua

    historial.append({"role": "assistant", "content": respuesta})
    return respuesta

```

Anexo 15: Ejemplos de funcionamiento y respuestas de modelo Fine-tuned.

```
historial = []

msg2 = "¿Cuánto cuesta un depar para 8 del 9 hasta el 15 de agosto de 2025"

print("Usuario:", msg2)
print("Asistente:", responder_chatbot(msg2, historial))
```

Usuario: ¿Cuánto cuesta un depar para 8 del 9 hasta el 15 de agosto de 2025

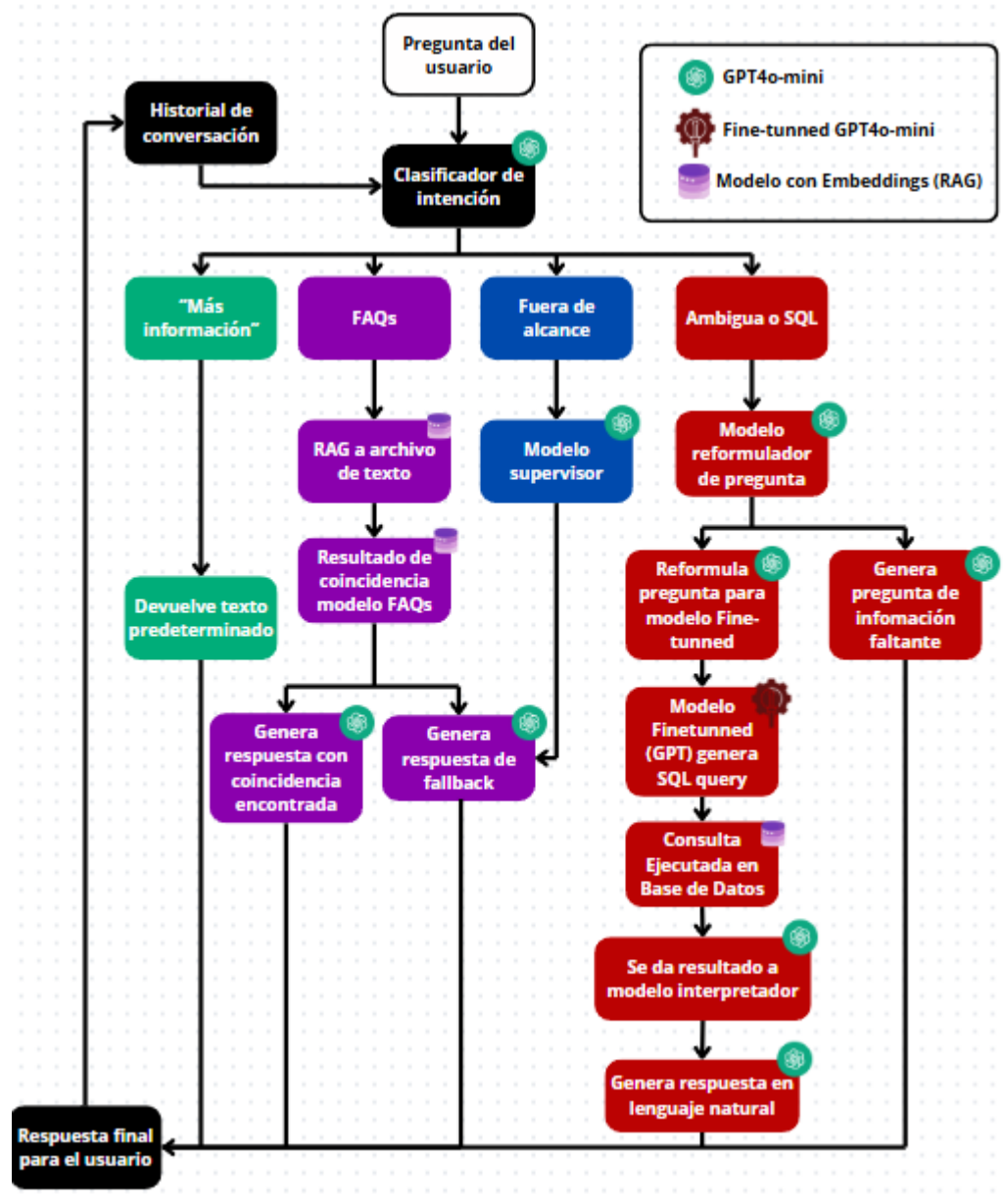
```
```sql
WITH fechas AS (
 SELECT DATE('2025-08-09') AS fecha
 UNION ALL
 SELECT DATE('2025-08-10')
 UNION ALL
 SELECT DATE('2025-08-11')
 UNION ALL
 SELECT DATE('2025-08-12')
 UNION ALL
 SELECT DATE('2025-08-13')
 UNION ALL
 SELECT DATE('2025-08-14')
 UNION ALL
 SELECT DATE('2025-08-15')
)
SELECT
 id_propiedad,
 nombre,
 capacidad,
 SUM(precio) AS total,
 GROUP_CONCAT(precio || ' (' || fecha || ' - ' || dia_semana || ' - ' || tipo_temporada || ')', ', ') AS detalle
FROM precio_por_fecha
GROUP BY id_propiedad, nombre, capacidad
HAVING COUNT(*) = 7;
```
```

Asistente: Claro, aquí tienes las opciones disponibles para tus fechas del 9 al 15 de agosto de 2025:

- Departamento 1703B (capacidad: 8 personas): \$1,520.00 total
 Detalle: \$240.00 el sábado 9 de agosto (FERIADO), \$240.00 el domingo 10 de agosto (FERIADO), \$200.00 el lunes 11 de agosto (ALTA), \$200.00 el martes 12 de agosto (ALTA), \$200.00 el miércoles 13 de agosto (ALTA), \$220.00 el jueves 14 de agosto (ALTA), \$220.00 el viernes 15 de agosto (ALTA)
- Departamento 1203A (capacidad: 8 personas): \$1,520.00 total
