UAB

Universitat Autònoma de Barcelona

Aplicación de la Inteligencia Artificial en la Gestión de la Aeronavegabilidad Continuada de las Aeronaves

Memoria del Trabajo Fin de Grado en Gestión Aeronáutica

realizado por

Sergio Alcaide Camacho

y dirigido por

Laura Calvet Liñán

Escuela de Ingeniería

Sabadell, Junio de 2025

La abajo firmante, Laura Calvet Liñán director/a del Trabajo de Fin de Grado, profesora de la Escuela de Ingeniería de la UAB,

CERTIFICA:

Que el trabajo al que corresponde la presente memoria ha sido realizado bajo su dirección por

Sergio Alcaide Camacho

Y para que conste firma la presente en Sabadell, Junio de 2025

Firmado: Laura Calvet Liñán

La abajo firmante, Sara Gracia Abilla, tutora del Trabajo de Fin de Grado, designada por Heliswiss Ibérica,

CERTIFICA:

Que el trabajo al que corresponde la presente memoria
ha sido realizado bajo su supervisión por

Sergio Alcaide Camacho

Y para que conste firma la presente en Sabadell, Junio de 2025

Firmado: Sara Gracia Abilla

Agradecimientos

En primer lugar, agradecer a Laura Calvet Liñán su apoyo para la realización de este TFG pese a las dificultades surgidas en mi caso particular. Por su paciencia y por sus palabras de ánimo en todo momento.

Del mismo modo, agradecer a mi hermana Beatriz Alcaide su ayuda con el desarrollo del código del modelo en el lenguaje de programación Python, ya que dicho lenguaje era completamente desconocido para mí a la hora de empezar con este TFG.

Por último, agradecer a Florencia Spaini, técnico de la oficina CAMO de Heliswiss, su tiempo para explicarme el proceso de análisis de Directivas que se lleva a cabo en la organización CAMO de Heliswiss, así como su ayuda a la hora de comparar los resultados obtenidos a través del modelo con los datos almacenados sobre aplicabilidad en Heliswiss.

Título del Trabajo de Fin de Grado: Aplicación de la Inteligencia Artificial en la Gestión de la Aeronavegabilidad Continuada de las Aeronaves.	
Autor[a]: Sergio Alcaide Camacho	**Fecha:** 26 de Junio de 2025
Tutor[a]/[es]: Laura Calvet Liñán; Sara Gracia Abilla	
Titulación: Gestión Aeronáutica	
Conceptos Clave: Inteligencia Artificial, Procesamiento Lenguaje Natural, Gestión Aeronavegabilidad Continuada, Directiva Aeronavegabilidad	

Resumen del Trabajo de Fin de Grado:

El presente trabajo, titulado "Aplicación de la Inteligencia Artificial en la Gestión de la Aeronavegabilidad Continuada de las Aeronaves", está centrado en el uso de aplicaciones procedentes de la Inteligencia Artificial para asistir al personal técnico en la Gestión de la Aeronavegabilidad de una aeronave.

El principal objetivo del proyecto es desarrollar un modelo práctico funcional para asistir al personal técnico responsable de la Gestión de la Aeronavegabilidad en el estudio de aplicabilidad de Directivas de Aeronavegabilidad. Se pretende analizar la forma en que se lleva a cabo este proceso para desarrollar un modelo que dé soporte al personal técnico y permita minimizar errores.

Tal y como se ha indicado, en el desarrollo del modelo se quieren integrar herramientas procedentes del desarrollo de la Inteligencia Artificial, especialmente de herramientas para el procesamiento de lenguaje natural que permitan extraer información concisa de textos codificados en lenguaje natural.

Como el principal objetivo del TFG es el desarrollo de un modelo práctico y funcional, una vez finalizado el modelo, éste se pondrá a prueba con el análisis de varias Directivas de Aeronavegabilidad. De forma adicional, el modelo se llevará a un escenario real donde se podrá comprobar si podría resultar válido de incorporarse a la organización CAMO.

Índice

Acrónimos utilizados

AD: *Airworthiness Directive* (Directiva Aeronavegabilidad)

ASB: *Alert Service Bulletin* (Boletín de Servicio de Alerta)

BBDD: Base de Datos

CAMO: *Continuing Airworthiness Management Organisation* (Organización de mantenimiento de la Aeronavegabilidad Continuada)

EASA: *European Aviation Safety Agency* (Agencia Europea Seguridad Aérea)

GMA: Gestión Mantenimiento Aeronaves. Software de gestión para el mantenimiento de aeronaves implementado en la empresa Heliswiss.

IA: Inteligencia Artificial

MRO: *Maintenance, Repair and Operations* (Mantenimiento, Reparación y Operaciones)

NLP: *Natural Language Processing* (Procesamiento de Lenguaje Natural)

NLTK: *Natural Language Toolkit*

OCR: *Optical Character Recognition* (Reconocimiento Óptico de Carácteres)

PDF: *Portable Document Format* (Formato de Documento Portátil)

P/N*: Part Number* (número de identificación de un producto o componente)

POS-tagging: *Part-of-speech tagging* (etiquetado de las categorías gramaticales)

Términos utilizados

ATA 100: clasificación para mantenimiento de los sistemas individuales de referencia común para todas las aeronaves.

Desempeño humano: se refiere a las capacidades y limitaciones humanas que tienen un impacto en la seguridad y eficiencia de las actividades aeronáuticas.

Factores Humanos: es cualquier cosa que afecte el desempeño humano, lo que significa principios que aplican a las actividades aeronáuticas, y que buscan una interfaz segura entre el ser humano y otros componentes del sistema considerando de forma apropiada el desempeño humano.

Lenguaje Natural: lenguaje usado para la comunicación por los seres humanos.

Organización Parte-145: Organización que cumple los requisitos para el mantenimiento de aeronaves y sus componentes.

Procesamiento Lenguaje Natural: subcampo de la ciencia computacional y especialmente de la inteligencia artificial, que proporciona a los ordenadores la capacidad de procesar datos codificados en lenguaje natural.

Token (clase Token): una unidad de texto (palabra).[11]

Tokenizar: dividir un texto en sus partes constituyentes.[11]

1. Introducción

1.1 Contexto y Conceptos Básicos

Para entender el contexto en el que se desarrolla este trabajo de Fin de Grado (TFG), es necesario definir qué es la Gestión de la Aeronavegabilidad Continuada de las Aeronaves.

Para garantizar la operación segura de las aeronaves en todos los Estados Miembros de la Unión Europea, la Agencia de Seguridad Aérea Europea (*EASA* por sus siglas en inglés - *European Aviation Safety Agency*) elabora distintas Reglamentaciones que regulan y desarrollan aspectos concretos de la operación y navegación aérea.

En concreto, la *Comission Regulation (EU) No 1321/2014* del 26 de Noviembre 2014 (conocido también como *Continuing Airworthiness Regulation*)[1] establece los requisitos técnicos y procedimientos administrativos comunes para garantizar la **Aeronavegabilidad Continuada** de una aeronave**,** incluyendo cualquier componente para su instalación, que esté registrada en un Estado Miembro de la Unión.

Figura 1: Portada de las Easy Access Rules for Continuing Airworthiness elaborada por EASA. Fuente: [1]

La *Continuing Airworthiness Regulation* define, en su artículo 2, la **aeronavegabilidad continuada** como "todos los procesos que garantizan que, en cualquier momento de su vida operativa, la aeronave cumpla con los requisitos de aeronavegabilidad vigentes y esté en condiciones de operar con seguridad".

La *Regulación* se divide en un total de 8 anexos, interrelacionados entre sí, donde cada anexo desarrolla un proceso de la aeronavegabilidad continuada. Este TFG se desarrolla principalmente en el contexto operativo y normativo de los siguientes dos anexos:

- El Anexo I (Parte-M) "*establece las medidas que se han de llevar a cabo para asegurar que la aeronavegabilidad de una aeronave es mantenida, incluyendo su mantenimiento*."
- El Anexo Vc (Parte *Continuing Airworthiness Management Organisation,* CAMO) "*establece los requisitos que deben cumplirse por una organización para aplicar a la emisión o*

renovación de un certificado para la gestión de la aeronavegabilidad continuada de una aeronave y de sus componentes."

Por tanto, la Gestión de la Aeronavegabilidad Continuada de las Aeronaves es la actividad que realizan las organizaciones CAMO para garantizar el cumplimiento continuo de las medidas indicadas en el Anexo I (Parte-M) de la Regulación en las aeronaves que gestiona bajo su aprobación.

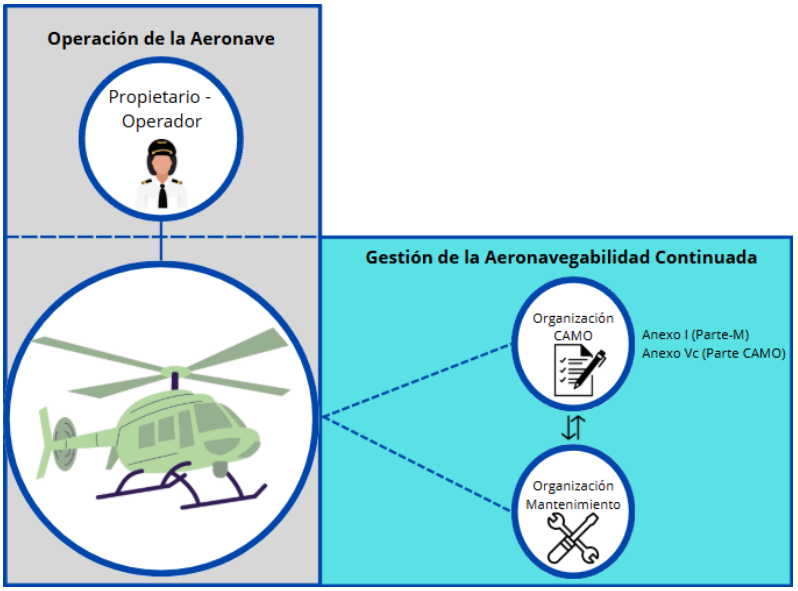


Figura 2: Esquema Gestión Aeronavegabilidad Continuada de las Aeronaves.
Fuente: Elaboración propia

A modo ilustrativo de lo indicado en el párrafo anterior, el punto normativo M.A.301 - Tareas de Aeronavegabilidad Continuada - indica que la aeronavegabilidad continuada de una aeronave debe asegurarse, entre otros, a través de:

- ✔ el cumplimiento de todo el mantenimiento de acuerdo al Programa de Mantenimiento;
- ✔ el cumplimiento de cualquier Directiva de Aeronavegabilidad (AD por sus siglas en inglés) y de medidas requeridas por la Autoridad Competente como acción inmediata ante un problema de seguridad.

Es la organización CAMO la que, a través de sus propios procesos y recursos (humanos, tecnológicos, etc.), debe garantizar el cumplimiento de los requisitos y tareas indicadas en la norma.

1.2 Motivación

Uno de los motivos principales para la realización de este TFG procede del desarrollo de mi actividad profesional como Ingeniero de Control de Conformidad y Auditor en la empresa Heliswiss Ibérica (https://heliswiss.es/), que me permite auditar y observar de forma directa los distintos procedimientos y tareas que personal de la organización CAMO lleva a cabo para gestionar la aeronavegabilidad continuada de las aeronaves que tiene contratadas.

La realización de estos procedimientos y tareas requieren de un elevado nivel de concentración por parte del personal técnico que, de mantenerse en el tiempo, pueden afectar de forma negativa al desempeño humano, pudiendo llegar a provocar **errores** en el desarrollo de su trabajo. De forma adicional, se deben tener en consideración factores humanos [2] tales como el estrés, motivación personal, salud emocional, etc., que pueden tener del mismo modo efectos negativos en la realización del trabajo.

El otro motivo principal es el auge del desarrollo de la Inteligencia Artificial (IA) y el creciente interés por parte del tejido empresarial e industrial[3] sobre las ventajas que supone su aplicación en el ámbito empresarial, que van desde la automatización de procesos hasta la reducción de errores humanos[4], mejorando de este modo la productividad de los empleados [5] (ver Figura 3).

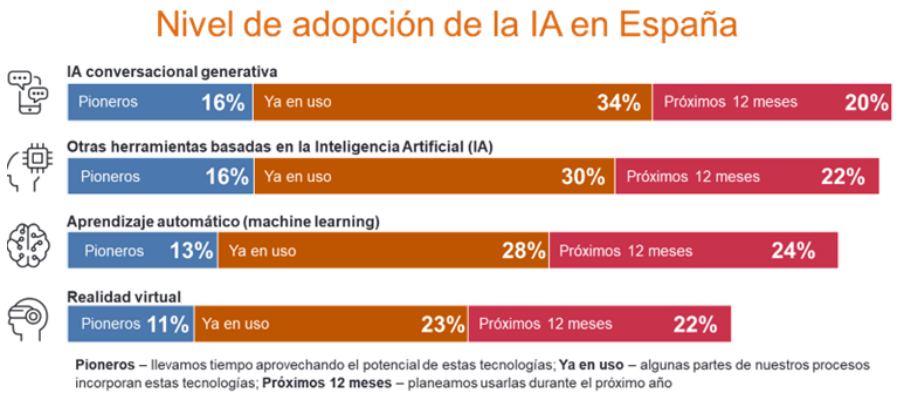


Figura 3: Gráfico del Nivel de adopción de la IA en España. Fuente: [3]

Es por ello que este TFG pretende explorar la introducción, a través de un modelo práctico y funcional, de nuevas aplicaciones procedentes del desarrollo de la IA (Sistemas de Gestión Documental, Procesamiento de Lenguaje Natural, etc.) en una organización CAMO. El modelo debe asistir al personal técnico de una organización CAMO en el desarrollo de sus procedimientos y tareas, reduciendo errores y aumentando su productividad.

