Universidad Zaragoza
1542

Trabajo Fin de Grado

Automatización de la Gestión de Incidencias en Servicios de Telecomunicaciones con Modelos de Lenguaje

Automation of Incident Management in Telecommunications Services with Language Models

Autor

Daniel Luna Goez

Directores

Dayana Ribas González

Jorge Carlos Gracia del Río

ESCUELA DE INGENIERÍA Y ARQUITECTURA

2025

RESUMEN

En el sector de las telecomunicaciones, la rapidez en la resolución de incidencias es un factor decisivo para mantener la calidad del servicio y asegurar la satisfacción de los clientes. Sin embargo, el volumen creciente de tickets gestionados manualmente se ha convertido en un reto significativo para los equipos de soporte. En este contexto, surge la necesidad de aplicar tecnologías que automaticen la gestión de incidencias, reduciendo tiempos de respuesta y asegurando consistencia en las soluciones.

BTS es una operadora internacional de telecomunicaciones que gestiona un elevado volumen de tráfico de voz y datos y se sitúa entre las compañías relevantes del sector. El departamento de *Customer Service* (CS) atiende diariamente incidencias asociadas a problemas recurrentes de telefonía IP, como baja tasa de conexión (ASR), duración anómala de llamadas (ALOC), bucles, rechazos de llamadas y otras degradaciones técnicas.

Este Trabajo Fin de Grado se desarrolla en colaboración con BTS y aborda la creación de un sistema capaz de automatizar el análisis y la resolución de estas incidencias. La solución integra modelos de lenguaje entrenados con datos reales de la empresa para clasificar automáticamente el tipo de degradación y generar propuestas de respuesta alineadas con el flujo de trabajo del equipo de CS.

La solución no se limita a la clasificación textual, sino que también incorpora el análisis de métricas técnicas de red y registros de llamadas, emulando el flujo de trabajo actual del equipo de CS. Esto permite al sistema actuar con datos reales.

El proyecto se ha desarrollado en fases progresivas: desde el estudio y definición del flujo de resolución de incidencias, la preparación de un dataset representativo, el entrenamiento y validación del modelo, hasta su integración con herramientas corporativas de gestión de incidencias como Jira y bases de datos internas. A través de pruebas controladas, el sistema ha demostrado su capacidad para reducir la carga manual del equipo y mejorar los tiempos de respuesta, aportando valor directo a la compañía.

El objetivo general es evaluar el impacto de la automatización en la gestión de tickets, analizando beneficios en eficiencia, escalabilidad y calidad del servicio. Este trabajo abre la puerta a un futuro en el que los sistemas de soporte en telecomunicaciones sean más ágiles, inteligentes y adaptados a las demandas de un mercado global altamente competitivo.

Abstract

In the telecommunications sector, the speed at which incidents are resolved is a decisive factor for maintaining service quality and ensuring customer satisfaction. However, the growing volume of tickets handled manually has become a significant challenge for support teams. In this context, there is a clear need to apply advanced artificial intelligence technologies to optimize incident management, reduce response times, and guarantee consistency in the solutions provided.

BTS is an international telecommunications operator that manages a high volume of voice and data traffic and is positioned among the key companies in the sector. The Customer Service (CS) department handles daily incidents associated with recurring IP telephony issues such as low connection rates (ASR), abnormal call duration (ALOC), call loops, call rejections, and other technical degradations.

This Bachelor's Thesis is developed in collaboration with BTS and addresses the creation of a system capable of automating the analysis and resolution of these incidents. The solution integrates language models trained with real company data to automatically classify the type of degradation and generate response proposals aligned with the CS team's workflow.

The solution goes beyond textual classification, incorporating the analysis of technical network metrics and call records, emulating the current workflow followed by the CS team. This enables the system to act based on real operational data.

The project has been developed through progressive phases: from studying and defining the incident-resolution workflow, preparing a representative dataset, training and validating the model, to integrating it with corporate tools such as Jira and internal databases. Through controlled testing, the system has demonstrated its ability to reduce the manual workload of the team and improve response times, providing direct value to the company.

The overall objective is to evaluate the impact of automation on ticket management, analyzing improvements in efficiency, scalability, and service quality. This work opens the door to a future in which support systems in telecommunications are more agile, intelligent, and aligned with the demands of a highly competitive global market, positioning BTS as a key player in the adoption of AI-based solutions for customer service.

Índice

Capítulo 1

Introducción y objetivos

1.1. Introducción

La fiabilidad de los servicios de telecomunicaciones se sostiene, en gran medida, en la capacidad de detectar y resolver incidencias con rapidez. En organizaciones que operan a escala global, como Business Telecommunications Services (BTS)[1], la atención al cliente opera de forma ininterrumpida, veinticuatro horas al día, prestando soporte a clientes de los cinco continentes.

El departamento de Customer Service constituye la primera línea de soporte tanto para las incidencias reportadas por los clientes como para aquellas generadas internamente por los distintos equipos técnicos. Su estructura operativa se organiza en varios subgrupos especializados, cada uno encargado de gestionar los tickets asociados al ámbito técnico que cubren ——voz, mensajería, numeración——. Esta segmentación permite mantener una asignación precisa de responsabilidades y garantizar que cada incidente sea evaluado por personal con conocimiento específico del sistema afectado.

Dado que BTS opera a escala internacional el equipo se distribuye en tres turnos de guardia que garantizan atención sin interrupciones durante las 24 horas del día. Los turnos nocturnos y de fin de semana están atendidos por un grupo reducido cuya función es mitigar las incidencias más críticas.

Este departamento de CS gestiona a diario más de 2.000 tickets en Jira recibidos por correo electrónico — 800 específicos de voz—, los cuales describen problemas distintos como descensos en la tasa de conexión (ASR), incrementos anómalos en la duración media de las llamadas (ALOC), bucles en la señalización, entre otros. La totalidad de este volumen se procesa manualmente a través de Jira, siguiendo un flujo operativo bien estructurado pero altamente demandante. Esta carga termina afectando a los tiempos de respuesta, incrementa el riesgo de errores humanos, provoca escalados innecesarios al equipo técnico e impide que el equipo de CS —responsable de la primera línea de

[1] `https://bts.io/`

soporte— pueda dedicar tiempo a tareas de mayor complejidad y valor añadido.

Desde la perspectiva de las operaciones de red, el proceso de *Incident Management* inicia cuando un cliente reporta una anomalía. El ticket es entonces asignado a un ingeniero de guardia, que debe sumergirse en el problema, mitigarlo y, si es posible, encontrar su causa raíz para aplicar una solución definitiva. La heterogeneidad de los sistemas implicados, el volumen de datos a analizar y la necesidad de correlacionar métricas de distinta naturaleza convierten la investigación en un proceso complejo. Las plataformas tradicionales de gestión de incidencias no siempre ayudan, a veces se quedan cortas por exceso de información o simples barreras entre equipos, lo que deriva en tiempos de resolución más altos y diagnósticos inconsistentes.

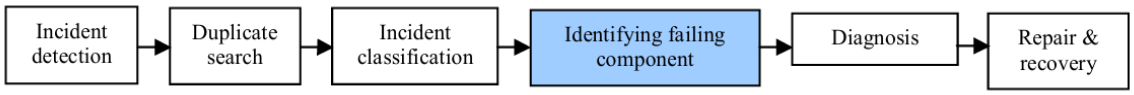

Figura 1.1: Flujo de gestión de incidencias [1]

En los últimos años, los avances en modelos de lenguaje de gran tamaño (LLM) han abierto una ventana de posibilidades para aliviar estas limitaciones. Estos modelos son capaces de ingerir datos masivos, extraer de ellos información estructurada e incluso automatizar tareas repetitivas. Integrarlos en las herramientas de soporte permite crear asistentes que no solo detectan patrones entre incidentes, sino que también pueden formular hipótesis y sugerir pasos de mitigación casi en tiempo real.

En este contexto surge *Hefestus*, la denominación elegida para el sistema de automatización de incidencias desarrollado en el marco de este proyecto. Su nombre sigue la convención de nomenclatura interna de la empresa y se emplea a lo largo de esta memoria para referirse al conjunto de módulos y procesos que conforman la solución propuesta. Esta denominación se explica con más detalle en el Anexo C.

1.2. Motivación y objetivos

El proyecto nace como respuesta a un problema muy concreto: la carga de trabajo del equipo de CS de BTS es cada vez más desbordante. Gestionar manualmente más de 2000 incidencias al día no solo implica un esfuerzo enorme, sino que obliga a los agentes a enfrentarse una y otra vez a tareas repetitivas de triaje, revisión de registros y comunicación con el cliente. Además, cada ticket llega acompañado de correos con estructuras particulares y referencias a logs de llamadas. Para resolverlo hay que consultar bases de datos internas con información detallada sobre las conexiones. Todo este recorrido provoca tiempos de respuesta altos, decisiones poco coherentes

entre agentes y frecuentemente escalados manuales que podrían evitarse. Y es que la necesidad de reducir la *mean-time-to-resolution* y liberar al equipo para centrarse en incidencias más complejas es la principal motivación de este proyecto.

La llegada de los modelos de lenguaje de gran tamaño ha supuesto un verdadero cambio de paradigma en el análisis de incidencias. Estos sistemas son capaces de procesar volúmenes inmensos de texto no estructurado, detectar patrones que pasan desapercibidos a simple vista y ofrecer contexto adicional que ayuda a entender qué está ocurriendo realmente.

Los objetivos de este trabajo se estructuran en dos niveles. El **objetivo general** consiste en diseñar e implementar un sistema basado en modelos de lenguaje que automatice la gestión de incidencias en servicios de telecomunicaciones. Para alcanzar este fin se plantean varios **objetivos específicos**:

O1. **Analizar el flujo actual de resolución de incidencias** y describirlo con detalle, identificando los puntos críticos en los que se pierde tiempo o se cometen errores.

O2. **Preparar un conjunto de datos representativo**, recopilando tickets históricos y depurando los textos de Jira para eliminar formatos indeseados y ruido, de modo que sean aptos para el entrenamiento del modelo.

O3. **Desarrollar y adaptar modelos de lenguaje al contexto de BTS**, abordando tanto la clasificación de tickets como la extracción de información estructurada (p. ej. números ANI/DNI, fechas y causas) y evaluando configuraciones como BERT y Llama.

O4. **Integrar el modelo con Jira, correo electrónico y bases de datos internas**, desarrollando un pipeline que lea incidencias recientes, aplique reglas heurísticas de fallback y actualice campos específicos en la plataforma.

O5. **Generar respuestas automáticas**, evaluando la capacidad del sistema para extraer información relevante para el análisis de la incidencia y sus respuesta.

O6. **Medir y optimizar la eficacia del sistema**, comparando la reducción de tiempos de respuesta y la precisión de las predicciones frente al proceso manual e investigando las limitaciones.

La hipótesis de partida es que un agente basado en grandes modelos de lenguaje puede automatizar una parte significativa de los tickets y, al mismo tiempo, proporcionar al personal de CS herramientas de apoyo que mejoren su productividad.

Sin embargo, se asume que las tareas de decisión final y los escalados a técnicos expertos seguirán siendo responsabilidad de los operadores humanos

1.3. Organización de la memoria

La memoria se estructura del siguiente modo:

- **Capítulo 1**: Se introducen el contexto general del proyecto, las motivaciones, los objetivos planteados y la problemática operativa que da origen a la necesidad de automatizar la gestión de incidencias en BTS.
- **Capítulo 2**: Se desarrolla el marco teórico, donde se describen los conceptos fundamentales de la telefonía IP, los tipos de incidencias, las métricas técnicas, los elementos de procesamiento del lenguaje natural y las tecnologías empleadas. El capítulo incluye además un análisis del estado de la cuestión y de soluciones similares existentes.
- **Capítulo 3**: Se presenta la arquitectura completa del sistema Hefestus, detallando sus componentes, flujos internos, integración con Jira, bases de datos corporativas y modelos lingüísticos como Llama3 empleados.
- **Capítulo 4**: Se describe el proceso de entrenamiento, evaluación y comparación del modelo BERT desarrollado para la clasificación automática de tickets, incluyendo la preparación del dataset, las métricas obtenidas y el análisis de su rendimiento.
- **Capítulo 5**: Se explica el flujo de análisis específico para degradaciones de tipo ASR, incluyendo las consultas a Pinot, la lógica estadística aplicada y el método de detección de anomalías utilizado para evaluar clientes, destinos y proveedores.
- **Capítulo 6**: Se exponen las pruebas realizadas, los resultados experimentales y la evaluación comparativa entre la respuesta manual del equipo de CS y el sistema automatizado. También se analiza la satisfacción de los usuarios y el impacto operativo del sistema.

Finalmente, se incluyen varios anexos que amplían la información técnica del sistema, como ejemplos de tickets, prompts empleados, configuraciones internas y documentación complementaria utilizada durante el desarrollo del proyecto.

Capítulo 2

Marco Teórico

2.1. Conceptos Clave

A continuación se definen los principales conceptos relacionados con el dominio de las telecomunicaciones que serán empleados durante el desarrollo del trabajo.

CDR — Call Detail Record

Se refiere a un tipo de registro de datos generado por un sistema de telecomunicaciones para cada llamada realizada o recibida. Este registro contiene información como el número de teléfono del llamante, el número de teléfono del destinatario, la duración de la llamada y cualquier otro dato adicional relacionado con la misma.

Los CDR se utilizan ampliamente por las empresas de telecomunicaciones y los proveedores de servicios para rastrear y gestionar la actividad de las llamadas telefónicas. Pueden emplearse para fines de facturación, monitorización de la calidad de las llamadas, análisis de patrones de tráfico, detección de fraude y otros usos adicionales [2].

Dentro del CDR vamos a explicar algunos campos relevantes usados en el resto del trabajo:

- **ANI (Automatic Number Identification)**: Identifica el número de origen de la llamada y permite determinar qué abonado ha iniciado el establecimiento.

- **DNI (Dialed Number Identification)**: Representa el número marcado por el llamante, es decir, el destino al que se pretende conectar la llamada.

- **seizeTime**: Marca temporal que indica el instante exacto en el que se inicia el intento de establecimiento de la llamada en la red.

– **releaseCause**: Código de liberación que especifica el motivo por el cual la llamada finaliza, ya sea por rechazo, ocupación, fallo de señalización, cancelación por el usuario o terminación normal.

RTP — Real Time Protocol

El protocolo RTP, o protocolo de tiempo real, es una tecnología estandarizada que se utiliza para transmitir datos de audio y vídeo a través de redes. Opera en la capa de transporte del conjunto de protocolos de Internet y funciona junto con protocolos para gestionar eficientemente los paquetes de datos multimedia [3].

2.2. Tipos de incidencias

En el ámbito de la telefonía IP y el enrutamiento de llamadas internacionales [4], los proveedores deben monitorizar continuamente indicadores clave que reflejan el estado de la red y la calidad del servicio. Las incidencias reportadas al equipo de CS de la empresa suelen estar directamente relacionadas con degradaciones detectadas por clientes o socios de interconexión. Estas incidencias presentan patrones recurrentes que permiten clasificarlas en categorías técnicas bien definidas.

A continuación se resumen las degradaciones más relevantes en el contexto del proyecto:

ASR — Answer Seizure Ratio

[5] El ASR mide la proporción de llamadas contestadas respecto al total de intentos. Un ASR bajo suele indicar congestión, errores de señalización, problemas de encaminamiento o rechazo intencionado por parte de un proveedor intermedio. Un ticket típico incluye ejemplos de llamadas con códigos SIP (480, 487, 403...)

ALOC — Average Length of Call

ALOC representa la duración media de las llamadas contestadas. Una caída significativa puede deberse a problemas de audio, desconexiones tempranas, fraude, fallos de media (RTP) o rutas inestables. En BTS, las degradaciones de ALOC implican normalmente analizadores de CDRs internos para confirmar si la anomalía es reproducible.

SPAM

Incluye comunicaciones automáticas, suplantación de identidad del número llamante o patrones de llamadas sospechosas [6].

Rejection

Ocurre cuando una llamada es rechazada por la red antes de que llegue a estado de estilo *ringing* o *answered*. Se detecta porque el campo *releaseCause*, indica un rechazo explícito del lado remoto [7]. Lo que provoca problemas de rutas de operador, de traducción de números, capacidad insuficiente en la pasarela del proveedor, códigos de país mal configurados etc.

2.3. Procesamiento de Lenguaje Natural y Modelos de Lenguaje

NLP — Procesamiento del lenguaje natural

Es una rama de la inteligencia artificial que estudia técnicas para interpretar, estructurar y generar lenguaje humano. En un entorno como BTS, los tickets contienen texto libre, frecuentemente escrito por proveedores de múltiples países, con estructuras semiformales y términos técnicos mezclados con descripciones poco estandarizadas [8].