 Detalle: \$240.00 el sábado 9 de agosto (FERIADO), \$240.00 el domingo 10 de agosto (FERIADO), \$200.00 el lunes 11 de agosto (ALTA), \$200.00 el martes 12 de agosto (ALTA), \$200.00 el miércoles 13 de agosto (ALTA), \$220.00 el jueves 14 de agosto (ALTA), \$220.00 el viernes 15 de agosto (ALTA)
- Departamento 203A (capacidad: 8 personas): \$1,520.00 total
 Detalle: \$240.00 el sábado 9 de agosto (FERIADO), \$240.00 el domingo 10 de agosto (FERIADO), \$200.00 el lunes 11 de agosto (ALTA), \$200.00 el martes 12 de agosto (ALTA), \$200.00 el miércoles 13 de agosto (ALTA), \$220.00 el jueves 14 de agosto (ALTA), \$220.00 el viernes 15 de agosto (ALTA)

Si necesitas más información o deseas realizar una reserva, no dudes en decírmelo. ¡Estoy aquí para ayudarte!

Anexo 16: Flujo de modelo combinado. (Mismo que el Fine-tuned)



Anexo 17: Fracción de código de modelo combinado. (Mismo código del modelo Fine-tuned con ejemplos Few-shot en el prompt)

Cuando el usuario diga "desde [fecha] hasta [fecha]", interpreta que la fecha final es la de salida, por lo tanto ****no se paga**** esa noche. Solo calcula Ejemplo:

"Desde el 6 hasta el 9 de abril" significa:

- Se pagan las noches del 6, 7 y 8 de abril
- No se paga el 9

Nunca incluyas la fecha final del rango cuando construyas la tabla de fechas.

EJEMPLOS: EJEMPLO #1

Usuario: Quiero reservar para 6 personas del 18 al 21 de abril

SQL:

```
-- Esta consulta busca propiedades con capacidad >= 6
-- disponibles las noches del 18, 19 y 20 de abril
-- Aplica tipo_temporada FERIADO si corresponde, y descarta cualquier propiedad
-- que no esté disponible en al menos una de las fechas
-- Se excluyen propiedades que no están libres en las 3 fechas
```