1.3 Presentación de la Empresa

Heliswiss es una empresa ubicada en el Aeropuerto de Sabadell (Figura 4) que ofrece soluciones de Mantenimiento y Reparación de Aeronaves y de Gestión de Aeronavegabilidad (servicios CAMO) a través de sus aprobaciones como Organización Parte 145 (ES.145.023) y como Organización CAMO (ES.CAMO.021).

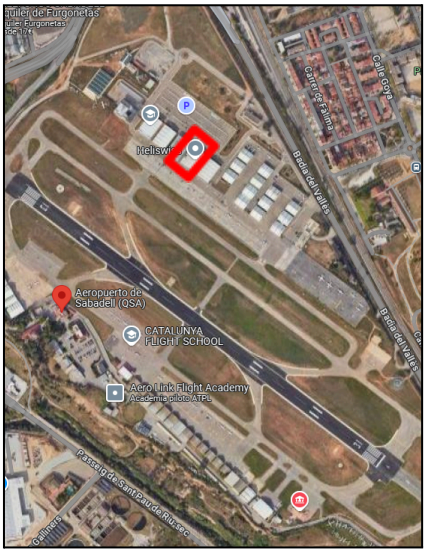


Figura 4: Ubicación Hangar Heliswiss: Hangar nº 4, Lado Norte Aeropuerto Sabadell.
Fuente: Google Maps

Fundada en 1981 en Barcelona, en un primer momento su actividad principal consistió en la operación de helicópteros y pequeñas aeronaves de ala fija. En 1993 obtiene autorización como organización JAR-145 (precursora de la actual autorización como organización EASA Parte-145), y en 2002 se establece en su ubicación actual, el Aeropuerto de Sabadell, donde desde entonces ha consolidado su estructura empresarial y de servicios incorporando la pintura, distribución o la compra, venta y arrendamiento de aeronaves (Figura 5).

Figura 5: Servicios y Estructura Organizativa actual de Heliswiss Ibérica.
Fuente: https://heliswiss.es/

La aprobación como centro de mantenimiento Parte-145 para realizar mantenimiento incluye, dentro de su alcance, a las siguientes familias de aeronaves: Cirrus SR series, Airbus Helicopters (AS350B Series, EC135 Series, H120, etc.), Bell (212, 412) y Robinson Helicopter Company (R22, R44 y R66 Series). De forma adicional, la aprobación incluye alcance para realizar trabajos de mantenimiento sobre motores de turbina (ROLLS ROYCE, Safran, etc.) y diversas familias de componentes

(navegación, comunicaciones, baterías, etc.). El centro de mantenimiento cuenta con un total de 14 técnicos de mantenimiento para cubrir todo el alcance indicado anteriormente, y en 2024 se llevaron a cabo un total de 520 órdenes de trabajo de mantenimiento.

De forma similar, la aprobación como organización CAMO incluye alcance para la gestión de la aeronavegabilidad de aeronaves Cirrus SR Series, Airbus Helicopters (AS350B Series, EC135 Series, etc.), Bell (212, 412) y Robinson Helicopter Company (R22, R44 y R66 Series). Heliswiss ofrece este servicio a sus clientes (propietarios u operadores) de forma que éstos puedan prescindir del seguimiento de la aeronavegabilidad y se concentren en la operación de la aeronave. La organización CAMO cuenta con un total de 3 ingenieros y gestiona la aeronavegabilidad continuada de 17 aeronaves de diferentes clientes/operadores.

Actualmente, Heliswiss se encuentra en un proceso de expansión, incorporando nuevos modelos de aeronave al alcance de su aprobación de mantenimiento y CAMO (Cirrus SF50, H145, etc.) por tal de poder ofrecer sus servicios a una mayor red de clientes europeos, así como ampliando sus instalaciones con instalaciones especializadas de pintura aeronáutica y de MRO (*Maintenance, Repair and Operations)* para aviación ejecutiva.

Figura 6: Aeronave Cirrus SF50
Fuente: https://cirrusaircraft.es/

1.4 Objetivos

El objetivo general de este trabajo es explorar la introducción de nuevas aplicaciones procedentes del desarrollo de la IA en el estudio de la aplicabilidad de Directivas de Aeronavegabilidad para las aeronaves gestionadas por una organización CAMO.

Este objetivo general se puede descomponer en diferentes objetivos específicos:

1. Revisar la literatura de uso de aplicaciones de IA en la gestión de la aeronavegabilidad continuada de las aeronaves.
2. Definir los requisitos del modelo práctico a desarrollar.
3. Comparar y definir las tecnologías de IA para el estudio de la aplicabilidad de Directivas de Aeronavegabilidad.
4. Diseñar e implementar el modelo práctico funcional.
5. Realizar una prueba y analizar y discutir resultados.

1.5 Metodología

Para definir los requisitos necesarios que ha de cumplir el modelo práctico que se quiere desarrollar (inputs, outputs, etc.), se realizarán **entrevistas** al personal de la organización CAMO responsable de obtener, analizar y programar las Directivas de Aeronavegabilidad en el programa de gestión de aeronavegabilidad continuada de la empresa Heliswiss, denominado GMA. De forma adicional, también se llevará a cabo la **observación directa** de este proceso para obtener una aproximación realista de cómo debe trabajar el modelo práctico a desarrollar (es decir, qué características de uso debe tener para su uso práctico en la empresa).

Una vez definidos los **requisitos y características del modelo práctico**, se iniciará el diseño del software. Para diseñar el software se ha optado por el lenguaje de programación Python [6,7,8], debido a que se trata de un lenguaje:

- gratuito, lo que facilita su uso para proyectos sin un retorno económico;
- con licencia de código abierto (*open source license)*, lo que facilita su uso y distribución;
- sencillo y fácil de aprender, con una sintaxis muy limpia;
- popular, con una gran comunidad activa tanto a nivel internacional como a nivel nacional a la que poder realizar consultas a través de foros especializados, blogs, etc.;
- con una gran cantidad de documentación para consulta, debido a su carácter *open source* y a sus comunidades de usuarios;
- permite descargar y gestionar herramientas y librerías adicionales que se pueden utilizar para mejorar los programas.

El diseño del software se llevará a cabo en 2 fases, donde en cada fase se desarrolla una función principal del modelo:

- FASE 1 Procesamiento de Documentos: en esta primera fase se requiere que el modelo pueda extraer texto de documentos escaneados en PDF, para posteriormente extraer información clave de las AD (NLP).
- FASE 2 Comprobación de Aplicabilidad: en esta segunda fase el modelo debe comparar la información extraída de las AD con la información almacenada de las aeronaves en la base de datos (BBDD) e indicar si es necesario revisar las acciones requeridas en la AD para alguna de las aeronaves.

Finalmente, se procederá a **realizar pruebas** con el modelo práctico, realizando el análisis de Directivas de Aeronavegabilidad y comparando el resultado obtenido contra los resultados reales de una aeronave incluida en la CAMO de Heliswiss.

1.6 Cronograma

7/02 - 20/02: Elaborar el informe previo.
21/02 - 28/02: Revisar literatura.
01/03 - 06/03: Definir los requisitos del modelo.
07/03 - 13/03: Definir las aplicaciones de desarrollo de IA a utilizar en el modelo.
14/03 - 09/06: Desarrollo del modelo funcional.
10/06 - 20/06: Realizar pruebas con el modelo práctico y comparar resultados.

1.7 Riesgos

A continuación se exponen algunos de los riesgos identificados en la elaboración del TFG, así como propuestas para minimizar los mismos:

- *Dificultad técnica del modelo elevada*: el modelo práctico requiere el uso e integración de aplicaciones relacionadas con la IA, lo que puede dificultar su desarrollo a nivel de programación.
Para minimizar este riesgo:

- se acota el objetivo del modelo práctico al estudio de la aplicabilidad de las AD's, ya que éstas disponen de una estructura similar en su redacción que facilitará el análisis y extracción de datos;
- el modelo se desarrollará a través del lenguaje de programación de Python, que posee bibliotecas avanzadas para procesamiento de lenguaje natural y procesamiento de datos.

- *Modelo no resulta efectivo para su implementación en escenarios reales*: el presente TFG define como uno de sus objetivos principales el diseñar e implementar un modelo práctico funcional, por lo que uno de los principales riesgos que presenta la elaboración de este TFG es que el modelo finalmente no resulte funcional debido a un planteamiento incorrecto del mismo (no extraiga bien los datos de las AD, no pueda llevar a cabo la comprobación en la BBDD, etc.).
Para minimizar este riesgo:

- se definen desde el principio del proyecto los INPUTS que son necesarios extraer de las AD para que el modelo resulte efectivo en un escenario real, y se trabaja en el desarrollo del modelo a través de la obtención correcta de esos INPUTS.

1.8 Viabilidad Legal

En el capítulo 1.1 del presente TFG se establece que la *Comission Regulation (EU) No 1321/2014* del 26 de Noviembre 2014 es la normativa de aplicación para una Organización CAMO (a través de sus anexos I y Vc).

Para estudiar la viabilidad legal del modelo que se pretende desarrollar, por tanto, se lleva a cabo una revisión de los requisitos establecidos en dicha normativa relacionados con el estudio de aplicabilidad y registro de AD en la aeronavegabilidad continuada de una aeronave:

M.A.301 - Tareas de Aeronavegabilidad Continuada - indica que: *la aeronavegabilidad continuada de una aeronave debe asegurarse, entre otros, a través del cumplimiento de cualquier Directiva de Aeronavegabilidad (AD) aplicable.*

M.A.303 - Directivas de Aeronavegabilidad - indica que: *cualquier directiva de aeronavegabilidad debe realizarse dentro de los requisitos de dicha directiva de aeronavegabilidad, a menos que la Agencia especifique lo contrario.*

M.A.305 - Sistema de registro de la aeronavegabilidad continua de una aeronave - indica, en su epígrafe (c), que: *"los registros de aeronavegabilidad continuada de una aeronave deben incluir [...] el estado actual de las AD y medidas solicitadas por la autoridad competente como acción inmediata ante un problema de seguridad."*

M.A.401 - Datos de mantenimiento - indica que: *para el propósito de este Anexo, los datos de mantenimiento aplicables [...] es cualquier Directiva de Aeronavegabilidad.*

Por tanto, no se deriva de la normativa ninguna prohibición al uso de la IA al apoyo de estudio de aplicabilidad de AD's.

1.9 Estructura de la Memoria

1. Introducción
   Capítulo inicial de la memoria donde se describe el proyecto a desarrollar en el TFG, se definen los objetivos del trabajo y se ofrece información sobre la metodología y los riesgos.
2. Descripción de la tarea a realizar por el modelo
   Capítulo que especifica de un modo conceptual la tarea a realizar por parte del modelo a programar, indicando las condiciones necesarias para su funcionamiento.
3. Desarrollo del código del modelo para la extracción y procesamiento de los datos de la AD
   Capítulo con el detalle de la programación de la parte del modelo responsable de la extracción y el procesamiento de datos procedentes de la AD.
4. Desarrollo del código del modelo para llevar a cabo la comprobación de si una AD afecta a alguna de las aeronaves incluidas en una Organización CAMO
   Capítulo con el detalle de la programación de la parte del modelo responsable de comparar los datos extraídos de la AD con los datos almacenados en la BBDD de aeronaves de la CAMO.
5. Implementación en un escenario real
   Capítulo que recoge los resultados obtenidos a través del modelo, y donde se comparan éstos resultados con resultados obtenidos en escenarios reales.
6. Conclusiones
   Capítulo final de la memoria que indica las conclusiones obtenidas tras el desarrollo del modelo y su implementación en un escenario real.

2. Descripción de las tareas a realizar por el modelo

2.1 Tareas principales a realizar

El modelo a desarrollar en el presente TFG tiene como principal función asistir al personal técnico de una organización CAMO en el estudio de la aplicabilidad de Directivas de Aeronavegabilidad para las aeronaves que gestiona bajo su aprobación.

Para poder llevar a cabo esta función, se define que el modelo debe realizar las siguientes tres tareas principales:

1. **extraer** la información relevante contenida en una AD;
2. **comparar** esta información con la información de las aeronaves almacenada en la base de datos de la Organización CAMO, y ver si la información de alguna aeronave coincide con la información extraída;
3. **indicar el resultado** obtenido al personal técnico: indicar aquellas aeronaves que se ven afectadas por la AD para revisar las acciones requeridas, permitiendo descartar el resto de aeronaves incluidas en la Organización CAMO.

En los siguientes subapartados se desarrolla el proceso actual que desarrollan los técnicos de Heliswiss, y se definen en qué puntos interviene el modelo para realizar las tareas indicadas en el apartado anterior.

2.2 Tareas a realizar por el modelo en el inicio del proceso

Para el desarrollo de este apartado, se han llevado a cabo varias entrevistas y se ha observado de forma directa el proceso que realiza el personal de la CAMO de Heliswiss para el estudio de aplicabilidad de Directivas de Aeronavegabilidad.

El proceso puede dar **inicio** de dos formas diferentes, dependiendo de si el estudio de aplicabilidad se requiere llevar a cabo a una aeronave cuya gestión se encuentra ya incluida en la CAMO (es decir, una aeronave en cuya gestión se lleva ya tiempo trabajando), o si se requiere llevar a cabo en una aeronave transferida por parte de otra organización CAMO (ya sea debido a que el operador decide cambiar de CAMO, a la compra de una aeronave de segunda mano, etc.). En ambos casos, la forma en que se realiza el estudio de aplicabilidad de las AD es el mismo, únicamente varía el número de AD’s que se deben estudiar.

Para el caso en que una aeronave se encuentra ya incluida en la CAMO, el estudio de aplicabilidad de las AD se realiza conforme éstas se publican por parte de EASA. Al publicarse una AD, EASA envía un aviso automático que el personal de la CAMO recibe en su correo electrónico. Tras recibir el aviso, el personal de la CAMO se descarga la AD de la web de EASA (https://ad.easa.europa.eu/) y comprueba si dicha AD aplica a alguna de las aeronaves gestionadas. Por tanto, en este caso el estudio de aplicabilidad de las AD se realiza de una en una pero se comprueban todas las aeronaves.