El objetivo del NLP en este proyecto es:

- Clasificar la incidencia según su tipo técnico.
- Extraer entidades relevantes (ANI, DNI, fechas, duración, país destino).

Modelos de lenguaje LLM

Los LLM son modelos entrenados con grandes volúmenes de texto y capaces de realizar tareas como clasificación, resumen, extracción estructurada o generación natural.

En el proyecto se integran dos tipos de modelos:

- Modelos discriminativos (BERT) para clasificación de tickets [9].
- Modelos generativos (LLama3 vía Ollama) para la extracción estructurada de información compleja [10] [11].

BERT

BERT (*Bidirectional Encoder Representations from Transformers*) es un modelo de arquitectura Transformer [12] entrenado para comprender relaciones semánticas en texto. En Hefestus se utiliza BERT para la clasificación de cada ticket en una de las más de 20 degradaciones posibles.

Se entrenó un modelo específico con datos reales de BTS para mejorar la precisión respecto a BERT genérico.

LLAMA3

LLama3 es un modelo generativo de última generación, desplegado localmente mediante Ollama[1]. En el proyecto se utiliza para la extracción de entidades complejas de tickets ambiguos y desestructurados como llamadas, fechas y destinos, formateo de la información en JSON limpio y resolver inconsistencias del texto como fechas parciales, horas en zonas horarias diferentes.

Su capacidad para seguir instrucciones precisas permite que Hefestus extraiga informacón que sería inviable mediante expresiones regulares.

2.4. Gestión de incidencias

La gestión de incidencias en empresas como BTS es un proceso centralizado en plataformas como Jira[2]. Cada incidencia/ticket incluye un identificador único, resumen, descripción (donde suelen encontrarse las llamadas de ejemplo), organización del cliente y estado.

El flujo normal implica la lectura manual del ticket, identificación del tipo de incidencia, consulta de logs y CDRs en bases de datos internas, verificación de métricas (ASR, ALOC, PDD, rechazo), generación de respuesta al cliente, posible escalado a equipos técnicos o proveedor que da los problemas.

[1] `https://ollama.com/`
[2] `https://www.atlassian.com/software/jira/service-management`

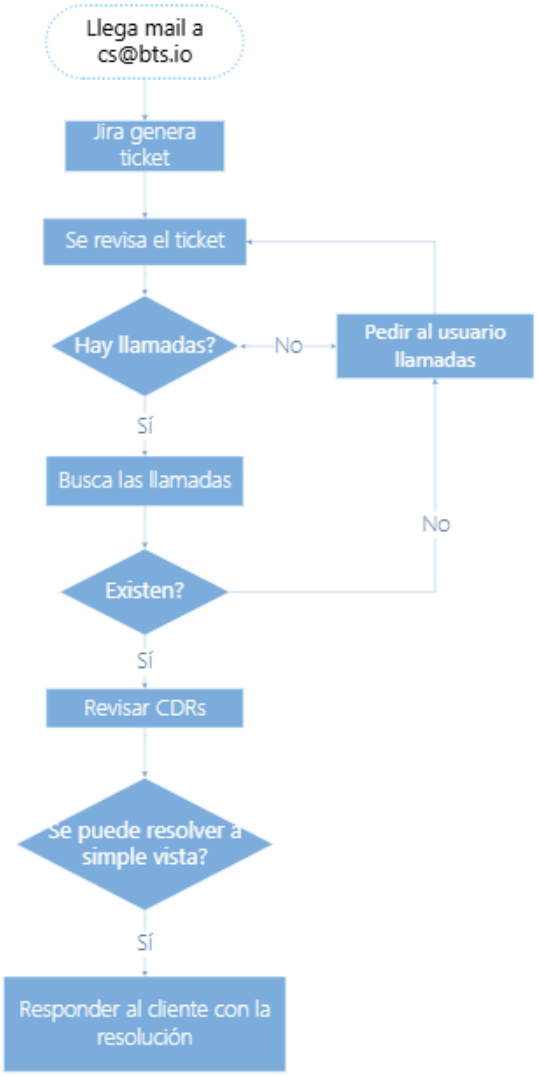

Figura 2.1: Flujo de resolución del departamento de CS

2.5. Métricas

La fiabilidad del sistema depende de medir correctamente el rendimiento de los modelos utilizados [13].

Accuracy

Proporción de predicciones correctas sobre el total de ejemplos. Ofrece una visión global pero puede ser engañosa en datasets desbalanceados.

Precision, Recall y F1-score

- ***Precisión***: mide la proporción de ejemplos predichos como una clase que realmente pertenecen a ella.

- ***Recall***: mide la proporción de ejemplos de una clase que fueron correctamente identificados.

- ***F1-score***: media armónica entre *precision* y *recall*, muy utilizada en clasificación multiclase.

En el proyecto se utiliza especialmente *F1-score ponderador*, que tiene en cuenta el tamaño de cada clase, y el *F1-score macro*, que da el mismo peso a todas las clases sin importar su frecuencia.

Top-k Accuracy

Proporción de veces que la clase verdadera está dentro del *top-k* de clases más probables predichas. En el proyecto se utiliza mayoritariamente el top-5, que sirve para ver que 5 clases tienen mayor probabiilidad de salir candidatas.

2.6. Tecnologías empleadas

Python

Se ha elegido como lenguaje principal del proyecto por su ecosistema maduro, librerías NLP/ML (Transformers[3], Torch[4], Pandas[5]), integración sencilla con APIs y bases de datos y su alto nivel de mantenibilidad.

Transformers (HugginFace)

La librería *Transformers*, desarrollada por HuggingFace, se ha consolidado como el estándar de facto para el uso de modelos basados en la arquitectura Transformer en entornos de procesamiento de lenguaje natural. Proporciona una colección unificada de modelos preentrenados —como BERT, RoBERTa, GPT o LLama— junto con utilidades para su carga, tokenización, ajuste fino y evaluación, lo que permite aplicar técnicas avanzadas de NLP sin necesidad de implementar los modelos desde cero.

[3] `https://huggingface.co/docs/transformers`
[4] `https://pytorch.org/`
[5] `https://pandas.pydata.org/`

Ollama[6]

Entorno local para alojar LLMs ligeros y rápidos, lo que permite integraciones reproducibles y evita depender y tratar información privada de la empresa con servicios externos [11].

PostgreSQL + psycopg2

Usado para la persistencia de tickets, llamadas y CDRs ya que se requieren relaciones 1:N entre tickets, llamadas y CDRs, además de operaciones intensivas de lectura durante el análisis de la incidencia.

Uvicorn + FastAPI (Webhook Server)[7]

FastAPI es un *framework* ligero para el desarrollo de APIs en Python que destaca por su sintaxis declarativa, soporte nativo para modelos de datos con Pydantic y una integración directa con tipado estático. Está diseñado para construir servicios HTTP de alta eficiencia, facilitando operaciones como el parseo de peticiones, la validación automática de parámetros y la generación de documentación interactiva mediante OpenAPI. Su rendimiento lo convierte en una alternativa adecuada para sistemas que requieren recibir eventos externos en tiempo real, como es el caso de la llegada de nuevos tickets desde Jira.

Uvicorn, por su parte, es un servidor ASGI (*Asynchronous Server Gateway Interface*) escrito en Python y basado en *uvloop* y *httptools*, lo que le permite alcanzar un rendimiento significativamente superior al de servidores WSGI tradicionales. Su arquitectura orientada a eventos permite gestionar múltiples conexiones concurrentes con bajo consumo de recursos, así como manejar de forma eficiente peticiones de corta duración propias de flujos webhook.

En el contexto del proyecto, la combinación FastAPI + Uvicorn proporciona un servidor ligero, asíncrono y fácilmente desplegable que actúa como punto de entrada del sistema. Jira envía los tickets recién creados mediante un webhook HTTP hacia un endpoint expuesto por FastAPI, mientras que Uvicorn se encarga de servir el tráfico, aceptar las solicitudes entrantes y garantizar una baja latencia en la recepción del evento [14].

[6] `https://github.com/ollama/ollama`
[7] `https://uvicorn.dev/`

2.7. Estado de la cuestión

La automatización de la gestión de incidencias ha sido un área activa de investigación durante más de dos décadas, con enfoques iniciales basados en minería de texto y modelos estadísticos simples, que permitían un triaje básico pero dependían fuertemente del vocabulario y mostraban dificultades ante descripciones ambiguas o dominios con alta carga técnica. Dentro de estas aproximaciones clásicas destaca el trabajo de Rajeev Kumar Gupta [1] que propuso un sistema capaz de analizar el texto del ticket, identificar entidades clave, localizar incidencias similares, correlacionar la información con la CMDB y estimar el componente responsable del fallo. Aunque este enfoque supuso un avance relevante al integrar información estructurada y no estructurada, seguía limitado por su dependencia de reglas manuales y búsquedas basadas en palabras clave, lo que restringía su capacidad para manejar la variabilidad lingüística y la complejidad semántica propias de dominios técnicos como las telecomunicaciones.

Con el avance del aprendizaje profundo y, en especial, de los modelos basados en arquitecturas *Transformer*, la automatización del análisis de tickets experimentó una mejora notable. Plataformas como ServiceNow [8] incorporaron módulos de clasificación automática basados en modelos neuronales, permitiendo detectar intención, agrupar incidencias similares y enrutar solicitudes con mayor precisión. Estas estrategias mejoraron significativamente la escalabilidad y la consistencia, pero seguían centradas en la clasificación textual y no en el análisis técnico de datos de red o métricas especializadas.

En paralelo, aparecieron plataformas de automatización orientadas específicamente al soporte técnico, como Algomox[9] o NetBrain[10]. Algomox propone asistentes basados en LLM para acelerar el triaje mediante análisis semántico y minería de conocimiento sobre repositorios técnicos, mientras que NetBrain introduce mecanismos de análisis asistido por IA para detectar patrones en tickets históricos y generar guías de resolución. Aunque ambos enfoques avanzan hacia un soporte más automático, su aplicación se orienta principalmente a infraestructuras TI genéricas y no integran análisis profundo de métricas VoIP, códigos de señalización o datos provenientes de CDRs.

Más recientemente, la propuesta TickIt [15] presenta un marco de automatización del ciclo de vida del ticket que utiliza LLMs para actualizar estados, extraer intención, detectar duplicados y asistir en el escalado. Su aportación clave es el uso de

[8] `https://www.servicenow.com/`
[9] `https://algomox.com`
[10] `https://www.netbraintech.com/`

embeddings[11] para detectar incidencias semánticamente equivalentes y gestionar la evolución temporal del ticket. Aun así, continúa siendo una solución genérica que no está preparada para un sector técnico específico que requiere soluciones a medida, como es el caso de las telecomunicaciones.

En el contexto específico del sector, destaca la iniciativa GSMA Open–Telco LLM Benchmarks [16]. Este conjunto de pruebas evalúa rigurosamente el rendimiento de los LLM en tareas propias del dominio, tales como comprensión de documentación 3GPP, razonamiento numérico sobre parámetros de red, identificación de degradaciones, análisis de fallos en infraestructuras de voz y clasificación de documentos técnicos. Los resultados publicados muestran que incluso los modelos más avanzados presentan limitaciones significativas cuando se enfrentan a lenguaje altamente especializado, lo que evidencia la necesidad de adaptar o entrenar modelos específicos para el sector. Este benchmark[12] constituye un punto de referencia fundamental, ya que establece un marco objetivo para medir la utilidad real de los LLM en entornos de telecomunicaciones.

En consecuencia, aunque existen soluciones comerciales y marcos de automatización capaces de clasificar tickets o asistir en su triaje, ninguna aborda de manera integral la naturaleza compleja y altamente especializada de las incidencias de voz en redes internacionales. Las plataformas generalistas operan sobre texto libre y modelos preentrenados, pero no integran de forma nativa análisis técnico de métricas VoIP, correlación con CDRs, ni razonamiento sobre valores como ASR, ALOC o causas de liberación. Asimismo, los estudios del benchmark GSMA Open-Telco LLM Benchmarks evidencian que incluso los LLM más avanzados muestran un rendimiento limitado al enfrentarse a tareas específicas del sector telco, lo que descarta el uso directo de soluciones basadas en estos modelos.

Este escenario justifica el desarrollo de un sistema completamente nuevo y adaptado al flujo real de BTS: una arquitectura híbrida que combina modelos de lenguaje entrenados con datos históricos de la empresa, mecanismos de extracción estructurada y validación técnica, además de implementar módulos explícitos para consultar la base de datos y analizar métricas en tiempo real.

Con este enfoque, el sistema no solo clasifica tickets, sino que reproduce el razonamiento técnico del equipo de CS, garantiza la confidencialidad al ejecutarse íntegramente en la infraestructura interna y permite automatizar decisiones basadas en datos de red que ninguna solución del mercado ofrece de forma equivalente. Esta integración de componentes lingüísticos y analíticos convierte la propuesta en una alternativa especializada, reproducible y alineada con las necesidades reales del entorno

[11] `https://www.ibm.com/mx-es/think/topics/embedding`
[12] `https://huggingface.co/blog/otellm/gsma-benchmarks`

de telecomunicaciones de BTS.

Adicionalmente, aunque el entorno telco de BTS incluye una amplia variedad de tipos de incidencia —cada uno con métricas específicas, flujos de análisis diferenciados y requisitos operativos particulares— este proyecto ha abordado únicamente un subconjunto reducido. Esta selección responde a la necesidad de construir un flujo prototípico funcional que permita validar la arquitectura general, evaluar su eficiencia y demostrar la viabilidad técnica de integrar automatización en el proceso de resolución de incidencias.

La automatización completa de todos los tipos de degradaciones excede el alcance razonable de un Trabajo Fin de Grado debido a la complejidad y al volumen de lógica operativa que implican algunos flujos avanzados. Por este motivo, la solución desarrollada se plantea explícitamente como una prueba de concepto que la empresa utilizará para determinar si resulta apropiado asignar recursos adicionales para extender la automatización al conjunto completo de incidencias gestionadas por el equipo de CS.

Aun así, el flujo implementado actúa como una base extensible: una vez validado su funcionamiento, la arquitectura puede ampliarse progresivamente para incorporar otros tipos de degradaciones y evolucionar hacia un sistema integral de automatización del soporte en telecomunicaciones.

Capítulo 3

Descripción del sistema

Este capítulo describe con detalle el sistema propuesto para automatizar la gestión de incidencias de telefonía IP en BTS. La herramienta, denominada Hefestus, se comporta como un agente de software capaz de leer tickets de Jira, clasificarlos, extraer información técnica, consultar bases de datos de red y proporcionar comentarios automáticos al equipo de CS.

La presente sección explica la arquitectura, los módulos que la componen, los flujos de procesamiento y las funcionalidades actualmente implementadas.

3.1. Visión general

Hefestus surge ante la necesidad de reducir la elevada carga manual asociada a la gestión diaria de incidencias en el departamento de CS. En la actualidad, el equipo debe procesar aproximadamente 800 tickets BTS-VOZ, 600 tickets SBTS-VOZ, 20 tickets relacionados con servicios SMS y alrededor de 5 tickets de Numbering cada día —estos subgrupos son los diferentes departamentos a los que CS cubre como primera línea de soporte—. Este volumen recae sobre un equipo compuesto por cuatro profesionales en Zaragoza, dos en Guatemala y cuatro en Ecuador, organizado bajo un esquema operativo 24×7 distribuido en tres turnos: Zaragoza cubre la franja de 09:00 a 18:00 (CET), Guatemala opera entre 15:00 y 00:00 (CET) y Ecuador entre 00:00 y 09:00 (CET), quedando los fines de semana y festivos asignados al turno en rotación.

Aunque esta organización permite gestionar adecuadamente el volumen diario de incidencias, muchas de las tareas iniciales —como la revisión de métricas, la consulta de bases de datos o la clasificación preliminar— continúan realizándose de forma manual. En este contexto, Hefestus no pretende sustituir este esquema operativo, sino complementarlo aportando automatización en etapas concretas del proceso donde puede ofrecer una ventaja objetiva en términos de rapidez, consistencia y reducción del esfuerzo repetitivo.

El sistema automatiza varias tareas del flujo de trabajo de CS:

– **Ingesta de tickets.** Mediante consultas JQL se recuperan tickets recientes de Jira filtrados por el grupo de CS. La ingesta también se puede realizar a través de webhooks en tiempo real con la creación de la incidencia.

– **Clasificación automática.** Se emplea un modelo BERT entrenado específicamente con un dataset de tickets históricos de BTS clasificados manualmente por el equipo de CS para predecir el tipo de degradación (ASR, ALOC, Rejection, SPAM, etc.) a partir del *summary* y la descripción del ticket. En la Figura 3.1 se muestra un ejemplo de incidencia en Jira donde se aprecian ambos campos. También se ha implementado una política de *fallback* que permite asignar la clase *Others* si la confianza del modelo es baja y un módulo de reglas por palabras clave puede sobrescribir la etiqueta cuando se detecten patrones claros.