```
WITH fechas AS (
  SELECT DATE('2025-04-18') AS fecha
  UNION ALL SELECT DATE('2025-04-19')
  UNION ALL SELECT DATE('2025-04-20')
```

```
),
```

```
temporada_por_fecha AS (
```

```
  SELECT
```

```
    f.fecha,
```

```
    (
```

```
      SELECT tipo_temporada
```

```
        WHEN '0' THEN 'DOMINGO'
```

```
        WHEN '1' THEN 'LUNES'
```

```
        WHEN '2' THEN 'MARTES'
```

```
        WHEN '3' THEN 'MIERCOLES'
```

```
        WHEN '4' THEN 'JUEVES'
```

```
        WHEN '5' THEN 'VIERNES'
```

```
        WHEN '6' THEN 'SABADO'
```

```
      END AS dia_semana
```

```
    FROM fechas f
```

```
  ),
```

```
fechas_completas AS (
```

```
  SELECT
```

```
    tf.fecha,
```

```
    tf.tipo_temporada,
```

```
    dd.dia_semana
```

```
  FROM temporada_por_fecha tf
```

```
  JOIN detalle_dias dd ON tf.fecha = dd.fecha
```

```
),
```

```
precio_por_fecha AS (
```

```
  SELECT
```

```
    pr.id AS id_propiedad,
```

```
    pr.nombre,
```

```
    pr.capacidad,
```

```
    f.fecha,
```

```
    f.tipo_temporada,
```

```
    f.dia_semana,
```

```
    p.precio
```

```
  FROM propiedades pr
```

```
  JOIN disponibilidad d ON d.id_propiedad = pr.id AND d.fecha = f.fecha AND d.reservado = 0
```

```
  JOIN fechas_completas f ON 1=1
```

```
  JOIN precios p ON p.tipo_dormitorios = pr.tipo_dormitorios
```

```
    AND p.tipo_temporada = f.tipo_temporada
```

```
    AND p.dia_semana = f.dia_semana
```

```

        AND p.tipo_temporada = t.tipo_temporada
        AND p.dia_semana = f.dia_semana
    )
    WHERE pr.capacidad >= 6
)
SELECT
    id_propiedad,
    nombre,
    capacidad,
    SUM(precio) AS total,
    GROUP_CONCAT(precio || ' (' || fecha || ' - ' || dia_semana || ' - ' || tipo_temporada || ')', ', ') AS detalle
FROM precio_por_fecha
GROUP BY id_propiedad, nombre, capacidad
HAVING COUNT(*) = 3
ORDER BY total ASC
LIMIT 2;

EJEMPLO #2
Usuario: Quiero reservar un departamento para 3 personas, del 3 al 5 de mayo de 2025.

SQL:
-- Esta consulta busca departamentos con capacidad >= 3
-- que estén disponibles el 3 y 4 de mayo (2 noches),
-- y calcula el precio por noche considerando si esas fechas caen en feriado.
-- Se prioriza tipo_temporada 'FERIADO' si aplica.
WITH fechas AS (
    SELECT DATE('2025-05-03') AS fecha
    UNION ALL
    SELECT DATE('2025-05-04')
),
temporada_por_fecha AS (
    SELECT
        f.fecha,
        (
            SELECT tipo_temporada
            FROM temporadas t
            WHERE f.fecha BETWEEN t.fecha_inicio AND t.fecha_fin
            ORDER BY es_feriado DESC
            LIMIT 1
        ) AS tipo_temporada
    FROM fechas f
),
detalle_dias AS (
    SELECT
        f.fecha,
        CASE strftime('%w', f.fecha)
        WHEN '0' THEN 'DOMINGO'
        WHEN '1' THEN 'LUNES'
        WHEN '2' THEN 'MARTES'
        WHEN '3' THEN 'MIÉRCOLES'
        WHEN '4' THEN 'JUEVES'
        WHEN '5' THEN 'VIERNES'
        WHEN '6' THEN 'SÁBADO'
        END AS dia_semana
    FROM fechas f
),
propiedades AS (
    SELECT
        id_propiedad,
        nombre,
        capacidad,
        precio,
        tipo_temporada,
        dia_semana
    FROM precio_por_fecha
    ORDER BY precio
    LIMIT 2;

```

Anexo 18: Ejemplos de funcionamiento y respuestas de modelo omdinado.