Por otro lado, para el caso en que la aeronave es transferida por otra organización CAMO, el estudio de aplicabilidad de las AD se realiza a través del listado de AD por modelo que se obtiene de la página web de EASA. Con la información sobre el modelo de aeronave a incorporar a la CAMO, el personal descarga todas las AD aplicables hasta la fecha de fabricación de la aeronave que aparecen en el listado de la página web. Por tanto, en este caso se estudian varias AD seguidas, pero para una única aeronave.

La Figura 7 representa el filtro aplicado en la web de EASA para mostrar las AD del modelo de aeronave AS 350 B3. Una vez aplicado, la página web muestra como resultado el listado que se observa en la Figura 8, donde se distingue: número de AD, fecha de publicación, título, fabricante y modelo, fecha efectiva y enlace para descargar el documento.

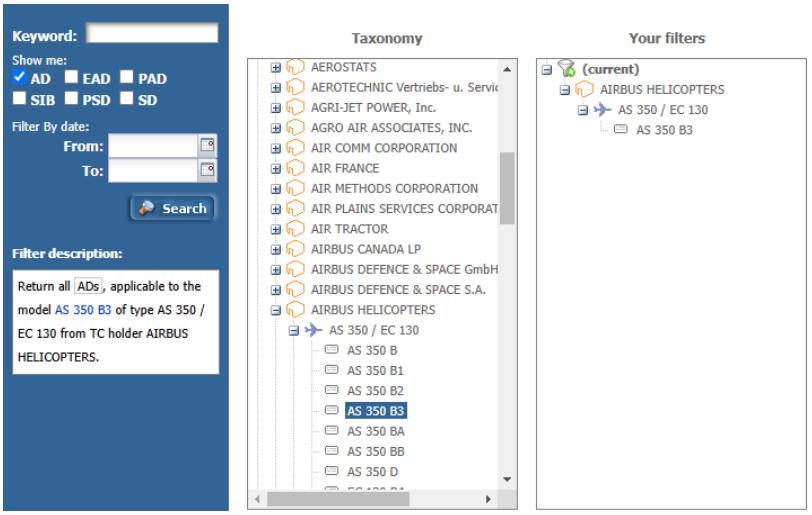


Figura 7: Filtro aplicado para ver las AD de las aeronaves modelo AS 350 B3
Fuente: [8]

Number	Issued by	Issue date	Subject	Approval Holder / Type Designation	Effective date	Attachment
2025-0036		2025-02-12	Equipment / Furnishings – Cargo Hook Assembly – Inspection	AIRBUS HELICOPTERS AS 350 / EC 130	2025-02-26	207 kb 384 kb
2025-0025		2025-01-23	Equipment / Furnishings – Emergency Release Control of Cargo-Swing Installation – Inspection	AIRBUS HELICOPTERS AS 350 / EC 130 AS 355	2025-02-06	196 kb
2021-0282R1		2025-01-08	[Correction] Tail Rotor – Tail Rotor Head Pitch Change Unit Bearing Spacer – Marking / Check	AIRBUS HELICOPTERS AS 350 / EC 130 AS 355	2024-07-17	206 kb 2 mb

Figura 8: Ejemplo de listado de AD obtenido con el filtro aplicado de la Figura 7.
Fuente: [8]

De los dos escenarios planteados anteriormente, los técnicos de la CAMO de Heliswiss indican que este segundo escenario resulta más complejo que el primero debido a la cantidad de información y registros a evaluar cuando se transfiere una aeronave de una organización CAMO a otra: asegurar los componentes instalados, evaluar y controlar las modificaciones realizadas, el estado de mantenimiento según el programa de mantenimiento, revisar las AD que se han aplicado anteriormente, etc. Es imprescindible garantizar que se dispone de toda la información correcta para no cometer futuros errores en el estudio de aplicabilidad de las AD una vez su gestión de la aeronavegabilidad se transfiere a la CAMO de Heliswiss.

Por último, es importante destacar que para el caso concreto de aeronaves nuevas incluidas en la CAMO (es decir, cuando se incluye en la CAMO una aeronave comprada directamente al fabricante), se procede de la misma forma que una aeronave que ya se encuentra incluida en la CAMO, ya que el estudio de aplicabilidad de AD emitidas con anterioridad a la fecha de fabricación la realiza el propio fabricante de la aeronave.

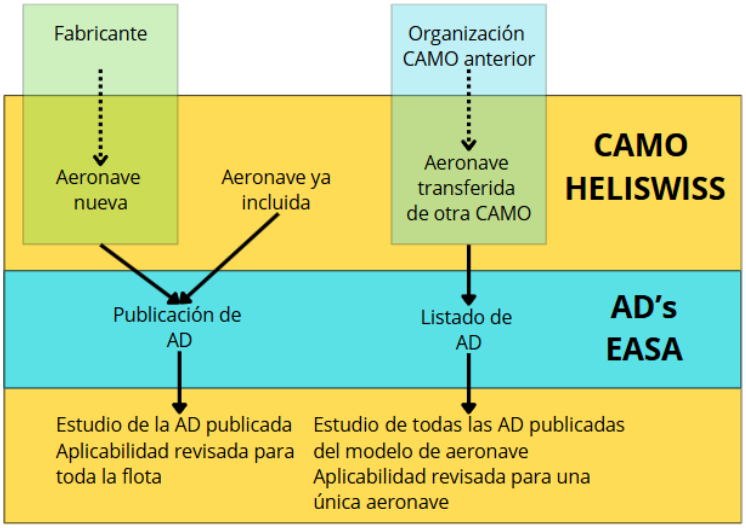


Figura 9: Inicio del proceso de estudio de aplicabilidad.
Fuente: Elaboración propia

En el inicio del proceso, ya se produzca por la emisión de una nueva AD o por la transferencia de una aeronave por otra CAMO, por tanto, el modelo no ejecuta ni realiza ninguna tarea debido a que los inputs necesarios para el funcionamiento del modelo son proporcionados de forma manual por los técnicos de la CAMO, que son quienes descargan las AD de la web de EASA.

2.3 Tareas a realizar por el modelo para interpretar la AD

2.3.1 Estructura de una Directiva de Aeronavegabilidad

Para poder entrar en detalle de las tareas que debe realizar el modelo a desarrollar, es necesario entender la forma en que se estructura una AD. Las AD publicadas por EASA mantienen siempre la misma **estructura**, facilitando su estudio e interpretación por parte de los técnicos. De este modo, la AD se estructura conforme:

1. **Cabecera**: donde se indica el número de Directiva y la fecha de publicación (ver Figura 10). El número de Directiva actúa como identificador único en el registro de Directivas de las aeronaves gestionadas por Heliswiss.

Figura 10: Cabecera de una Directiva de Aeronavegabilidad.

Fuente: [8]

2. **Detalles del fabricante**: indica el fabricante de la aeronave y la designación del fabricante del modelo afectado por la Directiva.

Design Approval Holder's Name:	Type/Model designation(s):
AIRBUS HELICOPTERS DEUTSCHLAND GmbH	EC135, EC635 and MBB-BK117 helicopters

Figura 11: Detalles del fabricante.
Fuente: [9]

3. **Título de la AD**: ofrece información sobre el sistema afectado según clasificación ATA100 y un breve resumen del contenido de la Directiva.

ATA 31 – Instruments – Warning Unit Emergency Off Switches – Operational Check

Figura 12: Título de la Directiva.
Fuente: [9]

4. **Aplicabilidad**: se indican los modelos, variantes y número de serie específicos de las aeronaves afectadas por la Directiva. Este apartado actúa de primer filtro para los técnicos de Heliswiss a la hora de evaluar la aplicabilidad de una Directiva a la flota de aeronaves gestionadas.

Applicability:
EC135 P1, EC135 P2, EC135 P2+, EC135 P3, EC135 T1, EC135 T2, EC135 T2+, EC135 T3, EC635 P2+, EC635 P3, EC635 T1, EC635 T2+ and EC635 T3 helicopters, all variants, all serial numbers (s/n).

MBB-BK117 C-2, MBB-BK117 D-2, MBB-BK117 D-3 and MBB-BK117 D-3m, helicopters, all variants, all s/n.

Figura 12: Aplicabilidad de la Directiva.
Fuente: [9]

5. **Definiciones**: en este apartado se identifican las definiciones de los conceptos necesarios para interpretar la AD. Este apartado actúa de segundo filtro para los técnicos de la CAMO, ya que en este apartado se identifican los componentes afectados en el modelo de aeronave a través del *Part Number* (P/N).

Definitions:
For the purpose of this AD, the following definitions apply:

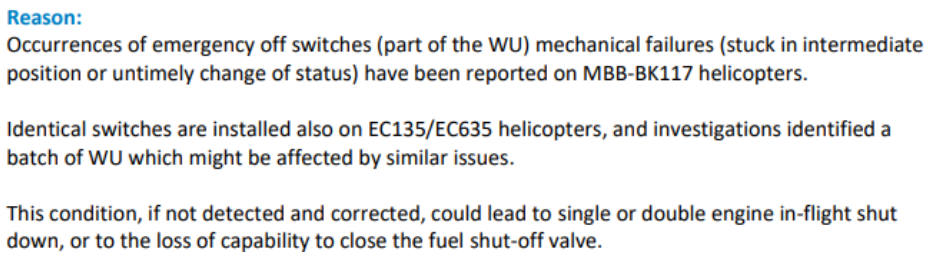
The ASB: AH Alert Service Bulletin (ASB) MBB-BK117-31-55-0001 or ASB EC135-31-55-0001, as applicable.

Affected part: Warning unit (WU), all Part Numbers (P/N), having an s/n as identified in the ASB, or having an unknown s/n.

Groups: Group 1 helicopters are those that have an affected part installed. Group 2 helicopters are those that do not have an affected part installed.

Figura 13: Definiciones para interpretar la Directiva.
Fuente: [9]

6. **Razón**: breve comentario donde EASA indica los motivos por los que se publica la Directiva (comportamientos anómalos de componentes que afectan a la seguridad de las operaciones, degradación de componentes bajo condiciones especiales, etc.). Esta información no es analizada en detalle por el personal de la CAMO de Heliswiss.



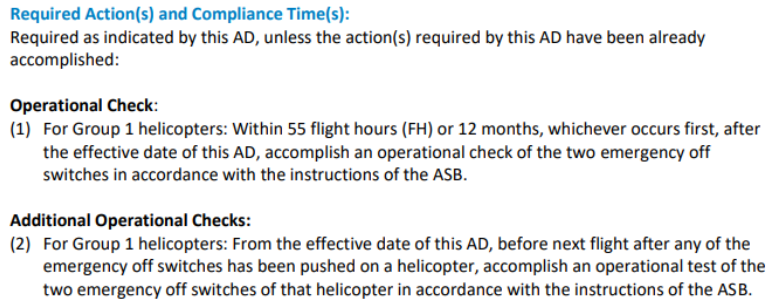
Reason:
Occurrences of emergency off switches (part of the WU) mechanical failures (stuck in intermediate position or untimely change of status) have been reported on MBB-BK117 helicopters.

Identical switches are installed also on EC135/EC635 helicopters, and investigations identified a batch of WU which might be affected by similar issues.

This condition, if not detected and corrected, could lead to single or double engine in-flight shut down, or to the loss of capability to close the fuel shut-off valve.

Figura 14: Razones para la aplicación de la Directiva.
Fuente: [9]

7. **Acciones requeridas y tiempo de cumplimiento**: parte troncal de la Directiva donde EASA especifica las acciones a llevar a cabo y el tiempo límite para la aplicación de dichas acciones en la aeronave y componentes afectados según la aplicabilidad y definiciones especificados en apartados anteriores.



Required Action(s) and Compliance Time(s):
Required as indicated by this AD, unless the action(s) required by this AD have been already accomplished:

Operational Check:
(1) For Group 1 helicopters: Within 55 flight hours (FH) or 12 months, whichever occurs first, after the effective date of this AD, accomplish an operational check of the two emergency off switches in accordance with the instructions of the ASB.

Additional Operational Checks:
(2) For Group 1 helicopters: From the effective date of this AD, before next flight after any of the emergency off switches has been pushed on a helicopter, accomplish an operational test of the two emergency off switches of that helicopter in accordance with the instructions of the ASB.

Figura 15: Acciones requeridas y tiempo de cumplimiento de la Directiva.
Fuente: [9]

Revisada y explicada la estructura, podemos definir los inputs y outputs del modelo conceptual como:

- INPUTS que proporciona la AD: modelos, variantes y número de serie de la aeronave; P/N del componente afectado.
- OUTPUTS que proporciona la AD: acción requerida; tiempo de cumplimiento.

En el caso de la AD tomada como ejemplo en las Figuras 10-15 para el desarrollo de este apartado, a través de los siguientes INPUTS que proporciona la Directiva:

- Aplicabilidad: EC135T3 con S/N - todos -
- Componente: Warning Unit con P/N - todos -

En el caso que una aeronave gestionada por la CAMO cumpla con las características indicadas por los INPUTS (se encuentra en el Grupo 1 indicado en las definiciones de la AD), se definirán los siguientes OUTPUTS (acciones a realizar):

- Realizar un chequeo funcional de los dos "emergency off switches" de acuerdo al ASB antes de las 55 horas de vuelo o 12 meses, lo que primero ocurra.
- Realizar un test operacional de los dos "emergency off switches" antes del siguiente vuelo después de que cualquiera de los "emergency off switches" hayan sido presionados en un helicóptero.

2.3.2 Tareas a realizar por el modelo en su ejecución principal

Este apartado desarrolla el procedimiento que el personal técnico de la CAMO realiza una vez disponen de las AD necesarias para estudiar su aplicabilidad.