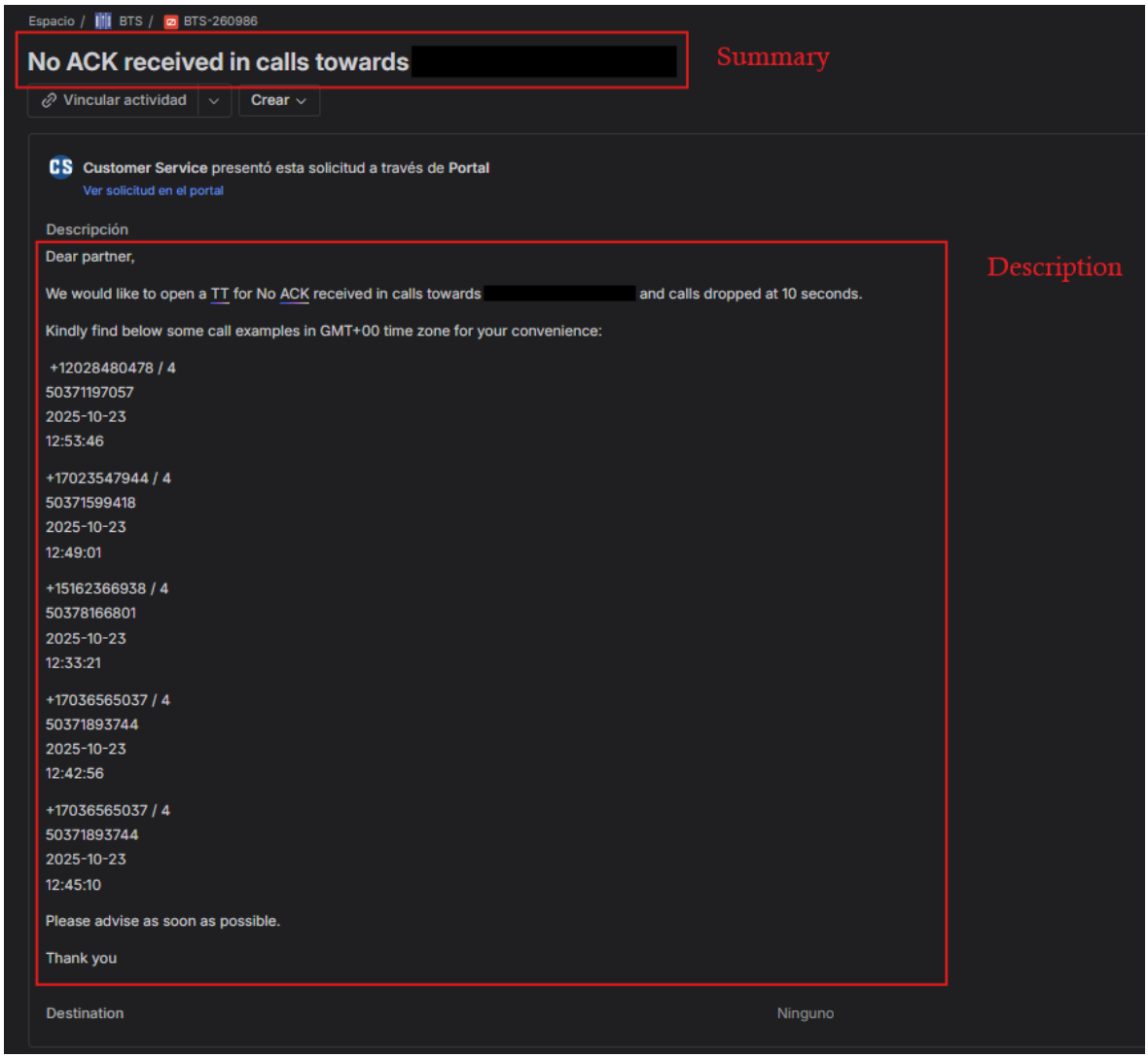

Figura 3.1: Ejemplo de Incidencia en la plataforma de Jira

– **Extracción de llamadas y destino.** Cuando la degradación requiere analizar métricas de red (p. ej. ASR o Rejection), se invoca a un modelo de lenguaje local (LLama-3 instalado en Ollama) para extraer del texto ejemplos de llamadas (ANI/DNI, fechas, horas, duración, código de liberación, zona horaria) y el país o destino mencionado. Lo que servirá para el posterior análisis de esta información extraída de forma estructurada en formato JSON.

– **Persistencia.** Todos los tickets, llamadas de ejemplo y sus correspondientes CDRs obtenidos de Apache Pinot se almacenan en PostgreSQL, utilizando índices únicos para evitar duplicados y reglas automáticas de limpieza.

– **Actualización de tickets.** Opcionalmente, Hefestus escribe en Jira el tipo de degradación inferida y añade comentarios con la información técnica extraída. Esta actualización es la que permite que el equipo de CS tenga a simple vista las métricas de interés y pueda decidir con rapidez si escala la incidencia.

3.2. Arquitectura y componentes

La arquitectura sigue un enfoque modular: los elementos responsables de acceder a servicios externos (Jira, PostgreSQL, Apache Pinot) están encapsulados en clientes, mientras que la lógica de negocio se organiza en servicios y *handlers*. La lista de módulos clave es la siguiente:

– **clients/jira.py** – conecta con Jira a través de su API v3 para buscar incidencias, recuperar campos de interés y actualizar campos personalizados.

– **nlp/classifier.py** – carga el modelo BERT entrenado mediante *fine-tuning*, gestiona el label encoder y aplica un umbral configurable de confianza. También incorpora políticas de fallback para agrupar incidencias dudosas en la clase «Otros».

– **nlp/rules.py** – define reglas heurísticas basadas en palabras clave. Se utiliza como respaldo del clasificador cuando éste no reconoce la clase o cuando hay coincidencias claras con patrones conocidos.

– **services/extraction.py** – prepara el texto combinando summary y descripción, invoca al LLM (Llama-3) y devuelve un diccionario con las llamadas extraídas y el destino detectado.

– **clients/db.py** – gestiona la conexión con PostgreSQL y asegura la existencia de las tablas tickets, calls y cdrs; ofrece métodos de inserción y actualización.

- **services/pipeline.py** – orquesta todo el flujo: ingesta de tickets, clasificación, reglas, extracción de datos, persistencia, consulta de CDRs y generación de comentarios.
- **services/handlers.py** – implementa lógica específica para cada tipo de degradación. Para la prueba de concepto se desarrollaron únicamente los handlers correspondientes a ASR y una primera aproximación al de Rejection debido a su gran correlación, dado que son dos de las categorías más frecuentes en el entorno operativo y permiten validar adecuadamente la arquitectura del sistema. La extensión a otros tipos de incidencia queda prevista para fases posteriores, una vez consolidado el funcionamiento básico del *pipeline* de automatización.
- **scripts/run-pipeline.py** – CLI de lanzamiento que admite distintos flags (–persist-db, –jira-modify, –jira-comment, etc.).

3.3. Infraestructura y despliegue

Esta sección describe el despliegue del sistema implementado en la empresa, para el cual se ha montado una infraestructura dentro de esta que pueda manejar la lógica de Hefestus y haga uso de los recursos de esta.

3.3.1. Máquina virtual y entorno de ejecución

El sistema Hefestus se encuentra desplegado en una máquina virtual dedicada dentro de la infraestructura de BTS. Esta VM ejecuta un sistema operativo Linux (Debian 12) y aloja todos los componentes necesarios para el funcionamiento del pipeline: el servidor webhook basado en FastAPI, el modelo de clasificación BERT, así como las conexiones a las bases de datos PostgreSQL y Apache Pinot y conexión al servidor de BTS donde se encuentra ollama con Llama3.

La máquina virtual dispone de recursos suficientes para soportar la carga de procesamiento de tickets en tiempo real, incluyendo la inferencia de modelos de lenguaje y la ejecución de consultas analíticas a Pinot. El despliegue en una VM aislada permite controlar el entorno de ejecución y gestionar las dependencias de forma independiente al resto de sistemas de producción.

3.3.2. Privacidad y seguridad en el tratamiento de datos

Un aspecto fundamental del diseño de Hefestus es el cumplimiento estricto de las políticas internas de privacidad y protección de datos de BTS. Tanto el modelo BERT como la instancia de Llama 3 ejecutada mediante Ollama procesan la información

exclusivamente en entornos locales, sin establecer conexiones salientes hacia servicios externos ni requerir infraestructura alojada por terceros.

El flujo completo de tratamiento —desde la recepción del webhook procedente de Jira hasta la clasificación del ticket, el análisis de métricas y la generación del comentario final— se mantiene dentro de la red interna de la empresa, garantizando que los datos no abandonan los sistemas controlados por BTS. En consecuencia, ningún contenido de los tickets, incluyendo descripciones, trazas, logs o identificadores asociados a clientes o proveedores, es transmitido a terceros ni utilizado para telemetría o analítica ajena al propio sistema.

Este enfoque asegura que el funcionamiento de Hefestus se alinea con las directrices corporativas de seguridad, minimiza la exposición de datos sensibles y evita cualquier dependencia de servicios externos que pudiera comprometer la confidencialidad de la información gestionada por el departamento de CS.

3.3.3. Túnel Cloudflare y accesibilidad externa

Para que Jira pueda enviar webhooks al sistema sin exponer directamente la infraestructura interna de BTS, se ha configurado un túnel de Cloudflare que publica el endpoint del webhook server de forma segura. Este túnel establece una conexión cifrada entre Cloudflare y la máquina virtual, permitiendo que el sistema sea accesible desde el exterior a través del dominio `hefestus-forge.bts.io`.

El endpoint `https://hefestus-forge.bts.io/webhook/jira` recibe las notificaciones de Jira cada vez que se crea o actualiza un ticket en el proyecto configurado. Cloudflare actúa como proxy reverso, gestionando certificados SSL/TLS y proporcionando una capa adicional de seguridad mediante su sistema de protección DDoS y filtrado de tráfico malicioso.

Esta arquitectura elimina la necesidad de configurar reglas complejas en el firewall corporativo y permite desplegar el sistema de forma ágil, manteniendo al mismo tiempo los estándares de seguridad de la empresa.

3.3.4. Proyecto de preproducción en Jira

Con el objetivo de probar y validar el sistema sin afectar al entorno de producción, se ha creado un proyecto específico en Jira denominado **HEF** (Hefestus). Este proyecto actúa como entorno de preproducción y replica automáticamente los tickets creados en el proyecto real de producción, identificado con el código **BTS**.

Se ha implementado un automatismo en Jira que se ejecuta cada vez que se crea un nuevo ticket en el proyecto BTS. Este automatismo realiza las siguientes acciones:

1. Copia el ticket recién creado desde el proyecto BTS al proyecto HEF, replicando todos los campos relevantes (*summary*, *description*, organización, prioridad, etc.).

2. Genera un nuevo identificador (key) para el ticket copiado en el proyecto HEF.

3. Envía un webhook al endpoint `https://hefestus-forge.bts.io/webhook/jira` con la clave del nuevo ticket del proyecto HEF.

De esta forma, Hefestus recibe notificaciones únicamente de tickets del proyecto HEF y todas las modificaciones que realiza el sistema (actualización de campos personalizados, adición de comentarios con métricas, clasificación automática) quedan reflejadas exclusivamente en estos tickets de preproducción. El proyecto BTS permanece intacto, evitando cualquier riesgo de interferencia con el flujo de trabajo real del equipo de CS. Esta estrategia permite:

- Validar el comportamiento del sistema con datos reales sin impacto en producción.
- Revisar y auditar los comentarios y actualizaciones generados automáticamente antes de aplicarlos al entorno real.
- Realizar ajustes en los modelos, *handlers* y configuraciones sin interrumpir la operativa del equipo.
- Facilitar la transición gradual hacia una integración completa con el proyecto BTS una vez que el sistema haya demostrado su fiabilidad.

El uso del proyecto HEF como entorno de preproducción constituye una práctica de despliegue seguro que minimiza riesgos y permite iterar sobre el sistema de forma controlada.

3.4. Flujo de procesamiento de tickets

El flujo de procesamiento se ha diseñado para reproducir, de forma automatizada, el razonamiento que realiza el equipo de CS. A continuación se detallan los pasos principales ejecutados por el *pipeline* principal representados por la Figura 3.2:

Ingesta y filtrado de tickets.

La aplicación se puede ejecutar en dos modos. *Pull* programado utilizando la CLI *scripts/run-pipeline.py* se consultan las incidencias creadas en la última hora (configurable) a través de JQL. Esta opción permite ejecutar análisis periódicos o

manuales, persistiendo los datos en PostgreSQL actualizando los tickets según los flags que haya activados. Esta opción permite paginación en la nueva API de Jira por lo que se pueden extraer tickets superando el límite de 100 sin utilizar paginación.

En el segundo modo de ejecución Jira puede invocar un endpoint HTTP cada vez que se crea una nueva incidencia en el proyecto de "HEF", el cual es un proyecto de desarrollo que copia todos los tickets creados en el proyecto "BTS"(producción). Hefestus incluye un servidor en webhook-server.py (basado en FastAPI) que expone /webhook/jira. Mediante ngrok o un proxy reverso se publica este endpoint; al recibir la clave del ticket se invoca a la API de Jira para obtener sus campos completos.

Preparación y limpieza del texto.

El campo description de Jira se devuelve en Atlassian Document Format (ADF), un JSON con etiquetas anidadas que puede superar los 5000 tokens. Para que el modelo de lenguaje pueda procesarlo, se realiza una limpieza exhaustiva que elimina etiquetas, metadatos y campos irrelevantes, obteniendo un texto plano conciso. A continuación se concatena el summary con la descripción y se pasa a minúsculas.

Clasificación del ticket

La identificación del tipo de incidencia se realiza mediante un modelo *BERT* entrenado con un histórico de tickets etiquetados manualmente por el equipo de CS. Este modelo se complementa con un clasificador basado en reglas heurísticas que actúa como mecanismo de *fallback* cuando la confianza del modelo principal es insuficiente. El proceso sigue las siguientes etapas:

- El texto limpiado por la fase anterior se envía al clasificador *BERT*, que devuelve una etiqueta candidata, su probabilidad asociada y el listado de las cinco clases con mayor probabilidad (*top-5*).

- Si la probabilidad de la etiqueta propuesta por *BERT* es inferior a un umbral configurable, el sistema activa el módulo de reglas heurísticas. Este clasificador analiza la presencia de palabras clave definidas en `rules.yml`; por ejemplo, términos como *fake DLR* o *false delivery report* fuerzan la asignación de la clase *Fake DLR*.

- Si ninguna regla coincide con el contenido del ticket y la confianza de *BERT* sigue siendo insuficiente, la incidencia se etiqueta como *Others*. Esta categoría indica que el ticket no puede ser procesado automáticamente, ya que no es posible asociarlo a un flujo técnico específico, por lo que se deriva directamente al equipo humano.

El resultado final de esta fase incluye la etiqueta asignada, la probabilidad obtenida, el conjunto de clases *top-5* y el origen de la decisión (modelo *BERT*, reglas heurísticas o *fallback*). Esta estrategia combinada garantiza robustez ante casos ambiguos y permite mantener un nivel adecuado de precisión incluso en incidencias poco frecuentes o difícilmente representadas en el dataset de entrenamiento.

Extracción de llamadas y destino

Si la degradación predicha es de las que requieren comprobar métricas (actualmente *ASR* y *Rejection*), se invoca a *ExtractionService*. Este servicio envía al LLM (Llama-3) el texto del ticket y obtiene una estructura con los siguientes campos por llamada: ANI, DNI, date, timeAnswered, timeFinished, duration, releaseCause, timezone y sourcehint. También identifica el país o destino. Esta extracción automatizada emula la tarea manual de revisar los mensajes del cliente y copiar las referencias de llamada. En el Anexo D se expone una muestra del prompt utilizado en las consultas a Llama 3 para la extracción de las llamadas.

Debido a que los textos de los tickets son muy ambiguos y diferentes entre ellos, el modelo puede fallar y presentar inconsistencias en la información extraída. Por lo que se realiza una validación de los datos extraídos por el modelo para asegurarse que este ha devuelto el formato adecuado —como números de menos de ocho dígitos o fechas en formato diferente al que soporta PostgreSQL—.

Persistencia

Si se activa el flag –persist-db, el ticket y sus llamadas se insertan en la base de datos. El módulo clients/db.py se encarga de verificar la existencia de las tablas, ejecutar inserciones idempotentes y, si es necesario, actualizar registros existentes.