```
print(responder_con_sql("Cuanto cuesta para 6 personas desde el 9 hasta el 10 de agosto de 2025"))
```

```

-- Esta consulta busca propiedades con capacidad >= 6
-- disponibles la noche del 9 de agosto de 2025,
-- y obtiene el precio de esa noche considerando tipo_temporada (priorizando feriado si aplica).

```

```

WITH fechas AS (
    SELECT DATE('2025-08-09') AS fecha
),
temporada_por_fecha AS (
    SELECT
        f.fecha,
        (
            SELECT tipo_temporada
            FROM temporadas t
            WHERE f.fecha BETWEEN t.fecha_inicio AND t.fecha_fin
            ORDER BY es_feriado DESC
            LIMIT 1
        ) AS tipo_temporada
    FROM fechas f
),
detalle_dias AS (
    SELECT
        f.fecha,
        CASE strftime('%w', f.fecha)
        WHEN '0' THEN 'DOMINGO'
        WHEN '1' THEN 'LUNES'
        WHEN '2' THEN 'MARTES'
        WHEN '3' THEN 'MIÉRCOLES'
        WHEN '4' THEN 'JUEVES'
        WHEN '5' THEN 'VIERNES'
        WHEN '6' THEN 'SÁBADO'
        END AS dia_semana
    FROM fechas f
),
propiedades AS (
    SELECT
        id_propiedad,
        nombre,
        capacidad,
        precio,
        tipo_temporada,
        dia_semana
    FROM precio_por_fecha
    ORDER BY precio
    LIMIT 2;

```

```

SELECT
    id_propiedad,
    nombre,
    capacidad,
    precio,
    tipo_temporada,
    dia_semana
FROM precio_por_fecha
ORDER BY precio
LIMIT 2;

```

Claro, aquí tienes las opciones disponibles para tus fechas del 9 al 10 de agosto de 2025:

```
- Departamento 1701B (capacidad: 6 personas): $180.00 total
  Detalle: FERIA DO el SÁBADO
```

```
- Departamento 201B (capacidad: 6 personas): $180.00 total
  Detalle: FERIA DO el SÁBADO
```

Si necesitas más información o deseas realizar una reserva, no dudes en decírmelo. ¡Estoy aquí para ayudarte!

```

from langchain.document_loaders import TextLoader
from langchain.vectorstores import FAISS
from langchain.embeddings import OpenAIEmbeddings
from langchain.chains import RetrievalQA
from langchain_openai import ChatOpenAI

# Cargar documento
loader = TextLoader(r"C:\Users\mateb\Documents\USFQ\Segundo 2024\TESIS\Modelo\FAQsRPA.txt")
docs = loader.load()

# Crear chunks y vector store
from langchain.text_splitter import CharacterTextSplitter
splitter = CharacterTextSplitter(chunk_size=500, chunk_overlap=50)
chunks = splitter.split_documents(docs)