- El técnico de la CAMO revisa la aplicabilidad de la Directiva según modelo, variante y número de serie de la aeronave (INPUTS) en la base de datos que contiene la información de las aeronaves de la flota de Heliswiss.
  Esta comparación se realiza actualmente de forma manual, debiendo revisar el propio técnico todas las aeronaves incluidas en la flota (en el caso de tratarse de una nueva AD publicada), o debiendo revisar todas las AD de un modelo en concreto para analizar si afectan por variante y número de serie (en el caso de tratarse de una aeronave transferida de otra CAMO).

  La tarea del modelo en este punto del proceso consiste en comparar el INPUT de aplicabilidad proporcionado por la Directiva con la información sobre modelos, variantes y números de serie contenida en la base de datos de la flota de Heliswiss, y poder indicar de este modo si la Directiva aplica o no a alguna de las aeronaves de la flota.

  Para poder actuar como primer filtro, el modelo debe reconocer e interpretar correctamente la información contenida en la AD. De este modo, es necesario que el modelo extraiga del PDF de la AD y reconozca posteriormente la siguiente información:
    - Modelos afectados
    - Variantes afectadas (un modelo puede tener diferentes variantes, dependiendo por ejemplo del motor equipado)
    - Números de serie afectados

- En caso de encontrar una aeronave en la flota afectada por la Directiva, el técnico revisa posteriormente las definiciones incluidas en la AD por tal de analizar los componentes afectados.
  Del mismo modo que en el caso anterior, esta comparación se realiza actualmente de forma manual, debiendo revisar el propio técnico si alguna de las aeronaves incluidas en la aplicabilidad de la AD tiene instalado o no el componente indicado en la descripción. La búsqueda se realiza a través del P/N, que actúa como identificador único para identificar componentes instalados en una aeronave.

La tarea del modelo en este punto del proceso consiste en comparar el P/N del componente indicado en la descripción de la AD con el listado de componentes instalados en las aeronaves (e incluidos de igual forma en la base de datos de Heliswiss) a las que le es de aplicación la AD.

Para poder actuar como segundo filtro, el modelo a desarrollar debe reconocer e interpretar correctamente la información contenida en la AD. De este modo, es necesario que el modelo extraiga del PDF de la AD y reconozca posteriormente la siguiente información:

- Partes afectadas
- Part Numbers afectados

- En último lugar, en caso de encontrar que una aeronave tiene instalado un componente afectado, el técnico analiza los OUTPUTS (las acciones requeridas según lo indicado en la AD y el tiempo máximo de cumplimiento) de la Directiva, para trasladarlos al programa de gestión de aeronavegabilidad de la aeronave.

  Los OUTPUTS pueden contener una elevada cantidad de información debido a que las acciones requeridas pueden depender de múltiples factores (la AD puede disponer de varios grupos de actividades a realizar según requisitos previos, estado del Programa de Mantenimiento, modificaciones de la aeronave, etc.).

  En el presente TFG se prioriza la comparación de los INPUTS con la información almacenada sobre las aeronaves en una base de datos, por lo que el OUTPUT que del modelo consistirá en indicar al técnico de la CAMO si hay o no coincidencia de la AD analizada con alguna aeronave incluida en la base de datos de la organización y si, por tanto, es necesario revisar o no las acciones a programar de esa AD.

3. Desarrollo del código del modelo para la extracción y procesamiento de los datos de la AD

3.1 Convertir un PDF en texto

Con objeto de extraer la información que conformarán los inputs para el análisis de las AD's, la primera tarea que necesita ejecutar nuestro modelo es la de recuperar el texto de los archivos PDF en que se descargan las AD's.

El formato de archivo PDF (Portable Document Format) proporciona a los usuarios una forma sencilla y segura de presentar e intercambiar documentos. Permiten integrar varios tipos de contenidos, como texto e imágenes. No obstante, este formato de archivo no permite al modelo extraer la información de la Directiva, por lo que es necesario recuperar el texto del archivo.

Para esta primera tarea, incorporaremos la librería PyPDF2 a nuestro modelo, accesible a través de la web Python Package Index (https://pypi.org). PyPDF2 es una librería PDF de código abierto que puede recuperar texto de archivos PDF [10].

Para incorporar la librería a nuestro modelo, tenemos que instalarlo con el instalador de paquetes de Python (pip) e importarlo a nuestro proyecto [8](Figura 16):

```
1   import PyPDF2
2
```

Figura 16: Importar librería PyPDF2 a nuestro modelo.
Fuente: Código propio

Con la librería importada, programamos la función para extraer el texto de un PDF (Figura 17):

```
9   # ========================
10  # Función para extraer texto de un PDF
11  # ========================
12
13  def extraer_texto_pdf(Directiva):
14      texto_completo = ""
15
16      # Abrir el archivo PDF en modo lectura binaria
17      with open(Directiva, "rb") as archivo_pdf:
18          lector_pdf = PyPDF2.PdfReader(archivo_pdf)
19
20          # Recorrer todas las páginas del PDF
21          for num_pagina in range(len(lector_pdf.pages)):
22              pagina = lector_pdf.pages[num_pagina]
23              texto_completo += pagina.extract_text() + "\n"
24
25      return texto_completo
26
27  # Directiva a procesar
28  Directiva = "AS350.pdf"
29
30  # Extraer texto del PDF
31  texto_extraido = extraer_texto_pdf(Directiva)
```

Figura 17: Función para extraer texto de un PDF.
Fuente: Código propio

Donde:

- en la línea 13, se define la función extraer_texto_pdf, en la que se crea la variable texto_completo (línea 14) para almacenar el texto extraído de cada página;
- en la línea 17, se abre el archivo en modo de lectura binaria ("rb") para leer su contenido como una secuencia de bytes;
- en la línea 18, se crea la variable lector_pdf. En esta variable se inicializa el objeto PdfReader, que permite acceder a las páginas del documento;
- en la línea 21, se itera a través de los comandos "for - in" (se recorre cada página del PDF) [8] para el total de páginas del documento;
- el total de páginas se obtiene a través de la función "len" (nos permite conocer la cantidad de elementos que contiene una lista [8]);
- en la línea 23 se usa la función extract_tex para obtener el texto de la página, y se guarda en la variable texto_completo creada en la línea 5;
- en la línea 31 se define la variable texto_extraido, que guarda el resultado de aplicar la función extraer_texto_pdf a una Directiva definida previamente;

Cuando otorgamos un valor concreto a la variable "Directiva" y ejecutamos el modelo, éste nos devuelve el texto que contiene la AD (Figura 18), y lo almacena dentro de la variable texto_extraido:

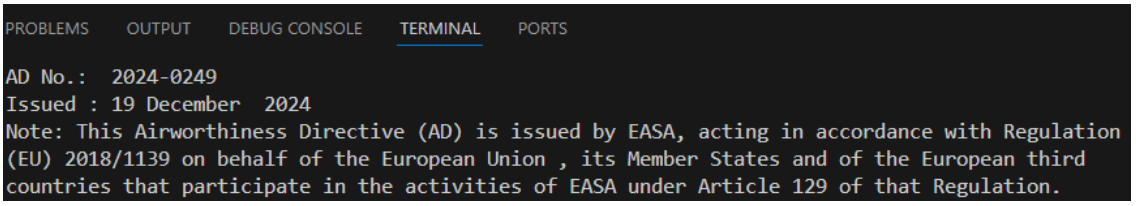


Figura 18: Resultado de ejecutar el código de las Figuras 16-17.
Fuente: Código propio

3.1.2 Convertir una imagen en texto

La información que necesita nuestro modelo para el análisis de las AD's puede encontrarse en otros formatos diferentes al formato PDF, como por ejemplo en formato de imágen (documento escaneado, fotografía, etc.).

Python dispone de librerías para trabajar con reconocimiento óptico de carácteres (OCR por sus siglas en inglés) que permiten extraer el texto contenido en una imagen. Una vez extraído el texto, el modelo puede trabajar con él del mismo modo que cuando extraemos el texto de un archivo PDF.

Esta función no se ha implementado en el modelo actual al disponer de la información necesaria en archivos PDF, pero puede ser necesaria de implementar en modelos con INPUTS procedentes de distintos tipos de formato.

3.2 Extraer información de la AD

El texto de la AD almacenado en la función extraer_texto_pdf(Directiva) contiene la información necesaria para actuar como INPUT de nuestro modelo.

Para procesar la información, nuestro modelo hace uso de herramientas para el procesamiento de lenguaje natural (NLP por sus siglas en inglés). Para nuestro modelo, se ha optado por el paquete de Python NLTK para el procesamiento de lenguaje natural.

NLTK es un conjunto de módulos interdependientes y organizados en una jerarquía integrados en un único paquete de Python que proporciona funcionalidades para el procesamiento del lenguaje natural. NLTK dispone de:

- módulos principales: definen los tipos de datos básicos que se utilizan en todo el paquete y
- módulos específicos: módulos dedicados a una tarea específica de procesamiento del lenguaje natural. Por ejemplo, el módulo específico *nltk.tokenizer* se encarga de tokenizar el texto. [11]

En el caso de nuestro modelo, NLTK nos permite extraer la información del texto de la AD como se muestra en la figura 19:

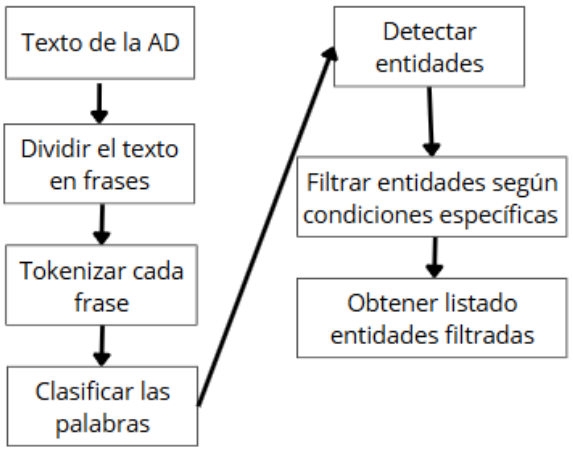


Figura 19: Arquitectura del sistema de extracción de información de nuestro modelo
Fuente: Elaboración propia

El apartado 3.2.1 ofrece el detalle de la arquitectura mostrada en la Figura 19 para la detección en nuestro modelo informático de los modelos de aeronaves y de partes afectadas incluidas en el texto de una AD.

Las figuras incluidas en el próximo apartado 3.2.1 con los resultados obtenidos de cada parte del código se han obtenido de la AD No.: 2025-0025. Estas figuras se han añadido como ayuda visual para entender qué hace cada parte del código indicado.

3.2.1 Detección de modelos de aeronaves y partes afectadas incluidas en el texto de una AD.

3.2.1.1 Extracción del texto de Aplicabilidad y Partes Afectadas

Para incorporar el paquete de Python NLTK a nuestro modelo, tenemos que instalarlo con el instalador de paquetes de Python (pip) e importarlo a nuestro proyecto [8].

Como los modelos de aeronave se encuentran en el campo "Aplicabilidad" de la AD (ver apartado 2.3.1), el modelo primero extrae el texto de esta parte específica de la AD (Figura 20):

```
74  # Extraer bloque "Applicability" a "Definitions"
75  match = re.search(r"Applicability(.*?)Definitions", texto_extraido, re.DOTALL | re.IGNORECASE)
76  if match:
77      texto_applicability = match.group(1)
78  else:
79      texto_applicability = texto_extraido
80
81  # Limpiar saltos de línea
82  texto_applicability = texto_applicability.replace("\n", " ")
83  texto_applicability = re.sub(r'\s+', ' ', texto_applicability)
```

Figura 20: Función para extraer el texto del campo Aplicabilidad.
Fuente: Código propio

Donde:

- en la línea 75 se define la variable match para la búsqueda (función *search*) del siguiente patrón: texto "Applicability" seguido de cualquier contenido hasta la palabra "Definitions" dentro de la variable texto_extraido (Figura 18);
- en la línea 77 se indica que, si hay una coincidencia en la función search, devuelve el texto que está entre "Applicability" y "Definitions" y lo almacena en la variable texto_applicability;
- en la línea 79 se indica que, si por el contrario, no hay coincidencia en la función search, la variable texto_applicability obtiene el valor de la variable texto_extraido (variable donde está guardado el texto completo).

El resultado que obtenemos al ejecutar el código de la Figura 20 es el mostrado en la Figura 21:

```
AS 350 B2, AS 350 B3, AS 355 NP, AS 355 N and EC 130 B4 helicopters, all serial numbers  equipped
with  ONBOARD systems 3500LB , having Part Number (P/N) 704A41811035  (manufacturer reference
528-023-51).
```

Figura 21: Resultado de ejecutar el código de la Figura 20
Fuente: Código propio

Para el caso de las partes afectadas, como éstas se encuentran en el campo "Definiciones" de la AD, donde se ubica la definición de *Affected Part* (ver apartado 2.3.1), el modelo extrae el texto de esta parte específica de la AD (Figura 22):

```
172  # Buscar TODAS las líneas de partes afectadas (bloques separados por punto y doble salto de línea)
173  matches = re.findall(
174      r"(Affected[^\n]*:\s*((?:.|\n)*?))\.\s*\n\s*\n",
175      texto_extraido,
176      re.IGNORECASE
177  )
178
179  # Almacenar todos los bloques encontrados en una lista
180  partes_affected = []
181  for encabezado, cuerpo in matches:
182      texto_affectedpart = cuerpo.strip().lstrip(": ").strip()
183      if texto_affectedpart:
184          partes_affected.append(texto_affectedpart)
185
```

Figura 22: Función para extraer el texto de los campos *Affected*.
Fuente: Código propio

Pese a que el funcionamiento es similar a la extracción del campo "Aplicabilidad", en este caso es necesario tener en consideración que una única AD puede contener más de una parte afectada (no es el caso de la AD tomada como ejemplo para la elaboración de esta memoria, pero sí de otras analizadas).