Verificación de CDRs

La base de datos consultada es un clúster de *Apache Pinot* configurado como almacén "OLAP"para analítica en tiempo real. Por política interna de la empresa, esta base de datos mantiene únicamente una persistencia de tres días, ya que está optimizada para cargas de trabajo de baja latencia y alto volumen, y no para almacenamiento histórico a largo plazo. En consecuencia, si las llamadas aportadas por el cliente fueron realizadas fuera de ese intervalo temporal, los CDRs correspondientes ya no estarán disponibles para su verificación y será necesario solicitar ejemplos más recientes.

A pesar de esta limitación temporal, *Pinot* constituye la fuente más adecuada para la verificación de CDRs en tiempo real debido a su capacidad para ejecutar consultas agregadas en milisegundos y por la estructura enriquecida de la tabla utilizada en

este proyecto. Dicha tabla no solo almacena el CDR bruto de cada llamada, sino que incorpora información adicional —como el cliente, el proveedor, el destino, las rutas implicadas y métricas complementarias— generada a través de procesos internos de ingesta y enriquecimiento de datos. Si estas búsquedas se realizaran en otras fuentes de datos internas, sería necesario reconstruir manualmente estas uniones y agregaciones, lo que incrementaría significativamente la complejidad y el tiempo de análisis.

Antes de describir las estrategias de búsqueda, se presenta una breve leyenda de las siglas utilizadas en las consultas a *Pinot*:

CN (*Calling Number*): número que inicia la llamada utilizado en las consultas a Pinot.

DN (*Dialed Number*): número al que va dirigida la llamada utilizado en las consultas a Pinot.

ANI Corresponde al *CN* extraído de las llamadas de ejemplo del ticket.

DNI Corresponde al *DN* extraído de las llamadas de ejemplo del ticket.

ST (*Seize Time*): instante de inicio de la llamada.

Para cada llamada extraída dentro del periodo de tres días, se busca en Apache Pinot un CDR que coincida con el ANI, DNI, la fecha y hora de la llamada, dentro de una ventana de ±2 minutos. Si se encuentran coincidencias, se asocian los CDRs encontrados con la llamada y se almacena en la tabla "cdrs"; de lo contrario, se genera un comentario pidiendo al cliente nuevas llamadas de ejemplo debido a que no se han podido encontrar en el sistema las aportadas para poder investigar la incidencia.

Se siguen dos estrategias diferentes en caso de que no se encuentren CDRs con los datos originales extraídos por el modelo. Las consultas SQL a Pinot suelen componerse de ANI, DNI y seizeTime, pero estas pueden fallar por lo que se han implementado diferentes estrategias en caso de que fallen el resto. Las cuatro estrategias posibles cuando se reciben dos numeros (ANI y DNI) y time_answered son las siguientes:

$$\begin{aligned}
\text{Estrategia base:} &\quad CN = ANI \land DN = DNI \land ST \\
\text{Estrategia base sin tiempo:} &\quad CN = ANI \land DN = DNI \\
\text{Estrategia invertida:} &\quad CN = DNI \land DN = ANI \land ST \\
\text{Estrategia invertida sin tiempo:} &\quad CN = DNI \land DN = ANI
\end{aligned}$$

La estrategia base consiste en buscar un registro cuyos campos `CN` y `DN` coincidan exactamente con el `ANI` y `DNI` proporcionados en el ticket, respectivamente, junto con el instante de inicio de la llamada (`ST`).

No obstante, esta correspondencia puede fallar en situaciones en las que el cliente haya intercambiado involuntariamente los números al redactar el ticket, o cuando el modelo Llama 3 haya interpretado de forma incorrecta cuál corresponde al número llamante y cuál al marcado. Para cubrir estos casos, se aplica una estrategia invertida, en la que se intercambian los parámetros de búsqueda.

De este modo, se contemplan ambos órdenes posibles de correspondencia entre los números de ejemplo y los campos de Pinot, aumentando la probabilidad de localizar los CDRs correctos cuando existe ambigüedad o error en los datos iniciales.

En caso de que solo se haya proporcionado un número entonces se sigue la siguientes estrategias en caso de que hayan pasado time_answered:

$$\begin{aligned} \text{Estrategia reducida:} \quad & CN = ANI \vee CN = DNI \wedge ST \\ \text{Estrategia reducida sin tiempo:} \quad & CN = ANI \wedge CN = DNI \end{aligned}$$

Esto aumenta el tiempo de búsqueda y el número de consultas, pero aumenta las probabilidades de encontrar los CDRS de cada llamada.

***Handlers* específicos**

Una vez localizados los CDRs, se ejecuta un handler específico según la degradación. Este módulo encapsula las consultas a Pinot necesarias para obtener métricas y generar el comentario final.

Actualmente está implementado el handler de ASR el cual se puede utilizar para la resolución de varios tipos de incidencia, no solo las específicas de ASR.

En este flujo se sacan métricas de ASR para el cliente que ha abierto el ticket y que aparece en las llamadas de ejemplo, de sus destinos y sus proveedores indicando los posibles problemas que pueden tener estos de ASR en comparación a su media.

Actualización y comentarios en Jira

Dependiendo de los flags configurados, Hefestus actualiza el campo personalizado typeOfDegradation con la etiqueta final y añade un comentario con las métricas generadas. Si no se han encontrado CDRs, el comentario invita al cliente a proporcionar registros adicionales. Todo esto queda visible para el equipo de CS en el ticket, evitando consultas manuales a múltiples sistemas.

Exportación a CSV

El pipeline puede exportar los tickets procesados, llamadas y CDRs a un fichero CSV para análisis posteriores. Esta opción se utiliza en pruebas controladas y evaluaciones

del sistema.

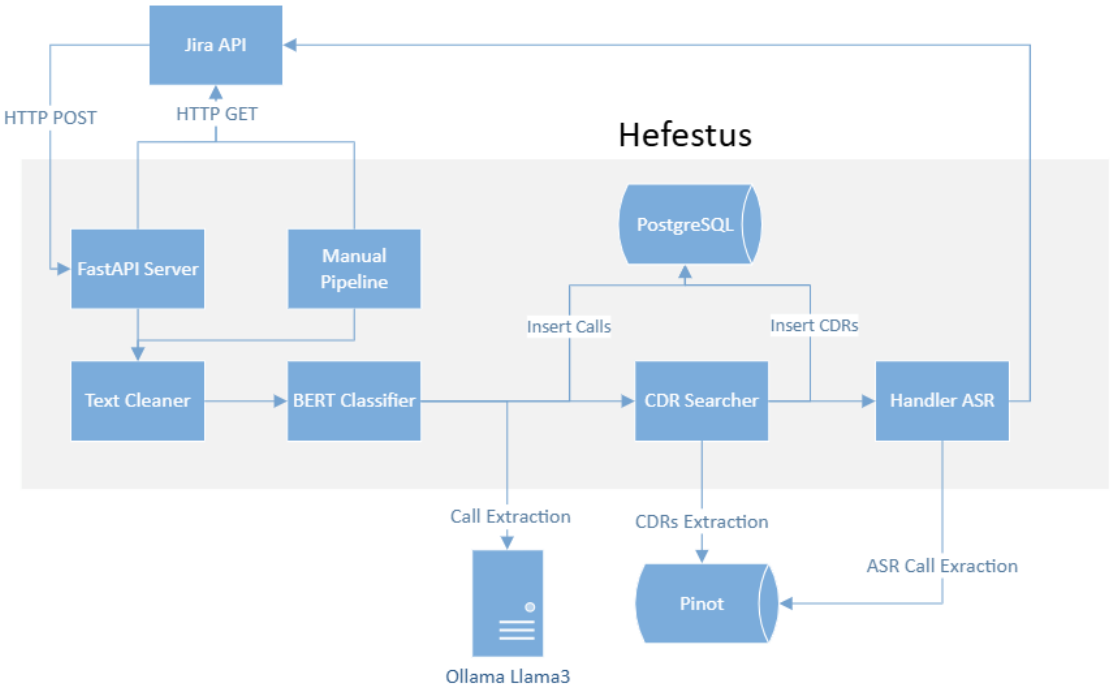

Figura 3.2: Diagrama de infraestructura

Capítulo 4

Comparativa y evaluación de modelos de clasificación de incidencias

4.1. Introducción

En este capítulo se describe el proceso de desarrollo y entrenamiento de modelos de lenguaje orientados a la clasificación automática de incidencias a partir de los textos contenidos en los tickets de la empresa. Para la construcción del conjunto de datos se empleó la API de Jira, a través de la cual se extrajo el histórico de tickets relevantes para el proyecto. Con esta información se elaboró un dataset estructurado que sirvió como base para el entrenamiento de los modelos.

El entrenamiento se llevó a cabo de manera iterativa, partiendo de una primera versión del modelo BERT y evolucionando progresivamente hacia una versión optimizada mediante ajustes en la preparación de datos y en la configuración del entrenamiento. Paralelamente, se desarrolló un modelo alternativo basado en reglas heurísticas y palabras clave (keyword matching), concebido como sistema de respaldo en aquellos casos en los que el modelo de lenguaje no alcanzase un nivel de precisión satisfactorio.

Con el fin de comparar ambos enfoques y analizar la evolución del sistema, se presentan métricas de evaluación ampliamente utilizadas en clasificación automática, tales como accuracy, precision, recall y F1-score. Estas medidas permitirán valorar objetivamente el rendimiento de cada modelo y visualizar sus diferencias a través de representaciones gráficas.

Este capítulo sienta las bases para demostrar la transición desde un enfoque inicial basado en heurísticas hasta una solución madura basada en aprendizaje profundo

4.2. Preparación de datasets

Para iniciar el proceso de entrenamiento, fue necesario realizar una exportación de tickets desde Jira, filtrando aquellos dirigidos al grupo CS, dado que el tratamiento de incidencias del proyecto se centra en este tipo específico de tickets.

El dataset resultante se almacenó en formato JSON, incluyendo para cada ticket los siguientes campos: `summary`, `type_of_degradation`, `organization` y `description`. Estos atributos constituyen la base de entrenamiento de los modelos desarrollados.

No obstante, durante las primeras pruebas se identificó un problema importante relacionado con el formato de los textos devueltos por la API de Jira. La descripción de los tickets se encontraba representada en un lenguaje propio ADF, compuesto por etiquetas, estructuras anidadas y metadatos que introducían un elevado nivel de ruido. Esto se tradujo en un promedio de 5.314 tokens por ticket, cifra muy superior al máximo de 512 tokens que soporta el modelo BERT-base uncased. En la práctica, esto implicaba que únicamente se podían utilizar los primeros 512 tokens de cada ticket, descartando información potencialmente relevante.

Para solventar esta limitación, se desarrolló un proceso de limpieza de texto que elimina etiquetas, metadatos y campos irrelevantes, generando una representación más concisa en texto plano.

Dado que los tickets completos resultan demasiado extensos para incluirlos en este capítulo, se han incorporado ejemplos íntegros tanto del formato original exportado desde Jira como del formato limpiado en el Anexo A.

El impacto de esta transformación se refleja en la Tabla 4.1 , donde se observa la reducción en el número de tokens:

	Dataset base	Dataset limpiado
Media	5314	212
Mínimo	9	4
Máximo	185418	18322

Tabla 4.1: Comparativa del número de tokens en el dataset original frente al dataset procesado

Como puede apreciarse, el dataset procesado resulta mucho más eficiente para el entrenamiento, evitando la pérdida sistemática de información y reduciendo de forma drástica el coste computacional.

Además de su utilidad para el entrenamiento, este procedimiento de limpieza también se aplicará en la fase de inferencia, permitiendo procesar y analizar tickets en tiempo real con un formato adecuado para su clasificación automática.

En cuanto a la distribución de clases, el dataset completo contenía 59.982 tickets, de los cuales 43.748 correspondían a la clase SPAM, lo que suponía un fuerte desbalanceo. Este sesgo dificultaba el entrenamiento eficaz del modelo, ya que las clases minoritarias quedaban infrarepresentadas. Para mitigar este problema, en la versión final del dataset se limitaron el máximo de 2.000 tickets por clase para reducir el desbalanceo de clases y se introdujeron pesos por clase durante el entrenamiento para favorecer un mayor equilibrio en la predicción de clases minoritorias. Un ejemplo del desbalanceo de clases en el dataset usado incialmente para el entrenamiento se puede encontrar en el Anexo B.7.

En conclusión, se trabajó con dos datasets diferenciados:

- **Dataset inicial (Base)**, compuesto por 59.982 tickets sin limpieza y con un fuerte desbalanceo de clases, empleado en la primera versión de BERT.

- **Dataset optimizado (enhanced)**, con un máximo de 2.000 tickets por clase y pesos de balanceo, acompañado de un proceso de limpieza textual, utilizado en la versión final del modelo BERT.

4.3. Entrenamiento de modelos

En esta sección se describe el proceso de entrenamiento y configuración de los tres enfoques evaluados en este trabajo: (i) el modelo inicial `BERT_Base`, entrenado sobre un dataset desbalanceado y con menor preprocesamiento; (ii) la versión mejorada `BERT_Enhanced`, optimizada mediante técnicas de limpieza de datos y balanceo de clases; y (iii) el modelo alternativo basado en reglas heurísticas, concebido como sistema determinista de respaldo.

4.3.1. BERT Base

Como se comenta en la sección anterior este primer modelo se ha entrenado a partir de un dataset compuesto por 59982 tickets extraídos desde Jira, sin aplicar procesos de limpieza de texto más allá de la concatenación de los campos summary y description. Este conjunto de datos presentaba un fuerte desbalanceo de clases, donde la clase SPAM representaba más del 70 % de los registros. Estos tickets estaban en formato ADF y se dividieron en 60 % entrenamiento, 10 % validación y 30 % tests. Por lo que el soporte que se daba en el entrenamiento para las diferentes clases se ve representado en el Anexo B.7, donde comparado con el soporte que se da a la clase SPAM, el resto no parecen relevantes para el entrenamiento.

En cuanto a la configuración del entrenamiento se usó el modelo `bert-base-uncased`[1], la tokenización estaba truncada a 128 tokens por ticket. Un batch size de 16, 3 épocas, evaluación al final de cada época y se usaron las métricas accuracy y F1-score ponderado.

Aun siendo un entrenamiento básico sin ajustes en el dataset y mejoras en el proceso de entreno el modelo mostró un alto rendimiento global, con valores de accuracy cercanos al 95 %. Sin embargo, este desempeño estaba sesgado hacia clases mayoritarias, en especial SPAM y Rejection. Estos datos se pueden apreciar en la Figura **??** . Entre las limitaciones más destacadas se encuentran:

- Sobreajuste a clases dominantes, debido a la falta de balanceo.
- Bajo rendimiento en clases minoritarias, con métricas pobres en categorías con menos de 50 instancias.
- Ruido en los textos de entrenamiento, al no haber aplicado limpieza previa sobre el formato ADF.

4.3.2. Análisis de los resultados de BERT_Base con dataset balanceado

La evaluación de BERT_Base sobre el conjunto balanceado y preprocesado mostró un descenso notable en las métricas globales respecto al ensayo inicial. En concreto, el *F1-score* ponderado se redujo de valores superiores al 94 % hasta aproximadamente un 80 %, mientras que la *accuracy* ponderada cayó cerca de 15 puntos porcentuales. Este comportamiento no obedece a un fallo del modelo, sino a la propia naturaleza de los datos y a las características del entrenamiento original.

El entrenamiento de BERT_Base se realizó con un corpus de 59 982 tickets en formato ADF sin limpieza y altamente desbalanceado: más del 70 % de las muestras correspondían a *SPAM* y cerca de un 7 % a *Rejection*. Dicho desbalanceo impulsó artificialmente las métricas globales, ya que el modelo aprendió a optimizar su precisión en las clases mayoritarias. Al evaluarse sobre un dataset balanceado, las clases minoritarias adquieren un peso proporcional mucho mayor. El modelo, al no haber visto suficientes ejemplos de estas clases durante el entrenamiento, muestra un recall bajo y consecuentemente un F1 menor en categorías con pocas instancias, afectando a las métricas globales.

Por otro lado, la limpieza del texto eliminó etiquetas ADF y metadatos irrelevantes, reduciendo la longitud media de cada ticket de 5.314 a 212 tokens. Aunque esta

[1] `https://huggingface.co/google-bert/bert-base-uncased`

depuración facilita la comprensión del contenido por parte del modelo, también elimina parte de la redundancia y de los patrones superficiales que BERT_Base explotaba en su versión original. El truncado a 128 tokens durante el entrenamiento inicial hacía que la mayoría de las entradas contuvieran únicamente información procedente del `summary`, por lo que la pérdida de contexto al limpiar los datos penaliza especialmente a las clases menos frecuentes. Además, BERT_Base no utilizó técnicas de *undersampling* ni pesos por clase, lo que contribuyó a que la red se ajustase en exceso a las categorías dominantes.