# Crear embeddings y base de búsqueda
vectorstore = FAISS.from_documents(chunks, OpenAIEmbeddings())

# Crear cadena de recuperación tipo QA
faq_chain = RetrievalQA.from_chain_type(
    llm=ChatOpenAI(model="gpt-4o-mini"),
    retriever=vectorstore.as_retriever()
)

```

Anexo 19: Ejemplos de preguntas de cada categoría.

| pregunta_usuario | tipo_consulta |
|---|---------------|
| ¿Cuánto cuesta un departamento para 4? Del 28 al 30 | sql |
| ¿Cuánto cuesta un depar para 7 del 1 hasta el 6? | sql |
| Entre semana cuanto cuesta? | ambigua |
| ¿Cuánto cuesta un departamento para 4? | ambigua |
| Cuál es la política de mascotas? | faq |
| Qué incluye el precio? | faq |
| ¿Cuál es la política de cancelación por mal clima? | fuera_alcance |
| ¿El jacuzzi está disponible 24/7? | fuera_alcance |

Anexo 20: Archivo de texto guardado para preguntas frecuentes.

FAQ

Q: ¿Cuál es el horario de check-in?:

A: El check-in es a partir de la 1:00 pm. Se puede ingresar hasta máximo las 11:00 pm.

Q: ¿Cuál es el horario de check-out?:

A: El check-out es hasta las 11:00 am

Q: ¿Cuál es la política de mascotas?

A: Se permiten mascotas pequeñas o medianas con un costo adicional de \$20 USD por noche, más una garantía de daños reembolsable si todo está en orden.

Q: ¿Qué incluye el precio de la reserva?

A: El uso del departamento totalmente equipado, brazaletes para acceder a todas las instalaciones del condominio, parqueadero, e incluye limpieza final. No incluye alimentación.

Q: ¿Qué incluyen los departamentos?

A: Todos los departamentos cuentan con cocina totalmente equipada, sábanas, toallas (para uso dentro del departamento), balcón con vista al mar, internet y DIRECTV.

Q: ¿Las duchas tienen agua caliente?

A: Si, todas las duchas cuentan con agua caliente.

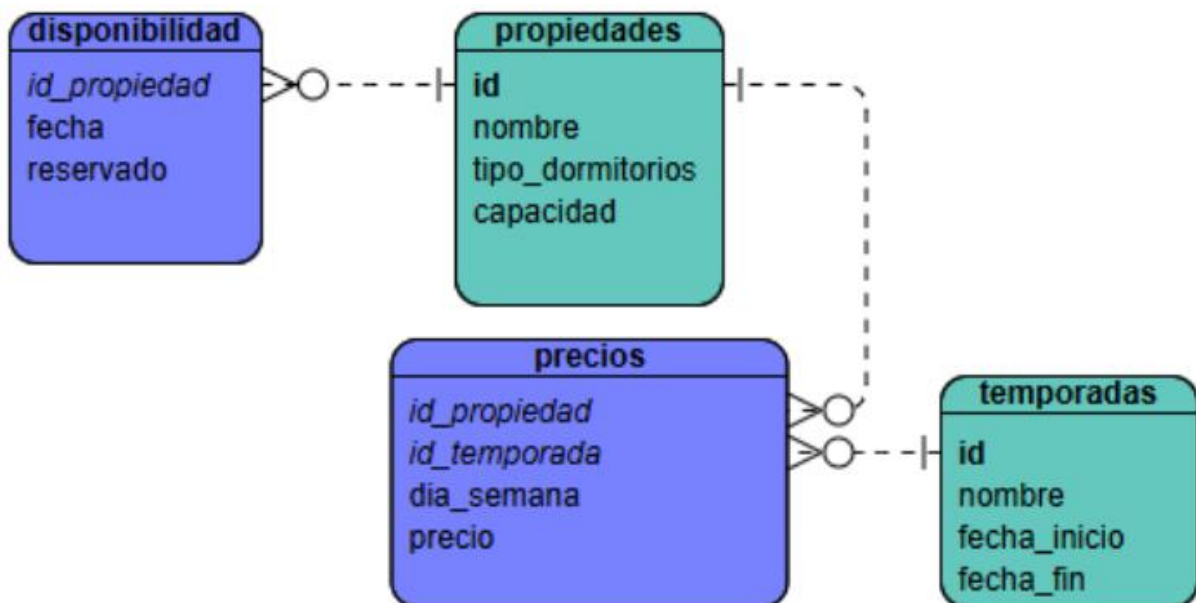
Q: ¿Cómo es la seguridad en el condominio?

A: El condominio tiene seguridad 24/7. Es un lugar muy seguro.

Q: ¿Incluye desayuno u otros alimentos?

A: No. La reserva no incluye alimentos. Sin embargo, dentro del condominio hay un restaurante y un bar.

Anexo 21: ERD de base de datos de SQL.



Anexo 22: Diagrama de resultados prueba de Friedman en evaluación general.