Para no dejar fuera ninguna parte afectada de una AD, en el modelo:

- en la línea 173 se define la variable matches para la búsqueda (función *findall*) del siguiente patrón: texto "Affected" seguido de cualquier contenido hasta el próximo punto o doble salto de línea dentro de la variable texto_extraido (Figura 18);
- en la línea 180 se crea la variable partes_affected, que es una lista vacía donde se irán almacenando los textos extraídos de cada bloque que cumpla con el patrón definido;
- en las líneas 183 y 184, el texto extraído de cada bloque de parte afectada se guarda dentro de la lista partes_affected

El resultado que obtenemos al ejecutar el código de la Figura 22 es el mostrado en la Figura 23:

```
['ONBOARD systems 3500LB , having Part Number (P/N) 704A41811035  (manufacturer \nreference 528-023-51), equipped with  dropping control , havi
ng P/N 704A41811037  (manufacturer \nreference  26802400 )']
```

Figura 23: Resultado de ejecutar el código de la Figura 22
Fuente: Código propio

3.2.1.2 Tokenización y POS-tagging

Para que nuestro modelo pueda llevar a cabo la tokenización del texto extraído, previamente se le ha de incorporar el módulo específico para tokenizar (Figura 24):

```
5   from nltk.tokenize import regexp_tokenize
```

Figura 24: Incorporación al modelo del módulo específico para tokenizar incluido en NLTK.
Fuente: Código propio

Una vez extraído y almacenado el texto de los campos "Aplicabilidad" y "Partes Afectadas" de la AD en sus respectivas variables (Figuras 20 y 22), el módulo específico para tokenizar divide el texto en frases, y cada frase a su vez en palabras. [12]

Para ello, incorporamos al modelo la función regexp_tokenize (Figura 25):

```
87    # Tokenizar solo el bloque
88    tokens_app = regexp_tokenize(texto_applicability, r"\w+(?:-\w+)*|\S")
```

```
225       tokens_aff = regexp_tokenize(parte_limpia, r"\w+(?:-\w+)*|\S")
```

Figura 25: Función para tokenizar el texto extraído
Fuente: Código propio

Donde:

- en la línea 88 se define la variable tokens_app donde se almacenan todos los tokens encontrados en la variable texto_applicability.
- en la línea 225 se define la variable tokens_aff donde se almacenan todos los tokens encontrados en la variable parte_limpia (la variable parte_limpia es una variable creada a partir de la variable partes_affected una vez eliminado contenido entre paréntesis y espacios con guiones).
- el uso de la función regexp_tokenize nos permite tokenizar incluyendo reglas personalizadas (por ejemplo, no considerar palabras separadas por guiones como dos tokens diferenciados).

El resultado que obtenemos al ejecutar el código de la Figura 25 es el mostrado en la Figura 26:

```
[':', 'AS', '350', 'B2', ',', 'AS', '350', 'B3', ',', 'AS', '355', 'NP', ',', 'AS', '355', 'N', 'and', 'EC', '130', 'B4', 'helicopters', ',', 'all', 'ser
ial', 'numbers', 'equipped', 'with', 'ONBOARD', 'systems', '3500LB', ',', 'having', 'Part', 'Number', '(', 'P', '/', 'N', ')', '704A41811035', '(', 'manu
facturer', 'reference', '528-023-51', ')', '.']
```

```
['ONBOARD', 'systems', '3500LB', ',', 'having', 'Part', 'Number', '704A41811035', ',', 'equipped', 'with', 'dropping', 'control', ',', 'having'
, 'P', '/', 'N', '704A41811037']
```

Figura 26: Tokenización del texto incluido en las variables texto_applicability y parte_limpia
Fuente: Código propio

Finalizada la tokenización del texto "Aplicabilidad" y "Partes Afectadas" de la AD, es necesario clasificar las palabras en sus categorías gramaticales y etiquetarlas para poder llevar a cabo las tareas de procesamiento de lenguaje natural.

El proceso de clasificar las palabras en sus categorías gramaticales y etiquetarlas como corresponde se conoce como etiquetado gramatical (POS-tagging).

Para que nuestro modelo pueda llevar a cabo el etiquetado gramatical de los tokens almacenados en la variable tokens_app y tokens_aff, previamente se le ha de incorporar el módulo específico para dicho etiquetado gramatical (Figura 27):

```
6    from nltk import pos_tag, RegexpParser
```

Figura 27: Incorporación al modelo del módulo específico para el etiquetado gramatical de los tokens incluido en NLTK.
Fuente: Código propio

Posteriormente, el modelo ejecuta la función pos_tag sobre la variable tokens_app / tokens_aff y guarda dicha información en la variable tags_app / tags_aff (Figura 27), asignando una categoría gramatical a cada token (Figura 28):

```
90    # Etiquetar solo el bloque
91    tags_app = pos_tag(tokens_app)
```

```
226       tags_aff = pos_tag(tokens_aff)
```

Figura 27: Creación de la variable tags_app y tags_aff para la clasificación de los tokens en categorías gramaticales
Fuente: Código propio

```
[(':', ':'), ('AS', 'IN'), ('350', 'CD'), ('B2', 'NNP'), (',', ','), ('AS', 'IN'), ('350', 'CD'), ('B3', 'NNP'), (',', ','), ('AS', 'IN'), ('35
5', 'CD'), ('NP', 'NNP'), (',', ','), ('AS', 'IN'), ('355', 'CD'), ('N', 'NNP'), ('and', 'CC'), ('EC', 'NNP'), ('130', 'CD'), ('B4', 'NNP'), ('
helicopters', 'NNS'), (',', ','), ('all', 'DT'), ('serial', 'JJ'), ('numbers', 'NNS'), ('equipped', 'VBN'), ('with', 'IN'), ('ONBOARD', 'NNP'),
 ('systems', 'NNS'), ('3500LB', 'CD'), (',', ','), ('having', 'VBG'), ('Part', 'NNP'), ('Number', 'NNP'), ('(', '('), ('P', 'NNP'), ('/', 'NNP'
), ('N', 'NNP'), (')', ')'), ('704A41811035', 'CD'), ('(', '('), ('manufacturer', 'NN'), ('reference', 'NN'), ('528-023-51', 'JJ'), (')', ')'),
 ('.', '.')]
```

```
[('ONBOARD', 'NNP'), ('systems', 'NNS'), ('3500LB', 'CD'), (',', ','), ('having', 'VBG'), ('Part', 'NNP'), ('Number', 'NNP'), ('704A41811035',
'CD'), (',', ','), ('equipped', 'VBN'), ('with', 'IN'), ('dropping', 'VBG'), ('control', 'NN'), (',', ','), ('having', 'VBG'), ('P', 'NNP'), ('
/', 'NNP'), ('N', 'NNP'), ('704A41811037', 'CD')]
```

Figura 28: Resultado obtenido de la categorización gramatical de cada token almacenado en la variable tokens_app y tokens_aff.
Fuente: Código propio

A modo de muestra, a continuación se indican:

- las categorizaciones gramaticales para los modelos de aeronaves incluidos en la variable texto_applicability y que se pueden observar en la figura 28:

- AS 350 B2: "AS" se categoriza como "IN" (preposición); "350" se categoriza como "CD" (número cardinal); "B2" se categoriza como "NNP" (nombre propio singular).
- AS 350 B3: "AS" se categoriza como "IN" (preposición); "350" se categoriza como "CD" (número cardinal); "B2" se categoriza como "NNP" (nombre propio singular).
- AS 355 NP: "AS" se categoriza como "IN" (preposición); "350" se categoriza como "CD" (número cardinal); "B2" se categoriza como "NNP" (nombre propio singular).
- AS 355 N: "AS" se categoriza como "IN" (preposición); "350" se categoriza como "CD" (número cardinal); "B2" se categoriza como "NNP" (nombre propio singular).
- EC 130 B4: "EC" se categoriza como "NNP" (nombre propio singular); "130" se categoriza como "CD" (número cardinal); "B4" se categoriza como "NNP" (nombre propio singular).

- las categorizaciones gramaticales para las partes afectadas incluidas en la variable parte_limpia y que se pueden observar en la figura 28:

- ONBOARD systems 3500 LB: "ONBOARD" se categoriza como "NNP" (nombre propio singular); "systems" se categoriza como "NNS" (nombre plural); "3500LB" se categoriza como "CD" (número cardinal).

3.2.1.2 Chunking

El Chunking es la técnica de la que hace uso nuestro modelo informático para la detección de entidades. Las entidades son agrupaciones de palabras etiquetadas gramaticalmente de forma individual que forman una estructura significativa. [12]

Nuestro modelo hace uso del chunking con expresiones regulares. Este método nos permite definir qué tipo de entidad (estructura significativa) queremos detectar, almacenar y procesar en nuestro modelo.

Con la información obtenida sobre la categorización gramatical de los modelos de aeronaves y partes afectadas, definimos las entidades "CANDIDATO_MODELO" y "CANDIDATO_PARTE" con sus respectivas expresiones regulares para un modelo de aeronave (Figura 28) o una parte afectada (Figura 29):

```
44  grammar = r"""
45      CANDIDATO_MODELO:
46          {<IN><CD><NNP|NN>}
47          {<NNP><CD><NNP>}
48  """
```

Figura 28: Entidad "CANDIDATO_MODELO" con las estructuras gramaticales que pueden resultar ser un modelo de aeronave.
Fuente: Código propio

```
50  chunk_grammar = r"""
51      CANDIDATO_PARTE:
52          {<JJ|NN|NNS|NNP|NNPS|CD|RB|VBD|VBZ>{2,7}}
53  """
```

Figura 29: Entidad "CANDIDATO_PARTE" con las estructuras gramaticales que pueden resultar ser una parte afectada.
Fuente: Código propio

Donde:

- en la línea 46 se define la estructura gramatical: preposición + número cardinal + nombre propio singular/nombre. Esta estructura gramatical coincide con modelos de aeronaves como AS 350 B2 o AS 355 NP.
- en la línea 47 se define la estructura gramatical: nombre propio singular + número cardinal + nombre propio singular. Esta estructura gramatical coincide con modelos de aeronaves como EC 130 B4.
- en la línea 50 se define una estructura gramatical que puede estar compuesta por: adjetivo y/o nombre singular y/o nombre plural y/o nombre propio plural y/o número cardinal y/o adverbio y/o verbo en pasado y/o verbo en presente, con una extensión de un mínimo de dos palabras y un máximo de 7.

Tal y como se puede observar, la expresión regular que define la entidad CANDIDATO_PARTE no se encuentra tan acotada como la expresión regular que define la entidad CANDIDATO_MODELO. Esto se debe a que las partes afectadas, a diferencia de los modelos de aeronaves, no siguen una estructura similar entre ellas, e incluyen palabras etiquetadas gramaticalmente de diversa índole.

La expresión regular de la entidad CANDIDATO_PARTE se ha definido de forma menos restrictiva que la expresión regular de CANDIDATO_MODELO (no se indica el orden exacto de las palabras, por ejemplo) para evitar que la expresión regular excluya del chunk las partes afectadas indicadas en la AD.

Tras definir las entidades y sus expresiones regulares, definimos las variables y funciones necesarias para llevar a cabo el chunking (Figura 30):

```
89    # Chunking sobre el bloque de Applicability - CANDIDATO_MODELO
90    chunk_parser = RegexpParser(grammar)
91    chunked_app = chunk_parser.parse(tags_app)
```

```
230       # Chunking sobre los bloques de Partes Afectadas - CANDIDATO_PARTE
231       chunk_parser = RegexpParser(chunk_grammar)
232       chunked_aff = chunk_parser.parse(tags_aff)
233       todos_los_chunks.append((parte_limpia, chunked_aff))
```

Figura 30: Chunking sobre los bloques de texto extraídos según las expresiones regulares definidas
Fuente: Código propio

Donde:

- en las líneas 90 y 231 definimos la variable chunk_parser para crear un analizador de entidades según las expresiones regulares "grammar" y "chunk_grammar" definidas anteriormente.
- en las líneas 91 y 232 definimos las variables chunked_app y chunked_aff que revisan las variables tags_app y tags_aff y agrupan las palabras según las reglas de las expresiones regulares.

El resultado que obtenemos al ejecutar el código de la Figura 30 es una estructura de árbol como el mostrado en la Figura 31:

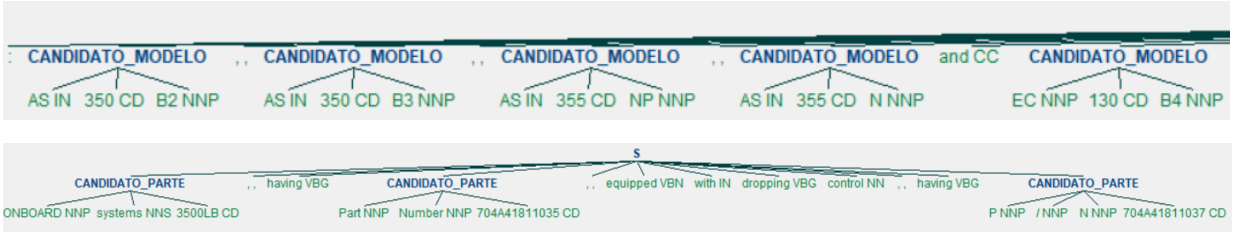


Figura 31: Estructura de árbol tras ejecutar el chunking con las entidades obtenidas según las expresiones regulares definidas
Fuente: Código propio

3.2.1.3 Filtrar entidades según condiciones específicas

En la Figura 31 podemos ver como, en el caso de la AD tomada como ejemplo para el desarrollo de esta memoria, todas las entidades de CANDIDATO_MODELO son, efectivamente, modelos de aeronaves.

No obstante, no todas las entidades de CANDIDATO_PARTE son realmente partes afectadas de la aeronave indicadas en la AD. Esto sucede debido a que, como hemos comentado anteriormente, la expresión regular de la entidad CANDIDATO_PARTE se ha definido de forma menos restrictiva, por lo que captura conjuntos de palabras que también cumplen con la expresión regular.