La combinación de un conjunto de pruebas equilibrado, una entrada de texto más concisa y la ausencia de balanceo en el entrenamiento explica la caída de rendimiento observada en BERT_Base. Este resultado refuerza la necesidad de emplear métodos de balanceo de clases y preprocesamiento avanzado, como se realizó en la versión BERT_Enhanced, para obtener modelos robustos y generalizables.

4.3.3. BERT Enhanced

Con el fin de solventar las limitaciones del modelo inicial, se entrenó una segunda versión del clasificador basada en mejoras tanto en la preparación del dataset como en la función de entrenamiento. En cuanto a las mejoras implementadas se eliminaron etiquetas, metadatos y campos irrelevantes reduciendo la media de tokens de 5.314 a 212 por ticket, lo que permite aprovechar mejor la capacidad de BERT. Se aplicó undersampling de hasta 2000 instancias por clase, lo que redujo la influencia de SPAM y equilibró el dataset. También se incorporó una función de pérdida ponderada (*CrossEntropyLoss* con pesos por clase) para compensar las clases minoritarias. Para finalizar se usó "*DataCollatorWithPadding*"para aplicar *padding* dinámico, reduciendo el cómputo innecesario en cada lote.

La configuración del entrenamiento es la siguiente:

- Modelo bert-base-uncased
- Dataset balanceado: máximo 2000 tickets por clase
- Tokenización: 256 tokens por ticket.
- Batch size: 16
- Épocas: 8
- Estrategia de evaluación: al final de cada época.
- Métricas accuracy y F1-score ponderado.

Una vez aplicado se ha visto mayor rendimiento en clases minoritarias, evitando que quedaran eclipsadas por SPAM, generalización más robusta gracias al uso de un dataset limpio y balanceado y una reducción de tokens innecesarios con tiempos de entrenamiento más eficientes.

4.3.4. Modelo de Reglas

De forma complementaria a los modelos basados en aprendizaje profundo se desarrolló un clasificador determinista basado en reglas escritas en formato YAML. Este enfoque se fundamenta en la detección de palabras clave asociadas a cada tipo de incidencia.

Cada clase se define mediante un conjunto de expresiones clave que, en caso de aparecer en el texto del ticket, determinan directamente la categoría asignada.

Ejemplo:

```
- target: "Fake DLR"
  keywords:
    - "fake dlr"
    - "fake delivery report"
    - "false dlr"
    - "dlr mismatch"
```

Este clasificador no aprende patrones complejos, sino que aplica coincidencia exacta o aproximada de cadenas. Su ejecución es inmediata y no requiere de entrenamiento, lo que lo hace útil como fallback cuando BERT no alcanza un nivel aceptable de confianza.

4.4. Evaluación y resultados

Vamos a analizar los resultados obtenidos por los tres enfoques de clasificación de incidencias: Bert_Enhanced, Bert_Base y Reglas. Para cada uno se han calculado métricas globales (*accuracy, precisión, recall y F1-score* en sus variantes *macro* y *weighted*) así como métricas específicas por clase.

4.4.1. Resultados globales

La evaluación comparativa de los tres enfoques se llevó a cabo empleando métricas estándar en problemas de clasificación: *accuracy, precision, recall y F1-score*, tanto en sus variantes macro-promediadas como ponderadas (weighted).

En la figura 4.1 se observa la comparación de *accuracy* y *F1-score* (*weighted*) de los tres modelos. El modelo Bert_Base alcanzó un mejor rendimiento global, con valores superiores al 94 % en ambas métricas. Por su parte, la versión Bert_Enhanced obtuvo resultados algo inferiores (entorno al 87 %), y el modelo de reglas les sigue alcanzando

un 60 % lo que pone de manifiesto las limitaciones de un enfoque determinista basado exclusivamente en palabras clave.

De forma complementaria , en esta misma figura se muestran los resultados de precisión y recall en su variante ponderada (weighted). Al igual que en el caso anterior, BERT inicial mantiene el mejor desempeño global

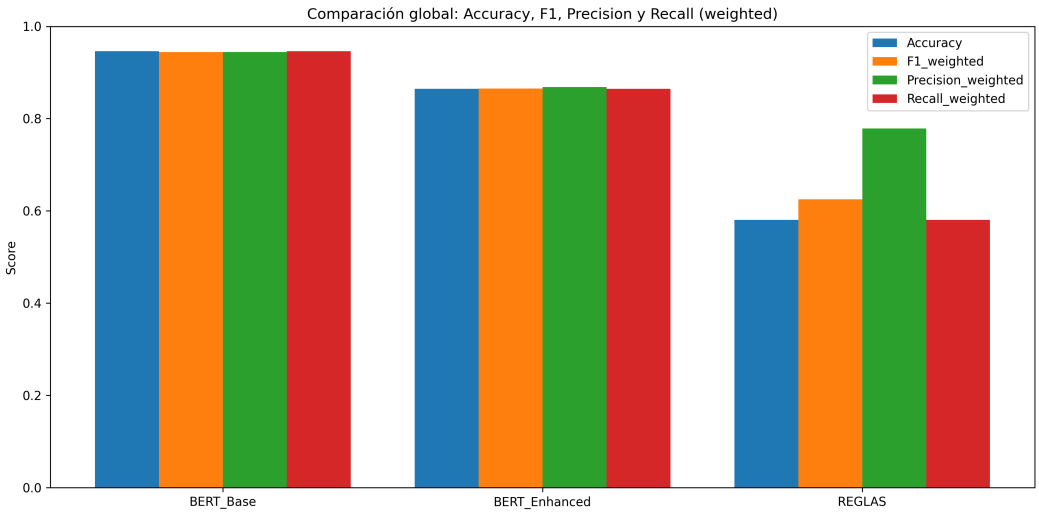

Figura 4.1: Comparación de accuracy, f1-score, precisión y recall weighted de los modelos analizados

En cuanto a las métricas en variantes macro vemos resultados diferentes representados en la figura 4.2. Esta recoge los resultados promedio macro (es decir, dando el mismo peso a cada clase sin considerar su frecuencia). En este caso las diferencias entre ambos modelos BERT y las reglas se hacen más evidentes. Mientras que BERT inicial y mejorado alcanzan valores próximos al 70 %, el modelo de reglas cae respecto a los otros.

También se puede apreciar cómo el modelo final de BERT que parecía inferior en las gráficas anteriores ahora alcanza un 80 % en precisión, recall y f1 mientras que el modelo inicial cae a un 70 %.

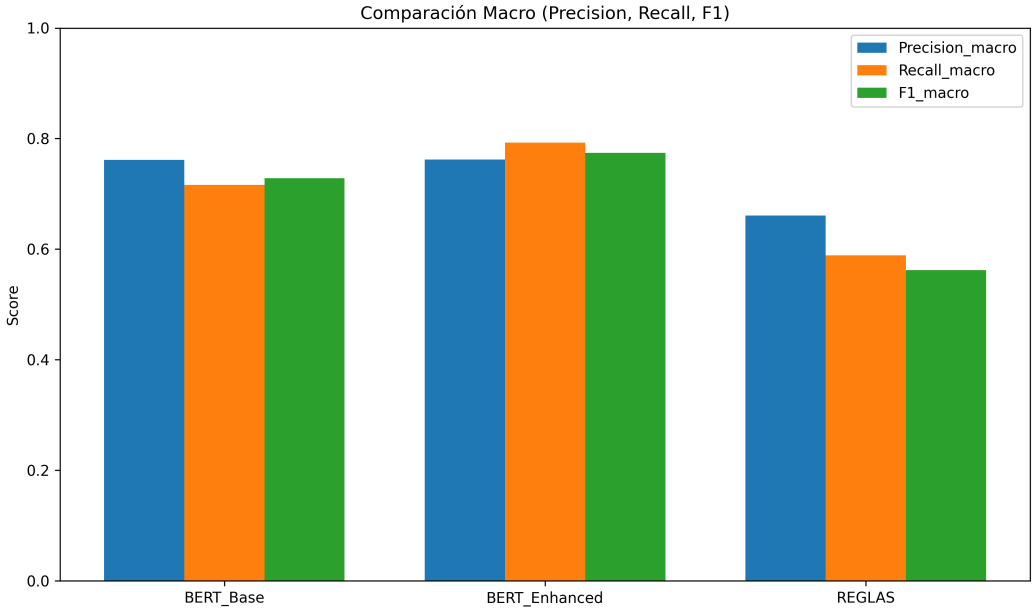

Figura 4.2: Comparación de accuracy, f1-score, precisión y recall macro de los modelos analizados

Con el fin de disponer de una comparación más alineada con el comportamiento esperado en producción, se realizó una evaluación adicional aplicando a BERT_Base exactamente el mismo procedimiento de limpieza y el mismo conjunto de test balanceado utilizado para BERT_Enhanced. Esta prueba no pretende reanalizar en detalle las causas ya discutidas relativas al desbalanceo del entrenamiento original, sino verificar cómo se comportan ambos modelos bajo condiciones homogéneas y comparables.

Los resultados, representados en la Figura 4.3, muestran que el rendimiento ponderado de BERT_Base desciende hasta valores próximos a 0.80, mientras que BERT_Enhanced mantiene valores en torno a 0.86. Esta diferencia no añade nueva información respecto a las limitaciones estructurales del modelo base —ya analizadas previamente—, pero sí permite constatar que, en un escenario realista donde el sistema opera con textos depurados y clases equilibradas, BERT_Enhanced ofrece un comportamiento más estable y menos sesgado. El modelo de reglas, por su parte, se mantiene claramente por debajo de ambos modelos BERT, lo que confirma su papel como mecanismo auxiliar solo útil en casos muy específicos.

Esta comparación final tiene, por tanto, un carácter validatorio: demuestra que, una vez eliminados los sesgos del dataset original, la ventaja de BERT_Enhanced no es circunstancial, sino consecuencia directa de un entrenamiento más adecuado para un entorno de producción. En consecuencia, BERT_Enhanced se consolida como la opción recomendada para el despliegue operativo del sistema.

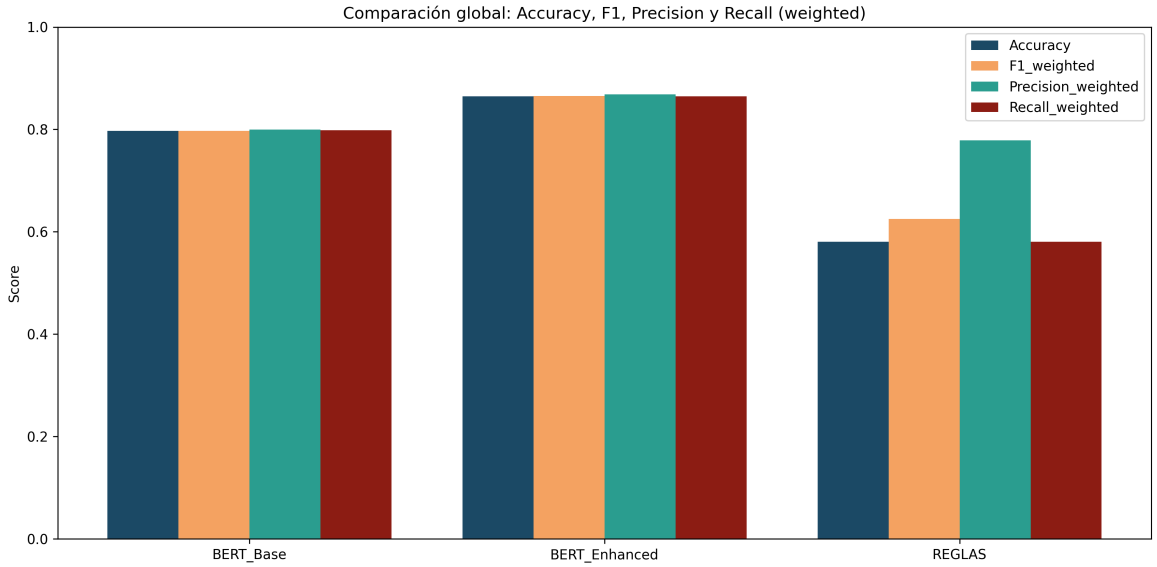

Figura 4.3: Comparación de accuracy, f1-score, precisión y recall weighted con el mismo conjunto de tests balanceados

4.4.2. Resultados por clase

La figura 4.4 muestra la comparación del F1-score a nivel de clase. Los resultados evidencian un cambio respecto al análisis global ya que el modelo BERT_Enhanced obtiene mejor rendimiento en la mayoría de categorías, con especial relevancia en clases minoritarias o de difícil predicción.

El modelo BERT_Base, en cambio, se mantiene competitivo en clases con mayor representación o con patrones consistentes como SPAM, Wangiri o Rejection. Sin embargo pierde precisión en categorías más desbalanceadas y tiende a favorecer a clases mayoritarias.

El modelo de Reglas heurísticas solo muestra utilidad en dos categorías: Fake DLR (F1=0.667) y Packet Loss (F1=0.789), donde la presencia de palabras clave explícitas permite una predicción efectiva. En el resto de clases, su desempeño es muy inferior frente a los modelos basados en aprendizaje profundo, lo que refleja su escasa capacidad de generalización.

En síntesis, el análisis por clase confirma que BERT_Enhanced ofrece un rendimiento más equilibrado y consistente, superando al modelo inicial en la mayoría de las categorías, especialmente en aquellas con menor frecuencia de ejemplos.

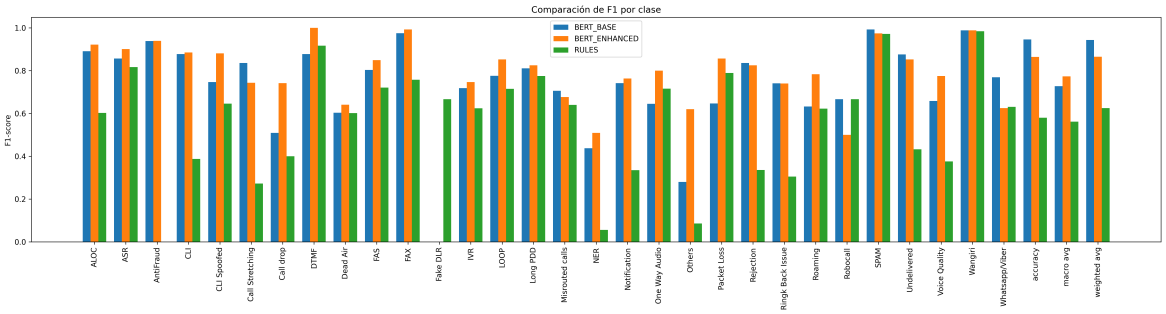

Figura 4.4: Comparación F1 por clase para los modelos propuestos

4.5. Discusión

Los resultados permiten extraer varias conclusiones relevantes:

- **BERT_Base** logra un rendimiento sobresaliente en términos globales (accuracy y F1-weighted), en gran medida porque fue entrenado con un dataset de aproximadamente 59 000 tickets, de los cuales más de 43 000 correspondían a la clase *SPAM* y alrededor de 4 000 a la clase *Rejection*. Este fuerte desbalance explica que el modelo obtenga valores globales muy elevados, ya que se encuentra optimizado para las clases mayoritarias, que dominan el conjunto de datos. No obstante, este sesgo compromete su capacidad de generalización hacia clases menos representadas, tal y como se observa en las métricas macro.

- **BERT_Enhanced**, a pesar de mostrar métricas globales algo inferiores, constituye una alternativa más sólida a nivel de clase. Su entrenamiento se realizó sobre un dataset balanceado mediante *undersampling*, limitando a 2 000 el número máximo de ejemplos por clase, e incorporando el uso de *class weights*. Esta estrategia permitió mejorar de forma significativa el desempeño en categorías minoritarias y más difíciles de distinguir, lo que aporta un mejor equilibrio en escenarios de producción donde no todas las incidencias aparecen con la misma frecuencia.

- El **modelo de Reglas heurísticas** se revela como un recurso útil únicamente en situaciones muy concretas, en las que la presencia de términos clave explícitos (p. ej., *Wangiri* o *FAX*) asegura la clasificación. Sin embargo, su naturaleza determinista y falta de capacidad de generalización hacen que no sea adecuado como sistema principal de clasificación.

En definitiva, los resultados refuerzan la superioridad de los modelos basados en aprendizaje profundo frente a un enfoque puramente heurístico. Si bien

BERT_Base muestra los mejores valores globales debido al sesgo del dataset original, **BERT_Enhanced** se configura como la opción más adecuada para un entorno real de producción, al equilibrar rendimiento y capacidad de generalización en clases menos frecuentes.

Capítulo 5

Procesamiento y Diagnóstico de Degradaciones ASR

5.1. Introducción

La Answer-Seizure Ratio (ASR) es la métrica estándar que utiliza la industria de las telecomunicaciones para medir la eficacia de las llamadas de voz. Se calcula como el porcentaje de llamadas contestadas sobre el total de llamadas realizadas. Junto con la duración media de la llamada (ACD/ALOC), el ASR permite evaluar la calidad de una ruta de voz y detectar anomalías en la red. La plataforma de BTS utiliza estas métricas para diagnosticar degradaciones en tiempo real y compararlas con valores históricos por cliente y proveedor. El objetivo de este capítulo es describir la comparativa de ASR que realiza el sistema para cada ticket clasificado como degradación de tipo ASR. Se explican las consultas a la base de datos analítica Pinot, el flujo de análisis y las operaciones estadísticas que permiten decidir si existe un problema en la ruta o si el comportamiento está dentro de lo esperado.

Para determinar si la degradación es relevante se compara el ASR actual de una hora con el promedio histórico y su desviación estándar. La comparación se realiza con un intervalo de confianza de 2σ. Si el ASR actual queda por debajo (o por encima) de ese intervalo se considera una disminución (o aumento) significativa.

5.2. Fuentes de datos y consultas en Pinot

Las métricas se extraen de Pinot, el sistema de analítica en tiempo real utilizado por BTS. La tabla `bts_completions` almacena los contadores agregados de llamadas por cliente (campo `sorgSubscriptionDescription`) y proveedor (`sdstSubscriptionDescription`) en intervalos de una hora. Los campos relevantes son:

- **n_cdrs:** número de registros de llamada (Call Detail Records)
- **n_answered:** número de llamadas contestadas
- **n_rejected:** número de llamadas rechazadas
- **n_seconds:** duración total de todas las llamadas en segundos
- **avg_answered y avg_cdrs:** medias históricas de llamadas contestadas e intentadas
- **dev_answered y dev_cdrs:** desviaciones estándar históricas

Para cada ticket de tipo ASR el sistema ejecuta consultas a Pinot que integran tanto el cálculo de métricas actuales como las medias históricas y desviaciones típicas en una única query por dimensión. Esto contrasta con implementaciones anteriores que requerían dos consultas separadas por dimensión.

5.2.1. Consultas por cliente

Para el cliente se ejecutan dos tipos de consultas: una para la última hora completa y otra opcional para el momento específico de las llamadas de ejemplo.

La consulta principal calcula simultáneamente las métricas actuales y el Base histórico. Para las métricas actuales se agregan los registros de la última hora completa:

$$\mathrm{ASR}_{\mathrm{actual}} = \frac{\sum n_{\mathrm{answered}}}{\sum n_{\mathrm{cdrs}}} \qquad (\mathrm{con} \ \sum n_{\mathrm{cdrs}} > 0)$$

$$\mathrm{ALOC}_{\mathrm{actual}} = \frac{\sum n_{\mathrm{seconds}}}{\sum n_{\mathrm{answered}} \cdot 60} \qquad (\mathrm{con} \ \sum n_{\mathrm{answered}} > 0)$$

En la misma consulta se obtiene el Base histórico utilizando los campos agregados `avg_answered` y `avg_cdrs`:

$$\mathrm{ASR}_{\mathrm{Base}} = \frac{\sum \mathrm{avg\_answered}}{\sum \mathrm{avg\_cdrs}}$$

La desviación típica del ASR se calcula como:

$$\sigma_{\mathrm{ASR}} = \frac{\mathrm{AVG(dev\_answered)}}{\mathrm{AVG(dev\_cdrs)}}$$

Adicionalmente, la consulta calcula directamente la desviación del ASR actual respecto al Base mediante una expresión condicional utilizada en SQL:

$$\mathrm{asrDev} = \begin{cases} \frac{\mathrm{ASR}_{\mathrm{actual}}}{\mathrm{ASR}_{\mathrm{Base}}} - 1 & \text{si } |\mathrm{ASR}_{\mathrm{actual}} - \mathrm{ASR}_{\mathrm{Base}}| > 2\sigma_{\mathrm{ASR}} \\ 0 & \text{en caso contrario} \end{cases}$$

Cuando existen llamadas de ejemplo con timestamp específico, el sistema ejecuta una consulta adicional usando `_query_client_asr_at_call_time` que replica el mismo cálculo pero en una ventana temporal de ±30 minutos alrededor del momento de la llamada:

$$\text{ventana} = [\text{call\_datetime} - 30\text{min}, \text{call\_datetime} + 30\text{min}]$$

5.2.2. Consultas por destino

Para los destinos se ejecuta una consulta agregada por el campo `bdestination`, calculando para cada destino tanto sus métricas actuales como su Base histórico:

$$\text{ASR}_{\text{Base, destino}} = \frac{\sum \text{avg\_answered}}{\sum \text{avg\_cdrs}}$$

$$\sigma_{\text{ASR, destino}} = \frac{\text{AVG}(\text{dev\_answered})}{\text{AVG}(\text{dev\_cdrs})}$$

La consulta ordena los resultados por volumen de llamadas descendente para identificar los destinos más relevantes. Al igual que con el cliente, cuando se dispone de llamadas de ejemplo, se ejecuta `_query_destinations_asr_at_call_time` que filtra únicamente los destinos presentes en dichas llamadas dentro de la ventana temporal de ±30 minutos.

5.2.3. Consultas por proveedor

Las consultas por proveedor se ejecutan sobre el campo `sdstSubscriptionDescription`, aplicando filtros para dirección de tráfico (`direction = 1` o `direction = 0` con `sdstSubscriptionId <0`). Los cálculos son análogos:

$$\text{ASR}_{\text{Base, proveedor}} = \frac{\sum \text{avg\_answered}}{\sum \text{avg\_cdrs}}$$

$$\sigma_{\text{ASR, proveedor}} = \frac{\text{AVG}(\text{dev\_answered})}{\text{AVG}(\text{dev\_cdrs})}$$

La función `_query_providers_asr_at_call_time` permite analizar el comportamiento de proveedores específicos en el momento de las llamadas de ejemplo, filtrando por la lista de destinos y proveedores extraídos de los CDRs.

5.2.4. Detección de anomalías

Todas las consultas incluyen lógica para detectar anomalías mediante la función `asr_ope` definida en el código, que compara el valor actual con el intervalo $[\mu - 2\sigma, \mu + 2\sigma]$:

$$\text{asr\_ope}(\text{actual}, \mu, \sigma) = \begin{cases} \frac{\text{actual}}{\mu} - 1 & \text{si actual} > \mu + 2\sigma \\ \frac{\text{actual}}{\mu} - 1 & \text{si actual} < \mu - 2\sigma \\ 0 & \text{en caso contrario} \end{cases}$$

El criterio de decisión para evaluar el ASR se basa en la estadística de control de calidad. Para cada cliente, destino y proveedor se dispone de dos parámetros: la media histórica de ASR (avg_asr) y su desviación típica (dev_asr). El algoritmo calcula los límites superior e inferior como:

Si el ASR está por debajo del límite inferior o por encima, se registra y posteriormente en la contrucción del comentario con el análisis de los datos se marca en rojo o verde en caso de que se haya registrado alguno de estos icrementos o decrementos relevantes.

Todas estas consultas utilizan funciones de Pinot como `FromDateTime` para acotar ventanas temporales, operadores de agregación con `FILTER`, y la configuración `useMultistageEngine=true` para aprovechar el motor de consultas distribuido. Al tratarse de un motor analítico OLAP, Pinot devuelve los resultados en milisegundos, permitiendo manejar grandes volúmenes de datos sin penalizar el rendimiento del sistema.

5.3. Evaluación y resultados

Para evaluar el impacto del sistema propuesto se diseñó un conjunto de pruebas controladas orientadas a comparar los tiempos de respuesta y resolución de incidencias, así como la calidad percibida en las respuestas generadas. Para ello se recopilaron 43 tickets reales procedentes de la plataforma de Jira, pertenecientes a dos tipos de incidencias habituales en entornos de telefonía IP: baja tasa de conexión (Answer Seizure Rate, ASR) y rechazos de llamada (Rejection). Este volumen de muestras es representativo del flujo medio de incidencias que recibe la plataforma en escenarios operativos reales.

Para cada ticket se registraron tres marcas temporales: la hora de creación, la hora en que el equipo de CS respondió y la hora en que el sistema automático Hefestus generó la respuesta. Con esta información se calcularon los tiempos de respuesta en minutos y se obtuvo una medida comparable del rendimiento de ambos sistemas. Adicionalmente,

algunos tickets fueron evaluados mediante una encuesta de satisfacción dirigida a los usuarios finales, con el fin de valorar la utilidad práctica de las respuestas generadas.

La tabla 5.1 recoge las medias de los tiempos obtenidos para el equipo de CS y para Hefestus. Para su análisis se distinguen dos tipos de tiempo:

- **Respuesta**: intervalo entre la creación de la incidencia y la emisión de la respuesta

- **Resolución**: intervalo entre el momento en que el sistema (CS o Hefestus) detecta o ve la incidencia y genera la respuesta.

Esta distinción permite separar el coste operativo humano — marcado por la carga de trabajo y disponibilidad — del tiempo estrictamente necesario para procesar la incidencia una vez iniciada su revisión.

	Resolución CS (min)	Respuesta CS (min)	Tiempo Hefestus (min)
Rejection	2.18	12.45	1.09
ASR	1.66	5.66	1.66

Tabla 5.1: Comparativa tiempo en minutos entre CS y Hefestus

La figura 5.1 presenta un diagrama de barras que compara los tiempos medios de respuesta y resolución entre el equipo de CS y el sistema de Hefestus para los tipos de incidencia *Rejection* y *ASR*. Los resultados muestran que Hefestus mantiene un comportamiento estable en ambos casos: el tiempo de respuesta coincide con el de la resolución porque el sistema procesa automáticamente la incidencia en el mismo instante en que es creada. Esta característica reduce a un único valor el ciclo de atención del sistema.

En contraste, los tiempos registrados por CS exhiben mayor variabilidad, especialmente en el tiempo de respuesta. Esta diferencia no se debe tanto al proceso de resolución —donde CS obtiene valores medios moderadamente superiores a los de Hefestus— sino al intervalo que transcurre hasta que un agente humano detecta y comienza a analizar el ticket. Para incidencias de tipo *Rejection*, la diferencia es especialmente significativa: CS presenta un tiempo medio de resolución cercano a 12 minutos, frente al minuto aproximado que requiere Hefestus. En el caso de incidencias *ASR*, aunque la diferencia es menor, Hefestus sigue siendo más eficiente, con una ventaja media de aproximadamente 4 minutos.

El análisis conjunto de las figuras confirma estos patrones: Hefestus mantiene un tiempo constante e independiente de la carga operativa, mientras que la dispersión

de tiempos observada en CS refleja la variabilidad del trabajo manual, influida por la disponibilidad del personal, el volumen de incidencias y las consultas adicionales necesarias para analizar cada caso. En consecuencia, el sistema automatizado proporciona una respuesta inmediata y sistemática, situándose como la opción más eficiente en ambos tipos de incidencia.

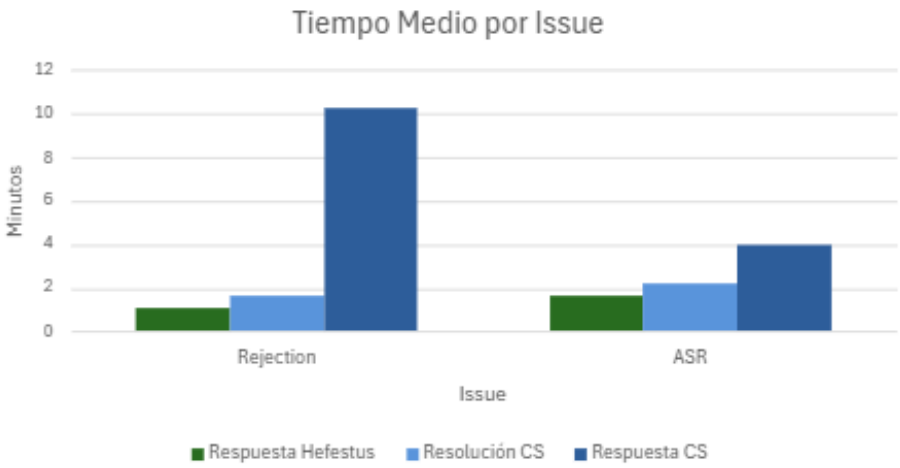

Figura 5.1: Comparativa de tiempos en minutos por tipo de incidencia

En cuanto a la evaluación de satisfacción, se recopiló la valoración de tres usuarios diferentes del departamento de CS sobre un total de 16 tickets seleccionados entre aquellos que el sistema fue capaz de resolver total o parcialmente dentro de las incidencias de tipo ASR y Rejection. La Tabla 5.2 muestra dicha distribución.

Satisfacción	**Apariciones**
1	7
2	1
3	9
4	7
5	24

Tabla 5.2: Frecuencia de aparición de cada nivel de satisfacción

A continuación, se calcularon las medias individuales para cada uno de los tres usuarios, con el fin de obtener una medida agregada del grado de satisfacción personal. La Tabla 5.3 presenta los valores medios por usuario, mientras que la Figura 5.2 ofrece una representación gráfica de estas diferencias.

Usuario	Media de satisfacción
User 1	4.00
User 2	3.94
User 3	3.81

Tabla 5.3: Satisfacción media por usuario

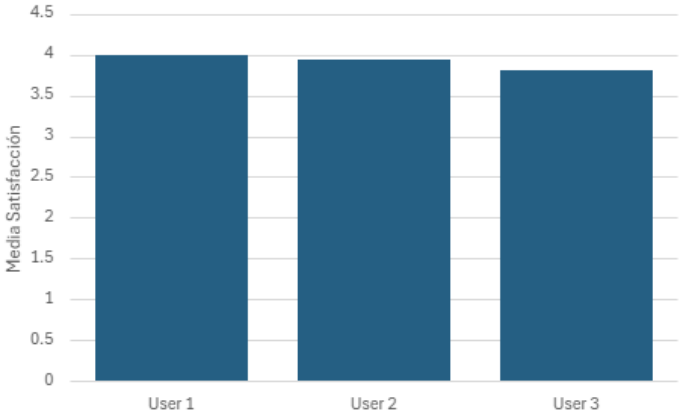

Figura 5.2: Satisfacción media por usuario

En conjunto, los resultados muestran que la mayoría de valoraciones se concentran en los niveles altos de satisfacción, lo que indica una recepción generalmente positiva del sistema. No obstante, los casos aislados de puntuaciones bajas aportan información relevante sobre limitaciones concretas, permitiendo identificar situaciones en las que el análisis automático genera respuestas demasiado extensas o imprecisas. Estas observaciones resultan esenciales para orientar mejoras posteriores del sistema.

Observando las valoraciones y preguntando al equipo de CS nos han informado que varias de las valoraciones bajas son por el tamaño de los datos que se expone en los comentarios de Jira, incidiendo en que no hace falta tanto el contexto global si no el contexto específico de esas llamadas. Otra razón por la que las valoraciones no lleguen al 5 se debe a que la información que se expone les es relevante para hacer el análisis del problema, pero faltan pasos a implementar para llegar a una automatización completa del flujo.

Para las valoraciones positivas se hace énfasis en que la información proporcionada es relevante para realizar los análisis a simple vista, lo que además mejoraría su tiempo de respuesta al ahorrarles el problema de buscar los datos por ellos mismos.

Un ejemplo de incidencia que obtuvo la satisfacción mínima sería el siguiente:

Capítulo 6

Conclusiones y Líneas Futuras

6.1. Conclusiones

A lo largo de este Trabajo Fin de Grado se ha diseñado, implementado y evaluado un sistema de automatización para la gestión de incidencias en servicios de telecomunicaciones, denominado Hefestus. El proyecto ha permitido validar la viabilidad de integrar modelos de lenguaje en un entorno operativo real, abordando desde la clasificación automática de tickets hasta el análisis de métricas técnicas de red. Los principales logros alcanzados se resumen a continuación:

Análisis del flujo de resolución de incidencias. Se ha documentado en detalle el proceso que sigue el equipo de Customer Service de BTS para gestionar las incidencias diarias. Este análisis permitió identificar los puntos críticos donde la automatización puede aportar mayor valor: la clasificación inicial del ticket, la extracción de información técnica relevante y la consulta de bases de datos internas para verificar los registros de llamada.