De este modo tenemos que, de los CANDIDATO_PARTE detectados, únicamente el de la Figura 32 es correcto, mientras que los mostrados en la Figura 33 no lo son:

Figura 32: CANDIDATO_PARTE detectado de forma correcta
Fuente: Código propio

Figura 33: CANDIDATO_PARTE detectado de forma errónea
Fuente: Código propio

Para evitar que el modelo trabaje con entidades erróneas, incorporamos al modelo una expresión regular adicional que ayude a filtrar las entidades obtenidas.

Para el caso de las entidades de CANDIDATO_MODELO, se define la expresión regular patron_modelo_general indicada en la Figura 34:

```
96  patron_modelo_general = re.compile(
97      r"""
98      \b
99      (                                   # Grupo para el prefijo (modelo base)
100         [A-Z]{2,4}                      # Prefijo, 2 a 4 letras mayúsculas
101         \s?                             # Espacio opcional
102         \d{2,3}                         # Número, 2 o 3 dígitos
103     )
104     (?:[\s\-]+                          # Separador (espacio o guion)
105     ([A-Z0-9\-\+]{1,5}[a-z]?))?         # Variante: letras mayúsculas, números, guion, +, y opcional letra minúscula al final
106     \b
107     """,
108     re.IGNORECASE | re.VERBOSE
109 )
```

Figura 34: Expresión regular para filtrar las entidades CANDIDATO_MODELO
Fuente: Código propio

Para el caso de las entidades de CANDIDATO_PARTE, se define la expresión regular patron_parte_general indicada en la Figura 35:

```
# Compilar el patrón para validar nombres de partes
patron_parte_general = re.compile(
    r"""
    ^
    (
        (?:
            [A-Z]{2,}                   # Palabra toda en mayúscula (ELEC)
            | [A-Z][a-z]+               # o capitalizada (Main, Rotor)
            | \d+[A-Za-z0-9\-]*         # o número con letras/códigos (3500LB, 704A41811035)
        )
        (?:
            (?:[\s\-\u2010]+            # incluye guion normal y guion unicode
            (?:
                [A-Z]{2,} |
                [A-Z][a-z]+ |
                [a-z]+ |
                \d+[A-Za-z0-9\-]*
            ))
        ){0,7}
        (?:\s+[sS])?                    # << acepta 's' o 'S' al final - quedan letras sueltas por problemas con el tokenizador
        (?:\s+[yY])?                    # << acepta 'y' o 'Y' al final - quedan letras sueltas por problemas con el tokenizador
    )
    $
    """,
    re.VERBOSE
)
```

Figura 35: Expresión regular para filtrar las entidades CANDIDATO_PARTE
Fuente: Código propio

3.2.1.4 Obtener listado de entidades filtradas

El modelo dispone en este punto de las entidades extraídas del texto que pueden ser modelos de aeronaves y partes afectadas, así como de las expresiones regulares para comparar dichas entidades y filtrar y descartar los falsos positivos.
A través del código de la figura 36:

```
modelos_detectados = set()

for subtree in chunked_app.subtrees():
    if subtree.label() == "CANDIDATO_MODELO":
        palabras = [token for token, tag in subtree]
        texto = " ".join(palabras)

        match = patron_modelo_general.search(texto)
        if match:
            modelo_base = match.group(1).replace(" ", "").upper()
            variante = match.group(2)

            if variante:
                modelo = f"{modelo_base} {variante.upper()}"
            else:
                modelo = modelo_base

            modelos_detectados.add(modelo)

# Mostrar modelos detectados
print("Modelos de aeronaves detectados:")
for modelo in sorted(modelos_detectados):
    print(f"- MODEL: {modelo}")
```

Figura 36: Código que compara las entidades obtenidas con la expresión regular para descartar falsos positivos
Fuente: Código propio

Para cada entidad CANDIDATO_MODELO detectado, se comprueba si coincide con la expresión regular patron_modelo_general (líneas 114 a 119); es decir, que la entidad parezca ser un modelo válido.

Si se produce coincidencia, el modelo estandariza el texto (líneas 120 a 127: elimina espacios, convierte en mayúsculas, etc.) y lo guarda en el listado definido modelos_detectados (línea 129).

Por último, el modelo muestra por pantalla los modelos detectados en la AD (líneas 131 a 134), tal y como se muestra en la Figura 37:

```
Modelos de aeronaves detectados:
- MODEL: AS350 B2
- MODEL: AS350 B3
- MODEL: AS355 N
- MODEL: AS355 NP
- MODEL: EC130 B4
```

Figura 37: Resultados mostrados por pantalla tras ejecutar el código de la Figura 36

Fuente: Código propio

Para el caso de cada entidad CANDIDATO_PARTE, el modelo funciona de forma similar. A través del código de la figura 38:

```
241 partes_detectadas = []
242
243 # Buscar chunks con etiqueta CANDIDATO_PARTE
244 for parte_limpia, chunked in todos_los_chunks:
245     for subtree in chunked.subtrees():
246         if subtree.label() == "CANDIDATO_PARTE":
247             palabras = [token for token, tag in subtree]
248             if len(palabras) < 2:
249                 continue
250
251             texto_parte = " ".join(palabras).strip()
252
253             if re.search(r"\b(part number|p/n|helicopters|manufacturer|applicable|document|
254                 continue
255             if re.fullmatch(r"[A-Z0-9\-\/_]+", texto_parte):
256                 continue
257
258             texto_parte_normalizado = texto_parte.strip().title()
259
260             # Aplicar patrón general de partes
261             match = patron_parte_general.fullmatch(texto_parte_normalizado)
262             if not match:
263                 continue
264
265             texto_parte_limpio = match.group(1).strip()
```

```
267             # Limpiar paréntesis, dobles espacios y normalizar
268             texto_parte_limpio = re.sub(r"\([^)]*\)", "", texto_parte_limpio)
269             texto_parte_limpio_norm = re.sub(r"\s+", " ", texto_parte_limpio).strip()
```

```
292                 # Agregar parte detectada con su(s) P/N
293                 partes_detectadas.append((texto_parte_limpio_norm.strip(), pn_list))
294
295 # ========================
296 # Mostrar resultados
297 # ========================
298 print("\nPartes afectadas detectadas:")
299 for parte, pn_list in partes_detectadas:
300     print(f"- PART: {parte} | P/Ns: {', '.join(pn_list)}")
```

Figura 38: Código que compara las entidades obtenidas con la expresión regular para descartar falsos positivos

Fuente: Código propio

Para cada entidad CANDIDATO_PARTE detectado, se comprueba si coincide con la expresión regular patron_parte_general (líneas 244 a 261); es decir, que la entidad parezca ser un modelo válido. En este caso, se añaden condiciones adicionales especiales para descartar falsos positivos (líneas 253 a 256), ya que la expresión regular patron_parte_general interpreta algunos falsos positivos como correctos y los muestra como partes extraídas de la AD.

Si se produce coincidencia, el modelo estandariza el texto (líneas 268 y 269: elimina espacios, limpia paréntesis, etc.) y lo guarda en el listado definido partes_detectadas (línea 293).

Por último, el modelo muestra por pantalla las partes afectadas detectadas en la AD (líneas 298 a 300), tal y como se muestra en la Figura 39:

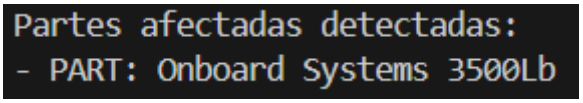


Figura 39: Resultados mostrados por pantalla tras ejecutar el código de la Figura 38
Fuente: Código propio

Tal y como se puede observar, el resultado obtenido solo muestra la entidad CANDIDATO_PARTE detectada de forma correcta (Figura 32), mientras que descarta las dos entidades CANDIDATO_PARTE detectados de forma errónea.

3.2.2 Detección de número de serie de aeronaves y P/N de partes afectadas incluidas en el texto de una AD.

3.2.2.1 Extraer el número de serie de la aeronave

De forma adicional al modelo de aeronave, es necesario que nuestro modelo también extraiga la información del número de serie de la aeronave para poder evaluar si una AD le es de aplicación o no a las aeronaves dentro de nuestra organización CAMO.

El número de serie de aeronave no se extrae usando el mismo método que el modelo de aeronave, ya que no se hace uso de herramientas para el procesamiento de lenguaje natural.

En su lugar, se realiza una búsqueda de palabras concretas en el texto para obtener esta información, tal y como se muestra en la figura 40:

```
# ========================
# Detección de números de serie de aeronaves
# ========================

seriales_detectados = set()

# Casos tipo "all serial numbers" o "all s/n"
if re.search(r"all[\s\n]+(serial numbers|\(s/n\)|s/n)", texto_applicability, re.IGNORECASE):
    seriales_detectados.add("ALL SERIAL NUMBERS")

# Rangos tipo "s/n 0001 through 2064"
re_rango_sn = re.compile(
    r"(serial numbers?\s*\(s/n\)|s/n)[^\dA-Z]{0,10}(\d{3,6})\s*(?:through|to|-)\s*(\d{3,6})",
    re.IGNORECASE
)
for _, inicio, fin in re_rango_sn.findall(texto_applicability):
    seriales_detectados.add(f"RANGE: {inicio} to {fin}")

# Detección de números de serie individuales tipo "s/n 1234, 1235"
re_individual_sn = re.compile(
    r"(?:serial numbers?\s*\(s/n\)|s/n)[\s:]*((?:\d{3,6}(?:,\s*)?)+)",
    re.IGNORECASE
)
for _, lista in re_individual_sn.findall(texto_applicability):
    numeros = [num.strip() for num in lista.split(",") if num.strip()]
    for numero in numeros:
        seriales_detectados.add(f"S/N: {numero}")

# Mostrar resultados de números de serie
print("\nNúmeros de serie de aeronaves detectados:")
for sn in sorted(seriales_detectados):
    print(f"- {sn}")
```

Figura 40: Código para la detección de los números de serie de las aeronaves.

Fuente: Código propio

Donde:

- en la línea 140 se crea la variable seriales_detectados donde almacenar todos los números de serie detectados;
- en las líneas 143 y 144, si en la variable texto_applicability (variable donde se guarda el texto extraído de Aplicabilidad de la AD) aparece el texto "all serial numbers" o "all s/n", el código extrae y guarda el número de serie ALL SERIAL NUMBERS en la variable seriales_detectados;
- en las líneas 147 a 152, si en la variable texto_applicability aparece un rango de números de serie (aparece el texto "serial numbers through/to"), el código extrae y guarda el rango de números de serie detectados en la variable seriales_detectados;
- en las líneas 155 a 162, si en la variable texto_applicability aparece un número o números de serie únicos definidos, el código extrae y guarda el los números de serie detectados en la variable seriales_detectados;
- finalmente, en las líneas 165 a 167 se muestra por pantalla los datos almacenados en la variable seriales_detectados (Figura 41).

```
Números de serie de aeronaves detectados:
- ALL SERIAL NUMBERS
```

Figura 41: Resultados mostrados por pantalla tras ejecutar el código de la Figura 40
Fuente: Código propio

3.2.2.2 Extraer el P/N de la Parte Afectada

Por último, es necesario que nuestro modelo extraiga los P/N de las partes afectadas de una AD ya que éstos actúan como identificador único de las partes afectadas (ya que el nombre puede no coincidir entre la AD y la base de datos).

Del mismo modo que ocurre con el número de serie de una aeronave, el P/N no se extrae haciendo uso de herramientas para el procesamiento de lenguaje natural.

En su lugar, se realiza una búsqueda de palabras concretas en el texto próximo a donde se ha detectado la parte afectada para obtener esta información, tal y como se muestra en la figura 42:

```
272             # Buscar contexto donde aparece la Parte Afectada para extraer los P/N próximos
273             affected_norm = re.sub(r"\s+", " ", parte_limpia).strip()
274             contexto = re.search(re.escape(texto_parte_limpio_norm) + r".{0,300}", affected_norm, re.IGNORECASE)
275
276             if contexto:
277                 fragmento = contexto.group()
278
279                 # Buscar múltiples P/N en el fragmento
280                 pn_matches = re.findall(
281                 r"\b(?:P\s*/\s*N|P\s*N|Part\s*N\s*umber|Part\s*Number)\s*[:\-]?\s*([A-Z0-9][A-Z0-9\-\/_]{2,})(?=\b|[.,\s])",
282                 fragmento,
283                 re.IGNORECASE
284             )
285
286                 if pn_matches:
287                     pn_list = list({pn.upper() for pn in pn_matches})  # eliminar duplicados
288                 elif re.search(r"\ball\b", fragmento, re.IGNORECASE):
289                     pn_list = ["ALL"]
290                 else:
291                     pn_list = ["UNKNOWN"]
292
293                 # Agregar parte detectada con su(s) P/N
294                 partes_detectadas.append((texto_parte_limpio_norm.strip(), pn_list))
295
296 # ========================
297 # Mostrar resultados
298 # ========================
299 print("\nPartes afectadas detectadas:")
300 for parte, pn_list in partes_detectadas:
301     print(f"- PART: {parte} | P/Ns: {', '.join(pn_list)}")
```

Figura 42: Código para la detección de los P/N de las partes afectadas.

Fuente: Código propio

Donde:

- en la línea 274 se crea la variable contexto, que se define como los 300 carácteres posteriores a la parte afectada detectada y almacenada en la variable texto_parte_limpio_norm;
- en las líneas 280 a 283, se define la variable pn_matches donde almacenar todos P/N localizados si en el texto se encuentran las palabras "P/N" o "Part Number";
- en las líneas 286 y 287 se indica que, siempre que la variable pn_matches contenga datos, se define la variable pn_list con el listado de todos los P/N localizados y almacenados en la variable pn_matches;
- en las líneas 288 y 289 se indica que, si se encuentra la palabra "all" en el contexto de la parte afectada, la variable pn_list tomará el valor de "ALL";
- en las líneas 290 y 291 se indica que, en caso de no encontrarnos en ninguno de los dos casos anteriores, la variable pn_list tomará el valor de "UNKNOWN";
- finalmente, en las líneas 299 a 301 se muestra por pantalla los datos almacenados en la variable pn_list (Figura 43).

```
P/Ns: 704A41811037, 704A41811035
```

Figura 43: Resultados mostrados por pantalla tras ejecutar el código de la Figura 42

Fuente: Código propio

4. Desarrollo del código del modelo para llevar a cabo la comprobación de si una AD afecta a alguna de las aeronaves incluidas en una Organización CAMO

El resultado que nos devuelve el desarrollo del modelo para la extracción y procesamiento de los datos de la AD se muestra en la Figura 44:

```
Modelos de aeronaves detectados:
- MODEL: AS350 B2
- MODEL: AS350 B3
- MODEL: AS355 N
- MODEL: AS355 NP
- MODEL: EC130 B4

Números de serie de aeronaves detectados:
- ALL SERIAL NUMBERS

Partes afectadas detectadas:
- PART: Onboard Systems 3500Lb | P/Ns: 704A41811037, 704A41811035
- PART: Dropping Control | P/Ns: 704A41811037
```

Figura 43: Resultados de la extracción y procesamiento de datos de la AD 2025-0025 mostrados por pantalla
Fuente: Código propio

Con la extracción y almacenamiento de los datos de una AD de forma satisfactoria inicia el desarrollo de la FASE 2, donde se compara dicha información extraída de las AD con la información almacenada de las aeronaves en la base de datos (BBDD).

Al finalizar esta segunda fase, el modelo debe indicar si es necesario revisar las acciones requeridas en la AD para alguna de las aeronaves gestionadas por la Organización CAMO.

4.1 Creación de la base de datos de las aeronaves

Para el desarrollo de este TFG se ha elaborado una BBDD que recoge la información de las aeronaves mostrada en la Figura 44:

```
matricula,modelo,sn,parte,pn
PH-NEU,EC120 B,3256,Rotor Brake,350A32-0500-02
PH-NEU,EC120 B,3256,Tail rotor hub body,C642A0100103
EC-SQE,EC135 T3,0298,Warning unit,ALL
EC-KLM,EC130 T2,1498,Main rotor blade,A710A0101101
N508VJ,AS350 B3,3369,Engine control unit,350A33-2167-07
SE-DBZ,H120,5578,Fuel pump,B123X7890004
```

Figura 44: Ejemplo de BBDD de aeronaves que recoge información sobre matrícula, modelo, número de serie de aeronave, parte afectada y P/N
Fuente: BBDD de elaboración propia

Para poder implementar el modelo de un modo práctico y funcional, se ha desarrollado una BBDD siguiendo el modelo de la Figura 44 con aeronaves pertenecientes a la CAMO de Heliswiss.

En dicha BBDD se han añadido todas las partes identificadas de las aeronaves con su correspondiente P/N. No obstante, al tratarse de información no accesible por parte de terceros, no es posible compartir esa información en este TFG.

Una vez creada la BBDD, hemos de cargarla en el modelo para que éste pueda trabajar con ella (Figura 45):

```
58  # ========================
59  # Cargar BBDD de aeronaves
60  # ========================
61
62  df = pd.read_csv("BBDD_Aeronaves_CAMO.csv")
```

Figura 45: Carga de la BBDD de aeronaves de la Organización CAMO en el modelo
Fuente: Código propio

4.2 Comprobar si una AD afecta a alguna de las aeronaves incluidas en la Organización CAMO

El modelo dispone en este punto de la información necesaria para llevar a cabo la comprobación de si una AD afecta a alguna de las aeronaves incluidas en la Organización CAMO.

Por un lado, el modelo dispone de los INPUTS extraídos de la AD: modelos de aeronaves detectados, números de serie de aeronaves detectados, partes afectadas y sus P/N. Por otro lado, dispone de los datos de matrícula, modelo, número se serie, partes y sus P/N de las aeronaves gestionadas por la Organización CAMO.

A través del código de la Figura 46, el modelo compara y busca coincidencias entre los INPUTS extraídos de la AD y la información de las aeronaves contenida en la BBDD:

```python
# Filtrar base de datos por coincidencia de modelo
for modelo_detectado in modelos_detectados:
    df_modelo = df[df['modelo'].str.upper() == modelo_detectado.upper()]

    if df_modelo.empty:
        continue

    for sn_detectado in seriales_detectados:
        sn_encontrado = False

        for _, row in df_modelo.iterrows():
            sn_bbdd = str(row['sn']).strip()
            pn_bbdd = str(row['pn']).strip().upper()
            matricula_bbdd = str(row['matricula']).strip()
            coincidencia_sn = False

            # Verificación del número de serie
            if sn_detectado.upper() == "ALL SERIAL NUMBERS" or sn_bbdd.upper() in ["ALL SERIAL NUMBERS", "ALL"]:
                coincidencia_sn = True
            elif "-" in sn_detectado:
                try:
                    inicio, fin = map(int, sn_detectado.replace("RANGE:", "").replace("TO", "-").replace("to", "-").split("-"))
                    if inicio <= int(sn_bbdd) <= fin:
                        coincidencia_sn = True
                except ValueError:
                    print(f"⚠️ Rango inválido en el número de serie detectado: {sn_detectado}")
            elif sn_detectado == sn_bbdd:
                coincidencia_sn = True
```

```python
            # Si el número de serie coincide
            if coincidencia_sn:
                sn_encontrado = True

                # Comparar con todos los P/N detectados
                pn_coincide = False
                for pn in pn_detectado:
                    if pn_bbdd.upper() == "ALL" or pn_bbdd.upper() == pn.upper():
                        pn_coincide = True
                        coincidencias_encontradas.append({
                            "Matrícula": matricula_bbdd,
                            "Modelo": modelo_detectado,
                            "S/N directiva": sn_detectado,
                            "S/N BBDD": sn_bbdd,
                            "P/N directiva": pn,
                            "P/N BBDD": pn_bbdd
                        })
                        break  # Ya encontramos coincidencia de P/N
```

Figura 46: Código para la búsqueda de coincidencias entre los INPUTS extraídos de las AD y la información almacenada en la BBDD

Fuente: Código propio

Según el resultado obtenido al ejecutar el código de la Figura 46, el sistema indicará al técnico de la oficina CAMO (Figura 47):

- **sí** ha encontrado alguna coincidencia: en este caso, le mostrará al técnico las matrículas de las aeronaves para las cuales se deben revisar las acciones a programar indicadas en la AD
- **no** ha encontrado ninguna coincidencia: en este caso, únicamente le mostrará al técnico un mensaje conforme no se ha encontrado ninguna coincidencia.

```
if coincidencias_encontradas:
    df_resultado = pd.DataFrame(coincidencias_encontradas)
    print("\n📋 Coincidencia encontrada en el listado de aeronaves gestionadas por la Organización CAMO:")
    print(df_resultado.to_string(index=False))
    print("\n La AD afecta a las aeronaves con las matrículas anteriormente indicadas. Revisar acciones a programar")
else:
    print("\n❌ No se encontró ninguna coincidencia completa entre modelo, número de serie y número de parte en las " \
    "aeronaves gestionadas por la Organización CAMO.")
```

Figura 47: Código para mostrar los resultados al técnico de la CAMO según se hayan encontrado o no coincidencias entre los INPUTS y la BBDD.

Fuente: Código propio

El resultado final que nos devuelve el modelo y nos muestra en pantalla se muestra en la Figura 48:

```
📋 Coincidencia encontrada en el listado de aeronaves gestionadas por la Organización CAMO:
Matrícula   Modelo      S/N directiva S/N BBDD P/N directiva     P/N BBDD
   EC-LXO AS350 B3 ALL SERIAL NUMBERS     7625  704A41811035 704A41811035
   EC-MXT AS350 B3 ALL SERIAL NUMBERS     8502  704A41811035 704A41811035
   EC-NDT AS350 B3 ALL SERIAL NUMBERS     8670  704A41811035 704A41811035
   EC-OBG AS350 B3 ALL SERIAL NUMBERS     9373  704A41811035 704A41811035
   EC-OHS AS350 B3 ALL SERIAL NUMBERS     4726  704A41811035 704A41811035

 La AD afecta a las aeronaves con las matrículas anteriormente indicadas. Revisar acciones a programar
```

Figura 48: Resultado mostrado por pantalla de ejecutar el código de las Figuras 46 y 47.

Fuente: Código propio

Como se puede observar, el modelo muestra todas las matrículas en las que se produce una coincidencia con INPUTS extraidos de la AD.

5. Implementación del modelo en un escenario real: Organización CAMO Heliswiss

En este capítulo del TFG se quiere poner a prueba el modelo programado en los capítulos anteriores, observando los resultados obtenidos en el análisis de varias Directivas de Aeronavegabilidad y comparando dichos resultados con el análisis realizado por el personal técnico de la Organización CAMO de Heliswiss.

Actualmente, la CAMO de Heliswiss gestiona la aeronavegabilidad de varios modelos de aeronaves, destacando modelos de los fabricantes Airbus Helicopters, Cirrus y Robinson Helicopters.

Para llevar a cabo la implementación del modelo en un escenario real, se escoge la flota de aeronaves de Airbus Helicopters gestionadas por la Organización CAMO de Heliswiss.

La flota de aeronaves Airbus Helicopters supone el 44% de las aeronaves gestionadas por la organización CAMO, siendo la flota con mayor presencia en la organización de gestión de la aeronavegabilidad. Los modelos incluidos dentro de la flota de Airbus Helicopters son principalmente el AS350 B3, EC130 y EC135.

Las características del escenario real que se plantea en este apartado son:

- se analizan las Directivas de Aeronavegabilidad de los últimos 5-7 años de los modelos de aeronave AS350 B3, EC130 y EC135. No se analizan Directivas anteriores a 2018 ya que éstas no mantienen la estructura observada en el capítulo 2 de esta memoria.
- se analizan los resultados obtenidos por el modelo (la Directiva afecta o no a alguna de las aeronaves incluida en la organización CAMO).
- se comparan los resultados obtenidos a través del modelo con el análisis de las Directivas de Aeronavegabilidad realizado por el personal técnico de la oficina CAMO.

5.1 Análisis de Directivas de Aeronavegabilidad - resultados obtenidos

En esta primera parte del escenario planteado, se han analizado un total de:

- 28 Directivas de Aeronavegabilidad para los modelos de aeronave AS350 B3 y EC130 (ya que las AD son las mismas para ambos modelos de helicóptero).
- 18 Directivas de Aeronavegabilidad para el modelo de aeronave EC135.

La Figura 49 muestra una tabla resumen con el resultado obtenido de este primer análisis:

	Nº Directiva	Análisis realizado correctamente por el modelo
1	2025-0062	SI
2	2023-0187R1	SI
3	2025-0036	SI
4	2025-0025	SI
5	2021-0282R1	SI
6	2024-0232	SI
7	2024-0144	SI
8	2024-0139	NO
9	2024-0133	SI
10	2022-0150R1	SI
11	2024-0113	SI
12	2024-0018	NO
13	2023-0214	NO
14	2023-0166	NO
15	2023-0131	SI
16	2023-0127	NO
17	2023-0107	NO
18	2023-0064	SI
19	2023-0044	NO
20	2022-0246	SI
21	2022-0128	SI
22	2022-0051	NO
23	2022-0053	NO
24	2021-0216	SI
25	2021-0195	SI
26	2021-0023	SI
27	2020-0187	SI
28	2017-0089R1	SI
29	2025-0113	SI
30	2025-0108	SI
31	2025-0051R1	NO
32	2025-0055	SI
33	2024-0249	SI
34	2024-0028R1	SI
35	2023-0197	NO
36	2023-0066	SI
37	2022-0097	SI
38	2022-0023	NO
39	2021-0149	SI
40	2021-0050	SI
41	2021-0011	SI
42	2020-0282	SI
43	2020-0105	SI
44	2020-0099	SI
45	2020-0064	NO
46	2019-0199	SI

Figura 49: Resultado del análisis de Directivas de Aeronavegabilidad con el modelo.

Fuente: Elaboración propia.

Los resultados obtenidos de este primer ejercicio muestran que, de un total de 46 Directivas analizadas, el modelo ha analizado bien un total de 33 AD y ha presentado fallo en 13, **por lo que en el 71% de los casos el modelo ha funcionado.**

Este primer ejercicio realizado por el modelo nos permite analizar los 3 principales errores y/o limitaciones que presenta el mismo:

- si los P/N de las partes afectadas no se especifican en el apartado de partes afectadas (se indican, por ejemplo, en un Anexo o en el campo de Aplicabilidad de la Directiva), el modelo no es capaz de extraer dicho P/N y buscarlo en la BBDD de las aeronaves.
- si los P/N de las partes afectadas sí se especifican en el apartado de partes afectadas, pero se hace con un formato tipo tabla, el modelo no es capaz de extraer dicho P/N y buscarlo en la BBDD de las aeronaves.
- si el número de serie de las aeronaves contiene alguna excepción (es decir, se indica algo como "todos los números de serie excepto" o "los números de serie en este rango excepto"), el modelo no es capaz de procesar dicha excepción y falla en la extracción de los modelos de aeronaves.

Para el error en los dos primeros casos, se podría modificar el modelo para que la detección y extracción de números de serie de la AD se realizase del mismo modo que los modelos de aeronaves y partes afectadas, definiendo una entidad denominada CANDIDATO_PN que extrajera todos los P/N del texto de la Directiva.

Como los P/N guardan todos una estructura gramatical similar (como ocurre con los modelos de aeronaves), esta forma de modificar el modelo debería permitir solucionar en gran medida el problema con la detección de los P/N que no se encuentran en el texto de partes afectadas.

5.2 Comparando los resultados obtenidos del modelo con el análisis realizado por los técnicos

A través de los resultados obtenidos en el ejercicio previo, descartamos todas las Directivas de Aeronavegabilidad que el modelo no es capaz de analizar correctamente.

Posteriormente, analizamos el resultado de aplicabilidad obtenido por el modelo para cada matrícula (se han modificado las matrículas reales de las aeronaves por tratarse de información reservada) de la organización CAMO y lo comparamos con los resultados del análisis de la Directiva realizado por los técnicos de la organización CAMO de Heliswiss.