Preparación de un conjunto de datos representativo. Se recopiló un *dataset* de aproximadamente 60.000 tickets históricos procedentes de Jira, aplicando técnicas de limpieza para transformar el formato ADF en texto plano. El proceso de preprocesamiento redujo la media de tokens por ticket de 5.314 a 212, permitiendo aprovechar de forma efectiva la capacidad del modelo BERT. Además, se implementaron estrategias de balanceo de clases mediante *undersampling* y pesos de clase para mitigar el sesgo hacia categorías mayoritarias como SPAM.

Desarrollo y adaptación de modelos de lenguaje. Se entrenaron dos versiones del clasificador basado en BERT: una versión inicial (*Base*) y una versión mejorada (*Enhanced*). Los resultados muestran que *BERT Enhanced* ofrece un rendimiento más equilibrado a nivel de clase, alcanzando un *F1-score macro* del 80 %, frente al 70 % del modelo inicial. Paralelamente, se integró Llama 3 mediante Ollama para la extracción estructurada de información compleja (ANI, DNI, fechas, zonas horarias, causas de

liberación), tarea que resultaría inviable mediante expresiones regulares debido a la variabilidad de los textos.

Integración con Jira y bases de datos internas. Se desarrolló un *pipeline* completo que conecta con la API de Jira para la ingesta de tickets (tanto mediante consultas JQL como a través de *webhooks* en tiempo real), persiste la información en PostgreSQL y consulta Apache Pinot para la verificación de CDRs. Esta arquitectura permite que el sistema opere de forma autónoma, emulando el flujo de trabajo manual del equipo de CS.

Generación de respuestas automáticas con análisis de métricas. Para incidencias de tipo *ASR* y *Rejection*, el sistema genera comentarios automáticos que incluyen métricas actuales y comparativas respecto al histórico, tanto a nivel de cliente como de destino y proveedor. La detección de anomalías se realiza mediante un criterio estadístico basado en el intervalo de confianza de 2σ, lo que permite identificar desviaciones significativas respecto al comportamiento esperado.

Evaluación del impacto en tiempos de respuesta. Las pruebas controladas realizadas sobre 43 tickets reales demuestran que Hefestus reduce significativamente los tiempos de respuesta. Para incidencias de tipo *Rejection*, el sistema alcanza un tiempo medio de aproximadamente 1 minuto, frente a los 12 minutos del equipo humano. La encuesta de satisfacción aplicada a tres usuarios del departamento de CS arrojó una valoración media de 3,9 sobre 5, lo que indica una recepción generalmente positiva del sistema.

Cumplimiento de requisitos de privacidad y seguridad. Todo el procesamiento se realiza íntegramente en la infraestructura interna de BTS, sin transmitir datos a servicios externos. Tanto el modelo BERT como Llama 3 se ejecutan localmente, garantizando la confidencialidad de la información gestionada por el departamento de CS.

En conclusión, este trabajo demuestra que la aplicación de modelos de lenguaje en la gestión de incidencias de telecomunicaciones es técnicamente viable y aporta beneficios tangibles en términos de eficiencia, consistencia y reducción de la carga manual. El sistema desarrollado constituye una prueba de concepto funcional que la empresa podrá utilizar como base para extender la automatización al conjunto completo de degradaciones gestionadas por el equipo de CS.

6.2. Líneas Futuras

El desarrollo de Hefestus ha permitido identificar múltiples oportunidades de mejora y extensión que exceden el alcance de este Trabajo Fin de Grado. A continuación se

describen las líneas de trabajo futuro más relevantes:

Extensión a otros tipos de incidencias. Actualmente el sistema implementa *handlers* específicos para *ASR* y *Rejection*. Una línea prioritaria consiste en desarrollar módulos análogos para el resto de degradaciones (*ALOC, SPAM, Voice Quality, One Way Audio*, entre otras), cada una con sus métricas y flujos de análisis particulares. Esta extensión requiere adaptar las consultas a las bases de datos y definir criterios de detección de anomalías específicos para cada categoría.

Implementación de un sistema de retroalimentación. La incorporación de un mecanismo que permita al equipo de CS validar o corregir las predicciones del sistema facilitaría la mejora continua del modelo mediante active learning [17]. Este enfoque permite que el propio clasificador identifique los tickets en los que presenta mayor incertidumbre y solicite la anotación humana solo para esos casos, optimizando el esfuerzo de etiquetado. Las correcciones realizadas por el equipo se incorporarían automáticamente al conjunto de entrenamiento, permitiendo refinar progresivamente la precisión del modelo y adaptarlo a nuevas variaciones en los patrones de degradación.

Desarrollo de una interfaz de usuario dedicada. Aunque el sistema opera correctamente mediante CLI y *webhooks*, el desarrollo de un panel de control web permitiría una gestión más accesible, incluyendo visualización de métricas, histórico de predicciones, configuración de umbrales y gestión de reglas heurísticas.

Evaluación de la escalabilidad. A medida que aumente el volumen de tickets procesados, será necesario evaluar el rendimiento del sistema bajo carga y, en su caso, implementar mecanismos de procesamiento paralelo o distribuido. La arquitectura actual, basada en una máquina virtual dedicada, podría evolucionar hacia un despliegue containerizado que facilite la escalabilidad horizontal.

Integración con sistemas de generación de respuestas completas. Actualmente el sistema genera comentarios técnicos con métricas. Una línea avanzada consistiría en emplear modelos generativos para redactar respuestas completas al cliente, incluyendo diagnóstico, recomendaciones y pasos a seguir, siempre bajo supervisión humana antes de su envío definitivo.

En definitiva, Hefestus representa un primer paso hacia la automatización integral de la gestión de incidencias en BTS. Las líneas futuras descritas configuran una hoja de ruta para transformar esta prueba de concepto en un sistema de producción robusto, escalable y capaz de adaptarse a la evolución continua del entorno de telecomunicaciones.

Glosario de términos y siglas

ACD : *Average Call Duration*. Duración media de las llamadas contestadas (sinónimo de ALOC en muchos contextos).

ADF : *Atlassian Document Format*. Formato estructurado utilizado por Jira para representar contenido enriquecido.

ALOC : *Average Length of Call*. Duración media de las llamadas contestadas.

ANI : *Automatic Number Identification*. Identificador del número de origen de una llamada.

API : *Application Programming Interface*. Conjunto de métodos y definiciones que permiten la comunicación entre sistemas software.

ASGI : *Asynchronous Server Gateway Interface*. Estándar para servidores y aplicaciones Python asíncronas.

ASR : *Answer Seizure Ratio*. Proporción de llamadas contestadas sobre el total de intentos.

BERT : *Bidirectional Encoder Representations from Transformers*. Modelo Transformer preentrenado para comprensión del lenguaje.

BTS : *Business Telecommunication Services*. Empresa donde se desarrolla el proyecto.

CDR : *Call Detail Record*. Registro detallado generado para cada llamada en un sistema de telecomunicaciones.

CET : *Central European Time*. Zona horaria UTC+1 utilizada en Europa central.

CLI : *Calling Line Identification*. Identificación del número telefónico de origen visualizado en destino.

CMDB : *Configuration Management Database*. Base de datos que almacena los elementos de configuración de un sistema TI.

DDoS : *Distributed Denial of Service*. Ataque que busca denegar servicio saturando un sistema mediante múltiples fuentes.

DNI : *Dialed Number Identification*. Número marcado por el usuario, correspondiente al destino de la llamada.

FAX : *Facsimile*. Servicio de transmisión de documentos mediante señalización

telefónica.

Fake DLR : *Fake Delivery Report*. Informe de entrega falsificado o incorrecto en servicios de mensajería.

GMT : *Greenwich Mean Time*. Tiempo medio del meridiano de Greenwich, equivalente práctico a UTC.

GSMA : *Global System for Mobile Communications Association*. Asociación internacional de operadores y empresas del sector móvil.

HTTP : *HyperText Transfer Protocol*. Protocolo principal de comunicación en la web.

IA : Inteligencia Artificial. Disciplina que desarrolla sistemas capaces de realizar tareas que requieren capacidades cognitivas humanas.

JQL : *Jira Query Language*. Lenguaje de consultas utilizado para filtrar y recuperar incidencias en Jira.

JSON : *JavaScript Object Notation*. Formato ligero de intercambio de datos basado en pares clave–valor.

LLM : *Large Language Models*. Modelos de lenguaje de gran escala basados en arquitecturas Transformer.

ML : *Machine Learning*. Conjunto de técnicas de aprendizaje automático basadas en datos.

NLP : *Natural Language Processing*. Área de la IA centrada en la interpretación y generación de lenguaje natural.

OLAP : *On-Line Analytical Processing*. Tecnología orientada al análisis multidimensional de datos para consultas agregadas y analíticas.

PDD : *Post-Dial Delay*. Tiempo que transcurre entre marcar un número y escuchar el primer tono de respuesta.

RTP : *Real-time Transport Protocol*. Protocolo para la transmisión en tiempo real de audio y vídeo.

SIP : *Session Initiation Protocol*. Protocolo de señalización utilizado para establecer, modificar y finalizar sesiones VoIP.

SQL : *Structured Query Language*. Lenguaje estándar para la gestión y consulta de bases de datos relacionales.

SMS : *Short Message Service*. Servicio estándar de mensajería de texto.

SPAM : Tráfico automatizado o no deseado que suele involucrar llamadas masivas, *robocalls* o intentos de fraude.

SSL/TLS : *Secure Sockets Layer / Transport Layer Security*. Protocolos criptográficos para comunicaciones seguras.

TI : Tecnologías de la Información. Conjunto de recursos tecnológicos destinados a gestionar y procesar información.

UTC : *Coordinated Universal Time*. Estándar internacional de referencia para zonas horarias.

VM : *Virtual Machine*. Entorno virtualizado que emula un sistema operativo completo.

YAML : *YAML Ain't Markup Language*. Formato de serialización de datos legible y utilizado en configuraciones.

Capítulo 7

Bibliografía

[1] Rajeev Gupta, K Hima Prasad, and Mukesh Mohania. Automating ITSM incident management process. In *2008 International Conference on Autonomic Computing*, pages 141–150, 2008.

[2] Sara B. Elagib, Aisha-Hassan A. Hashim, and R. F. Olanrewaju. Cdr analysis using big data technology. In *International Conference on Computing, Control, Networking, Electronics and Embedded Systems Engineering (ICCNEEE)*. IEEE, 2015.

[3] Tommi Koistinen. Protocol overview: Rtp and rtcp. *Documento técnico — Nokia Telecommunications*, 1999.

[4] J. A. Carballar Falcón. *VoIP: La telefonía de Internet*. Ediciones Paraninfo, 2007.

[5] Andrés Bertin, Julio Meléndez, and Roberto Leiva. Voip quality and service metrics — a comparative study of asr, aloc and other indicators. In *Proceedings of the International Conference on Telecommunications and Multimedia (TEMU 2014)*. IEEE, 2014.

[6] Ram Dantu and Prakash Kolan. Detecting spam in voip networks. In *Proceedings of the Steps to Reducing Unwanted Traffic on the Internet Workshop (SRUTI '05)*. USENIX Association, 2005.

[7] M. I. Iskandar and otros. Impact analysis of insufficient destination resources on call termination in voip systems. *Procedia Computer Science*, 2024.

[8] Daniel Jurafsky and James H. Martin. *Speech and Language Processing: An Introduction to Natural Language Processing, Computational Linguistics, and Speech Recognition with Language Models*. Online manuscript, 3rd edition, 2025.

[9] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. Bert: Pre-training of deep bidirectional transformers for language understanding. In *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, Minneapolis, Minnesota, 2019. Association for Computational Linguistics.

[10] Aaron Grattafiori, Abhimanyu Dubey, Abhinav Jauhri, Abhinav Pandey, Abhishek Kadian, and *et al.* The llama 3 herd of models. Technical report, Meta AI, 2024.

[11] D. Vake and otros. A secure, scalable framework for distributed ollama inference. *Future Generation Computer Systems*, 2025.

[12] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Łukasz Kaiser, and Illia Polosukhin. Attention is all you need. In *Advances in Neural Information Processing Systems*. NeurIPS, 2017.

[13] Taojun Hu and Xiao-Hua Zhou. Unveiling LLM evaluation focused on metrics: Challenges and solutions, 2024.

[14] Sebastián Ramírez. *FastAPI documentation — Quickstart and server deployment with Uvicorn*. FastAPI, 2025.

[15] Fengrui Liu, Xiao He, Tieying Zhang, Jianjun Chen, Yi Li, Lihua Yi, Haipeng Zhang, Gang Wu, and Rui Shi. Tickit: Leveraging large language models for automated ticket escalation. In *Proceedings of the (FSE Companion '25)*, pages 343–354, 2025.

[16] GSMA and HuggingFace. Open-telco LLM benchmark: Evaluating ai for the telecom industry, 2024.

[17] Burr Settles. Active learning literature survey. *Computer Sciences Technical Report 1648*, 2009.

[18] Robert Graves. *The Greek Myths*. Penguin Books, London, 1955. Descripción de Hefesto, su papel como dios de la forja, la técnica y los autómatas divinos.

Lista de Figuras

Lista de Tablas

Anexos

Anexos A

Ejemplos de tickets

A.1. Ticket original (formato ADF de Jira)

’type’: ’doc’, ’version’: 1, ’content’: [’type’: ’paragraph’, ’content’: [’type’: ’text’, ’text’: ’Greetings,’], ’type’: ’p -aragraph’, ’content’: [’type’: ’text’, ’text’: ’Verizon Business is requesting to open up a ticket on the following issue:’], ’type’: ’paragraph’, ’content’: [’type’: ’text’, ’text’: ’ISSUE: Low ASR/Machine Generated traffic issue towards Guatemala,Telgua’], ’type’: ’paragraph’, ’content’: [’type’: ’text’, ’text’: ’CALL EXAMPLE:’], ’type’: ’paragraph’, ’content’: [’type’: ’text’, ’text’: ’Start Date/Time (GMT)’, ’type’: ’hardBreak’, ’type’: ’text’, ’text’ : ’Calling Number’, ’type’: ’hardBreak’, ’type’: ’text’, ’text’: ’Called Number’, ’type’: ’hardBreak’, ’type’: ’tex t’, ’text’: ’Final Response’], ’type’: ’paragraph’, ’content’: [’type’: ’text’, ’text’: ’2025-08-26 08:00:00.271’, ’ty pe’: ’hardBreak’, ’type’: ’text’, ’text’: ’0050223560610’, ’type’: ’hardBreak’, ’type’: ’text’, ’text’: ’0050279464327’, ’type’: ’hardBreak’, ’type’: ’text’, ’text’: ’503’], ’type’: ’paragraph’, ’content’: [’type’: ’text’, ’text’: ’2025-08-26 07:59:56.818’, ’type’: ’hardBreak’, ’type’: ’text’, ’text’: ’0050223560610’, ’type’: ’hardBreak’, ’type’: ’text’, ’text’: ’0050279464327’, ’type’: ’hardBreak’, ’type’: ’text’, ’text’: ’503’], ’type’: ’paragraph’, ’content’: [’type’: ’text’, ’text’: ’2025-08-26 07:59:55.627’, ’type’: ’hardBreak’, ’type’: ’text’, ’text’: ’0050223560610’, ’type’: ’hardBreak’, ’type’: ’text’, ’text’: ’0050279464327’, ’type’: ’hardBreak’, ’type’: ’text’, ’text’: ’503’], ’type’: ’paragraph’, ’content’: [’type’: ’text’, ’text’: ’2025-08-26 07:59:52.381’, ’type’: ’hardBreak’, ’type’:

’text’, ’text’: ’0050223560610’, ’type’: ’hardBreak’, ’type’: ’text’, ’text’: ’0050279464327’, ’type’: ’hardBreak’, ’type’: ’text’, ’text’: ’503’], ’type’: ’paragraph’, ’content’: [’type’: ’text’, ’text’: ’2025-08-26 07:59:48.799’, ’type’: ’hardBreak’, ’type’: ’text’, ’text’: ’0050223560610’, ’type’: ’hardBreak’, ’type’: ’text’, ’text’: ’0050279464327’, ’type’: ’hardBreak’, ’type’: ’text’, ’text’: ’503’], ’type’: ’paragraph’, ’content’: [’type’: ’text’, ’text’: ’Originating IP : 65.198.234.193’, ’type’: ’hardBreak’, ’type’: ’text’, ’text’: ’65.198.234.194’, ’type’: ’hardBreak’, ’type’: ’text’, ’text’: ’65.198.234.195’, ’type’: ’hardBreak’, ’type’: ’text’, ’text’: ’65.198.234.196’], ’type’: ’paragraph’, ’content’: [’type’: ’text’, ’text’: ’Terminating IP : 79.170.68.166’], ’type’: ’paragraph’, ’content’: [’type’: ’text’, ’text’: ’Verizon Business ticket number: 2025082815799’, ’type’: ’hardBreak’, ’type’: ’text’, ’text’: ’Verizon Business Callback Number: 1-800-285-7253’, ’type’: ’hardBreak’, ’type’: ’text’, ’text’: ’Verizon Business Hours: 24x7’]]

A.2. Ticket formateado

```
Greetings,
Verizon Business is requesting to open up a ticket on the
following issue:
ISSUE: Low ASR/Machine Generated traffic issue towards
Guatemala, Telgua
CALL EXAMPLE:
Start Date/Time (GMT)
Calling Number
Called Number
Final Response
2025-08-26 08:00:00.271
0050223560610
0050279464327
503
2025-08-26 07:59:56.818
0050223560610
0050279464327
503
2025-08-26 07:59:55.627
0050223560610
0050279464327
503
2025-08-26 07:59:52.381
0050223560610
0050279464327
```

```
503
2025-08-26 07:59:48.799
0050223560610
0050279464327
503
Originating IP : 65.198.234.193
65.198.234.194
65.198.234.195
65.198.234.196
Terminating IP : 79.170.68.166
Verizon Business ticket number: 2025082815799
Verizon Business Callback Number: 1-800-285-7253
Verizon Business Hours: 24x7...
```

Anexos B

Tablas del clasificador de incidencias

B.1. Tabla métricas por clase BERT_Base

Clase	Precisión	Recall	F1-score	Soporte
ALOC	0.879	0.903	0.891	299.0
ASR	0.855	0.860	0.857	786.0
AntiFraud	0.899	0.983	0.939	172.0
CLI	0.923	0.837	0.878	172.0
CLI Spoofed	0.775	0.721	0.747	43.0
Call Stretching	0.868	0.807	0.836	57.0
Call drop	0.500	0.519	0.509	52.0
DTMF	1.000	0.783	0.878	23.0
Dead Air	0.692	0.535	0.603	101.0
FAS	0.783	0.825	0.804	573.0
FAX	0.986	0.965	0.975	142.0
Fake DLR	0.000	0.000	0.000	4.0
IVR	0.814	0.643	0.719	129.0
LOOP	0.918	0.672	0.776	67.0
Long PDD	0.841	0.784	0.811	74.0
Misrouted calls	0.720	0.692	0.706	26.0
NER	0.583	0.350	0.438	20.0
Notification	0.810	0.685	0.742	124.0
One Way Audio	0.870	0.513	0.645	39.0
Others	0.533	0.190	0.281	126.0
Packet Loss	0.688	0.611	0.647	18.0
Rejection	0.801	0.875	0.836	1258.0
Ringk Back Issue	0.688	0.805	0.742	41.0
Roaming	0.657	0.611	0.633	113.0
Robocall	0.500	1.000	0.667	2.0
SPAM	0.991	0.994	0.993	13125.0
Undelivered	0.815	0.946	0.876	56.0
Voice Quality	0.640	0.679	0.659	84.0
Wangiri	0.985	0.992	0.989	262.0
Whatsapp/Viber	0.833	0.714	0.769	7.0
accuracy	0.946	0.946	0.946	0.946040566824118
macro avg	0.762	0.717	0.728	17995.0
weighted avg	0.944	0.946	0.944	17995.0

B.2. Tabla métricas por clase BERT_Enhanced

Clase	Precisión	Recall	F1-score	Soporte
ALOC	0.957	0.890	0.922	299.0
ASR	0.896	0.907	0.901	600.0
AntiFraud	0.927	0.953	0.940	172.0
CLI	0.871	0.901	0.886	172.0
CLI Spoofed	0.902	0.860	0.881	43.0
Call Stretching	0.750	0.737	0.743	57.0
Call drop	0.800	0.692	0.742	52.0
DTMF	1.000	1.000	1.000	23.0
Dead Air	0.618	0.667	0.642	102.0
FAS	0.894	0.810	0.850	573.0
FAX	1.000	0.986	0.993	142.0
Fake DLR	0.000	0.000	0.000	4.0
IVR	0.728	0.767	0.747	129.0
LOOP	0.887	0.821	0.853	67.0
Long PDD	0.767	0.892	0.825	74.0
Misrouted calls	0.583	0.808	0.677	26.0
NER	0.419	0.650	0.510	20.0
Notification	0.786	0.742	0.763	124.0
One Way Audio	0.706	0.923	0.800	39.0
Others	0.600	0.643	0.621	126.0
Packet Loss	0.882	0.833	0.857	18.0
Rejection	0.826	0.825	0.826	600.0
Ringk Back Issue	0.750	0.732	0.741	41.0
Roaming	0.798	0.770	0.784	113.0
Robocall	0.500	0.500	0.500	2.0
SPAM	0.978	0.972	0.975	600.0
Undelivered	0.753	0.982	0.853	56.0
Voice Quality	0.734	0.821	0.775	84.0
Wangiri	0.992	0.985	0.989	262.0
Whatsapp/Viber	0.556	0.714	0.625	7.0
accuracy	0.864	0.864	0.864	0.8644910309055543
macro avg	0.762	0.793	0.774	4627.0
weighted avg	0.869	0.864	0.865	4627.0

Tabla B.2: Métricas por clase para BERT_Enhanced

B.3. Tabla métricas REGLAS

Clase	Precisión	Recall	F1-score	Soporte
ALOC	1.000	0.431	0.603	299.0
ASR	0.955	0.713	0.817	600.0
AntiFraud	0.000	0.000	0.000	172.0
CLI	0.800	0.256	0.388	172.0
CLI Spoofed	0.955	0.488	0.646	43.0
Call Stretching	1.000	0.158	0.273	57.0
Call drop	0.778	0.269	0.400	52.0
DTMF	0.880	0.957	0.917	23.0
Dead Air	0.510	0.735	0.602	102.0
FAS	0.877	0.613	0.721	573.0
FAX	0.613	0.993	0.758	142.0
Fake DLR	0.600	0.750	0.667	4.0
IVR	0.600	0.651	0.625	129.0
LOOP	0.700	0.731	0.715	67.0
Long PDD	0.721	0.838	0.775	74.0
Misrouted calls	0.667	0.615	0.640	26.0
NER	0.029	0.750	0.056	20.0
Notification	0.508	0.250	0.335	124.0
One Way Audio	0.690	0.744	0.716	39.0
Others	0.049	0.357	0.087	126.0
Packet Loss	0.750	0.833	0.789	18.0
Rejection	0.772	0.215	0.336	600.0
Ringk Back Issue	0.500	0.220	0.305	41.0
Roaming	0.580	0.673	0.623	113.0
Robocall	0.500	1.000	0.667	2.0
SPAM	0.980	0.965	0.972	600.0
Undelivered	0.889	0.286	0.432	56.0
Voice Quality	0.431	0.333	0.376	84.0
Wangiri	0.985	0.985	0.985	262.0
Whatsapp/Viber	0.500	0.857	0.632	7.0
accuracy	0.581	0.581	0.581	0.5805057272530797
macro avg	0.661	0.589	0.562	4627.0
weighted avg	0.779	0.581	0.625	4627.0

Tabla B.3: Métricas por clase para REGLAS

B.4. Tabla con el mejor modelo por clase

Clase	Mejor modelo	F1-score
ALOC	BERT_Enhanced	0.922
ASR	BERT_Enhanced	0.900
AntiFraud	BERT_Base	0.939
CLI	BERT_Enhanced	0.890
CLI Spoofed	BERT_Enhanced	0.886
Call Stretching	BERT_Enhanced	0.780
Call drop	BERT_Enhanced	0.747
DTMF	BERT_Enhanced	1.000
Dead Air	BERT_Enhanced	0.679
FAS	BERT_Enhanced	0.861
FAX	BERT_Enhanced	0.993
Fake DLR	Reglas	0.667
IVR	BERT_Enhanced	0.724
LOOP	BERT_Enhanced	0.848
Long PDD	BERT_Enhanced	0.833
Misrouted calls	BERT_Enhanced	0.735
NER	BERT_Enhanced	0.578
Notification	BERT_Enhanced	0.776
One Way Audio	BERT_Enhanced	0.835
Others	BERT_Enhanced	0.639
Packet Loss	Reglas	0.789
Rejection	BERT_Base	0.836
Ringk Back Issue	BERT_Enhanced	0.785
Roaming	BERT_Enhanced	0.783
Robocall	BERT_Base	0.667
SPAM	BERT_Base	0.993
Undelivered	BERT_Enhanced	0.813
Voice Quality	BERT_Enhanced	0.768
Wangiri	BERT_Base	0.989
Whatsapp/Viber	BERT_Base	0.769

Tabla B.4: Mejor modelo por clase según F1-score

B.5. Tablas comparación accuracy, f1-score, precisión y recall

Modelo	Accuracy	F1_weighted	Precision_weighted	Recall_weighted
BERT_Base	0.946	0.944	0.944	0.946
BERT_Enhanced	0.864	0.865	0.869	0.864
REGLAS	0.581	0.625	0.779	0.581

Tabla B.5: Comparación conjunta de Accuracy, F1, Precision y Recall weighted de test desbalanceados

Modelo	Precision_macro	Recall_macro	F1_macro
BERT_Base	0.762	0.717	0.728
BERT_Enhanced	0.762	0.793	0.774
REGLAS	0.661	0.589	0.562

Tabla B.6: Comparación conjunta de Accuracy, F1, Precision y Recall macro de test desbalanceados

Modelo	Accuracy	F1_weighted	Precision_weighted	Recall_weighted
BERT_Base	0.797	0.797	0.800	0.798
BERT_Enhanced	0.864	0.865	0.869	0.864
REGLAS	0.581	0.625	0.779	0.581

Tabla B.7: Comparación conjunta de Accuracy, F1, Precision y Recall weighted test balanceado

Modelo	Precision_macro	Recall_macro	F1_macro
BERT_Base	0.718	0.690	0.700
BERT_Enhanced	0.762	0.793	0.774
REGLAS	0.661	0.589	0.562

Tabla B.8: Comparación conjunta de Accuracy, F1, Precision y Recall macro test balanceado

B.6. Tabla comparación F1 por clase para los modelos propuestos

Clase	BERT_BASE	BERT_ENHANCED	RULES
ALOC	0.891	0.922	0.603
ASR	0.857	0.901	0.817
AntiFraud	0.939	0.940	0.000
CLI	0.878	0.886	0.388
CLI Spoofed	0.747	0.881	0.646
Call Stretching	0.836	0.743	0.273
Call drop	0.509	0.742	0.400
DTMF	0.878	1.000	0.917
Dead Air	0.603	0.642	0.602
FAS	0.804	0.850	0.721
FAX	0.975	0.993	0.758
Fake DLR	0.000	0.000	0.667
IVR	0.719	0.747	0.625
LOOP	0.776	0.853	0.715
Long PDD	0.811	0.825	0.775
Misrouted calls	0.706	0.677	0.640
NER	0.438	0.510	0.056
Notification	0.742	0.763	0.335
One Way Audio	0.645	0.800	0.716
Others	0.281	0.621	0.087
Packet Loss	0.647	0.857	0.789
Rejection	0.836	0.826	0.336
Ringk Back Issue	0.742	0.741	0.305
Roaming	0.633	0.784	0.623
Robocall	0.667	0.500	0.667
SPAM	0.993	0.975	0.972
Undelivered	0.876	0.853	0.432
Voice Quality	0.659	0.775	0.376
Wangiri	0.989	0.989	0.985
Whatsapp/Viber	0.769	0.625	0.632

Tabla B.9: Métricas F1 por clase y modelo

B.7. Ejemplo del desbalanceo de clases en el dataset inicial

Clase	frecuencia
SPAM	42956
FAS	27485
ASR	25139
CLI	19564
Rejection	12816
LOOP	6171
Notification	5700
ALOC	5667
IVR	3814
Dead Air	3530
Undelivered	2279
Roaming	2082
Wangiri	2013
Long PDD	1833
FAX	1680
NER	1569
Fake DLR	1289
One Way Audio	1140
Packet Loss	834
Call drop	816
Voice Quality	776
DTMF	739
Others	673
Call Stretching	448
Robocall	301
Misrouted calls	246
AntiFraud	151
Ringk Back Issue	83
CLI Spoofed	64
Whatsapp/Viber	6

Tabla B.10: Frecuencia de clases en el dataset inicial desbalanceado

Anexos C

Hefestus: origen del nombre

El nombre procede de *Hefesto*, divinidad de la mitología griega asociada a la forja, la técnica y la artesanía especializada. En la tradición helénica, *Hefesto* destaca por su capacidad para diseñar y construir artefactos complejos que resolvían problemas para el resto de los dioses, desde autómatas metálicos hasta armas de precisión. Esta representación simbólica encaja con el propósito del sistema, orientado a "forjar" respuestas, análisis y procesos automatizados que agilicen la resolución de incidencias en un entorno tecnológico altamente especializado.

La elección del nombre se alinea también con la convención de nomenclatura interna de la empresa, que asigna a sus servicios y plataformas nombres de origen mitológico, especialmente griego. Entre ellos destacan:

Talos, que actúa como *proxy SIP de entrada* del switch. Es el componente encargado de validar las llamadas entrantes de los clientes, consultar las bases de datos de *routing* y aplicar la lógica inicial de preprocesado antes de enviarlas a los *B2BUA*. Talos constituye, por tanto, el primer nivel de decisión técnica dentro del flujo de señalización.

Theseus, el *proxy SIP de salida*, recibe las llamadas procesadas por los *B2BUA* o directamente por Talos y se encarga de aplicarlas a los proveedores.

Ariadna, el servicio responsable de resolver el *routing* final de cada llamada. Recibe consultas desde Talos o Minos y determina la ruta óptima mediante información almacenada en distintas instancias de Redis (provisioning, portabilidad u otros servicios de consulta como RDOS).

En este ecosistema, *Hefestus* aporta coherencia conceptual y refuerza su identidad como herramienta diseñada para crear y articular procesos complejos que asisten al resto de servicios de la infraestructura, del mismo modo que *Hefesto* forjaba mecanismos y tecnologías al servicio de la comunidad divina [18].

Anexos D

Prompt para extracción de llamadas

D.1. System Prompt

```
Eres un extractor de datos determinista. Tu salida debe ser SIEMPRE un JSON
↪  válido y sintácticamente correcto.
NO EXPLIQUES NADA. NO AÑADAS TEXTO.
Debes:
1. Leer el texto del ticket.
2. Extraer el destino ("destination") y la lista de llamadas ("calls").
3. Cumplir estrictamente las reglas indicadas. Si violas alguna, tu
↪  respuesta será rechazada.
```

D.2. User Template

```
Extrae la información de llamadas y destino siguiendo exactamente el formato
↪  siguiente.
Devuelve solo este objeto JSON (nada más, sin texto ni comentarios):

{
  "destination": "string or 'Unknown'",
  "calls": [
    {
      "ANI": "string or null",
      "DNI": "string or null",
      "date": "YYYY-MM-DD or null",
      "time_answered": "HH:MM:SS or null",
      "time_finished": "HH:MM:SS or null",
      "duration": "HH:MM:SS or null",
      "release_cause": integer or null,
      "timezone": "string or null",
      "source_hint": "string or null"
    }
  ]
}
```

```
### REGLAS ESTRICTAS

1) Validación de números:
- Un número válido contiene SOLO dígitos o un '+' inicial. Ej: +34911223344,
↪  4740969372.
- Si el valor contiene letras (como "SIP", "BTS"), guiones o cualquier
↪  símbolo, descártalo.
- NO mantengas "4741378881SIP": debe convertirse en null o descartarse por
↪  completo.
- Si el número tiene menos de 8 dígitos después de eliminar símbolos,
↪  DESCÁRTALO.
- No incluyas números de contacto.
- Si pone "Sample Numbers", estos números son ANIs válidos y DNIs nulos.

2) Generación de llamadas:
- Solo crea una entrada si ANI y DNI son válidos simultáneamente.
- Si alguno no lo es, no incluyas esa llamada.
- Si no hay ninguna llamada válida, "calls": [].

3) Campos nulos:
- Usa null (sin comillas) para cualquier valor desconocido.

4) Destination:
- Usa palabras que indiquen país/red (Norway, Vodafone). Si no existe:
↪  "Unknown".

5) Fechas y horas:
- Si no aparece zona horaria explícita → timezone = null.
- Convierte fechas a YYYY-MM-DD.
- Extrae literal cualquier zona horaria presente.
- Usa la más cercana a la sección de ejemplos.
- Si hay hora general → úsala. Si hay específica → prevalece.
- Si no hay fecha u hora → null.

6) Salida:
- Devuelve solo el JSON.
- Máximo 5 llamadas.

7) Ignora completamente números de firma o contacto.

### Texto del ticket:
<<<
{ticket_text}
>>>
```