Los resultados obtenidos se muestran en la Figura 50 para las aeronaves AS350 B3 y EC130 y en la Figura 51 para las aeronaves EC135:

Nº Directiva	Afecta Aeronaves gestionadas por la CAMO	AERONAVES CAMO								Coincide con análisis técnico CAMO
		EC-AAA	EC-BBB	EC-CCC	EC-DDD	F-AAAA	EC-EEE	EC-FFF	EC-GGG	
2025-0062	NO									SI
2023-0187R1	NO									NO
2025-0036	NO									SI
2025-0025	SI	X	X	X	X		X			SI
2021-0282R1	SI					X	X			NO
2024-0232	NO									SI
2024-0144	SI								X	SI
2024-0133	NO									N/A
2022-0150R1	SI							X		SI
2024-0113	SI							X		SI
2023-0131	NO									SI
2023-0064	NO									NO
2022-0246	NO									SI
2022-0128	NO									NO
2021-0216	NO									SI
2021-0195	NO									SI
2021-0023	NO									SI
2020-0187	SI							X		SI
2017-0089R1	NO									NO

Figura 50: Comparativa de los resultados obtenidos por el modelo vs análisis realizado por los técnicos de la organización CAMO para los modelos de aeronave AS350 B3 y EC130

Fuente: Elaboración propia.

Nº Directiva	Afecta Aeronaves gestionadas por la CAMO	AERONAVES CAMO		Coincide con análisis técnico CAMO
		EC-HHH	EC-III	
2025-0113	NO			SI
2025-0108	NO			NO
2025-0055	NO			SI
2024-0249	NO			SI
2024-0028R1	SI		X	SI
2023-0066	NO			SI
2022-0097	SI			SI
2021-0149	NO			NO
2021-0050	SI	X	X	SI
2021-0011	NO			SI
2020-0282	SI	X	X	SI
2020-0105	SI	X	X	SI
2020-0099	NO			NO
2019-0199	NO			NO

Figura 51: Comparativa de los resultados obtenidos por el modelo vs análisis realizado por los técnicos de la organización CAMO para los modelos de aeronave AS350 B3 y EC135

Fuente: Elaboración propia.

Los resultados obtenidos al aplicar nuestro modelo al escenario real muestran que, de un total de 33 Directivas analizadas, el resultado mostrado por el modelo coincide con el análisis de los técnicos de la CAMO en un total de 24 ocasiones, por lo que **hay una coincidencia del 72% en los resultados obtenidos por los técnicos de la CAMO.**

El principal error detectado en el escenario real es el siguiente:

- en la base de datos de aeronaves, para cada aeronave, no se incorporan todas las partes de las mismas, sino que únicamente se incorporan aquellas partes con características especiales (vida límite, máximo número de usos, etc.).

Al no incorporar la información de todas las partes en la BBDD para cada aeronave, el modelo no es capaz de detectar la parte en la BBDD y, por tanto, indica que la Directiva no aplica a ninguna de las aeronaves gestionadas por la CAMO.

En entrevista con personal técnico de la CAMO nos indica que esto ocurre debido a la gran cantidad de partes que componen una aeronave. Incluir todas las partes de una aeronave en la base de datos de la CAMO supondría un tiempo elevado, y actualmente no es necesario.

Como solución a este problema, se comenta con el técnico de la organización CAMO que en el futuro la inclusión de partes de una aeronave en la base de datos se produjera de forma automática, cogiendo como fuente de información la documentación de la aeronave y la documentación específica de mantenimiento.

6. Conclusiones

Tal y como se indica en el capítulo 1 de esta memoria, el objetivo general de este trabajo es explorar la introducción de nuevas aplicaciones procedentes del desarrollo de la IA en el estudio de la aplicabilidad de Directivas de Aeronavegabilidad para las aeronaves gestionadas por una organización CAMO.

Tras la elaboración del modelo a través de aplicaciones de la IA como el NLP, los resultados obtenidos en el escenario real muestran que **hay una coincidencia del 72% entre los resultados obtenidos por nuestro modelo y los resultados obtenidos por los técnicos de la CAMO,** por lo que el modelo resulta práctico y funcional.

A continuación se enumeran las conclusiones que se pueden extraer de este TFG:

1. Las aplicaciones procedentes del desarrollo de la IA, tales como el procesamiento de lenguaje natural, pueden ayudar al personal de una organización CAMO a realizar su trabajo de gestión de la aeronavegabilidad de las aeronaves.
2. Al comparar datos de forma automática, se eliminan posibles errores producidos por factores humanos (error al revisar la información, por ejemplo).
3. El uso de aplicaciones procedentes de la IA pueden ayudar a digitalizar y llevar a cabo las tareas de forma más efectiva; no solo en el estudio de aplicabilidad de Directivas, sino también en otros ámbitos de la gestión de la aeronavegabilidad (elaboración programas de mantenimiento a partir de la documentación elaborada por los fabricantes, trazabilidad de revisiones, etc.)

7. Referencias y Bibliografía

[1] European Union Aviation Safety Agency. (s.f.). Continuing airworthiness. Recuperado el 26 de junio de 2025, de https://www.easa.europa.eu/en/regulations/continuing-airworthiness

[2] AVSAF. (s.f.). Introducción a los factores humanos. Recuperado el 26 de junio de 2025, de https://avsaf.es/post/introduccion-factores-humanos/

[3]ManpowerGroup España. (s.f.). La mitad de las empresas españolas ya han adoptado el uso de la inteligencia artificial. Recuperado el 26 de junio de 2025, de https://www.manpowergroup.es/notas-de-prensa/la-mitad-de-las-empresas-espanolas-ya-han-adoptado-el-uso-de-la-inteligencia-artificial

[4] Slack. (s.f.). Cómo las empresas están utilizando la inteligencia artificial. Recuperado el 26 de junio de 2025, de https://slack.com/intl/es-es/blog/transformation/inteligencia-artificial-en-las-empresas

[5] PwC España. (s.f.). AI Jobs Barometer. Recuperado el 26 de junio de 2025, de https://www.pwc.es/es/consultoria/inteligencia-artificial/ai-jobs-barometer.html

[6] Python Software Foundation. (s.f.). Python.org. Recuperado el 26 de junio de 2025, de https://www.python.org/

[7] Python Software Foundation. (s.f.). Python en español. Recuperado el 26 de junio de 2025, de https://es.python.org/

[8] Reyes Ochoa, X. (s.f.). Aprende Python desde cero hasta avanzado. Book Shelter Gmbh, 2024.

[9] European Union Aviation Safety Agency. (s.f.). Airworthiness Directives. Recuperado el 26 de junio de 2025, de https://ad.easa.europa.eu/

[10] Python Software Foundation. (s.f.). PyPI – The Python Package Index. Recuperado el 26 de junio de 2025, de https://pypi.org

[11] Bird, S., & Loper, E. (2004, julio). Proceedings of the ACL demonstration session (pp. 214–217). Association for Computational Linguistics. https://aclanthology.org/P04-5010/

[12] Bird, S., Klein, E., & Loper, E. (2019). Natural language processing with Python – Analyzing text with the Natural Language Toolkit. https://www.nltk.org/book/

Anexo 1 - Código completo del modelo

```
import PyPDF2
import nltk
import re
import pandas as pd
from nltk.tokenize import regexp_tokenize
from nltk import pos_tag, RegexpParser

# ==========================
# Función para extraer texto de un PDF
# ==========================

def extraer_texto_pdf(Directiva):
    texto_completo = ""

    # Abrir el archivo PDF en modo lectura binaria
    with open(Directiva, "rb") as archivo_pdf:
        lector_pdf = PyPDF2.PdfReader(archivo_pdf)

        # Recorrer todas las páginas del PDF
        for num_pagina in range(len(lector_pdf.pages)):
            pagina = lector_pdf.pages[num_pagina]
            texto_completo += pagina.extract_text() + "\n"

    return texto_completo

# Directiva a procesar
Directiva = "Incluir aquí la Directiva a procesar.pdf"

# Extraer texto del PDF
texto_extraido = extraer_texto_pdf(Directiva)

# ==========================
# Tokenizar todo el texto - se analiza con NLTK
# ==========================

tokens = regexp_tokenize(texto_extraido, r"\w+(?:-\w+)*|\S") # Tokenizar el texto sin separar guiones
tags = pos_tag(tokens) # Etiquetar las partes del habla del texto extraído completo (POS tagging)

# ==========================
# Aplicar chunking general
# ==========================

grammar = r"""
    CANDIDATO_MODELO:
        {<IN><CD><NNP|NN>}          # Para AS 350 B2, ya AS está como IN (preposición en inglés)
```

```
        {<NNP><CD><NNP>}            # Para EC 130 B4, cuando sí detecta EC como NNP
        {<NN><CD><NN>}
        {<NNP><NNP>}
        {<NNP>}
"""

chunk_grammar = r"""
    CANDIDATO_PARTE:
            {<JJ|NN|NNS|NNP|NNPS|CD|RB|VBD|VBZ|VBG>{2,7}}   # al menos dos palabras sustantivas o adjetivos
"""

# ==========================
# Cargar BBDD de aeronaves
# ==========================

df = pd.read_csv("BBDD_Aeronaves_CAMO.csv")

# ==========================
# Detección de modelos de aeronaves
# ==========================

#Corregir algunos fallos de nltk a la hora de extraer el texto
texto_extraido = re.sub(r'Definitio\s+ns', 'Definitions', texto_extraido, flags=re.IGNORECASE)

# Extraer bloque "Applicability" a "Definitions"
match = re.search(r"Applicability(.*?)Definitions", texto_extraido, re.DOTALL | re.IGNORECASE)
if match:
    texto_applicability = match.group(1)
else:
    texto_applicability = texto_extraido

# Limpiar saltos de línea
texto_applicability = texto_applicability.replace("\n", " ")
texto_applicability = re.sub(r'\s+', ' ', texto_applicability)

# Tokenizar solo el bloque
tokens_app = regexp_tokenize(texto_applicability, r"\w+(?:-\w+)*|\S")

# Etiquetar solo el bloque
tags_app = pos_tag(tokens_app)

# Chunking sobre el bloque de Applicability - CANDIDATO_MODELO
chunk_parser = RegexpParser(grammar)      # chunking con una expresión regular que estamos definiendo en "grammar"
chunked_app = chunk_parser.parse(tags_app)

# Regex para detectar patrones típicos de modelos de aeronaves
patron_modelo_general = re.compile(
```

```
    r"""
    \b
    (                           # Grupo para el prefijo (modelo base)
        [A-Z]{2,4}                  # Prefijo, 2 a 4 letras mayúsculas
        \s?                      # Espacio opcional
        \d{2,3}                     # Número, 2 o 3 dígitos
    )
    (?:[\s\-]+                      # Separador (espacio o guion)
      ([A-Z0-9\-\+]{1,5}[a-z]?))?         # Variante: letras mayúsculas, números, guion, +, y opcional letra minúscula al final
    \b
    """,
    re.IGNORECASE | re.VERBOSE
)

# Filtrar entidades que contienen al menos una palabra con número
modelos_detectados = set()

for subtree in chunked_app.subtrees():
    if subtree.label() == "CANDIDATO_MODELO":
        palabras = [token for token, tag in subtree]
        texto = " ".join(palabras)

        match = patron_modelo_general.search(texto)
        if match:
            modelo_base = match.group(1).replace(" ", "").upper()  # quita espacios
            variante = match.group(2)

            if variante:
                modelo = f"{modelo_base} {variante.upper()}"
            else:
                modelo = modelo_base

            modelos_detectados.add(modelo)

# Mostrar modelos detectados
print("Modelos de aeronaves detectados:")
for modelo in sorted(modelos_detectados):
    print(f"- MODEL: {modelo}")

# ==========================
# Detección de números de serie de aeronaves
# ==========================

seriales_detectados = set()

# Casos tipo "all serial numbers" o "all s/n"
if re.search(r"all[\s\n]+(serial numbers|\(s/n\)|s/n)", texto_applicability, re.IGNORECASE):
    seriales_detectados.add("ALL SERIAL NUMBERS")
```

```
# Rangos tipo "s/n 0001 through 2064"
re_rango_sn = re.compile(
    r"(serial numbers?\s*\(s/n\)|s/n)[^\dA-Z]{0,10}(\d{3,6})\s*(?:through|to|-)\s*(\d{3,6})",
    re.IGNORECASE
)
for _, inicio, fin in re_rango_sn.findall(texto_applicability):
    seriales_detectados.add(f"RANGE: {inicio} to {fin}")

# Detección de números de serie individuales tipo "s/n 1234, 1235"
re_individual_sn = re.compile(
    r"(?:serial numbers?\s*\(s/n\)|s/n)[\s:]*((?:\d{3,6}(?:,\s*)?)+)",
    re.IGNORECASE
)
for _, lista in re_individual_sn.findall(texto_applicability):
    numeros = [num.strip() for num in lista.split(",") if num.strip()]
    for numero in numeros:
        seriales_detectados.add(f"S/N: {numero}")

# Mostrar resultados de números de serie
print("\nNúmeros de serie de aeronaves detectados:")
for sn in sorted(seriales_detectados):
    print(f"- {sn}")

# ==========================
# Detección de partes afectadas
# ==========================

# Buscar TODAS las líneas de partes afectadas (bloques separados por punto y doble salto de línea)
matches = re.findall(
    r"(Affected[^\n]*:\s*((?:.|\n)*?))\.\s*\n\s*\n",
    texto_extraido,
    re.IGNORECASE
)

# Almacenar todos los bloques encontrados en una lista
partes_affected = []
for encabezado, cuerpo in matches:
    texto_affectedpart = cuerpo.strip().lstrip(": ").strip()
    if texto_affectedpart:
        partes_affected.append(texto_affectedpart)

#print(partes_affected)

# Almacenar todos los chunks detectados
todos_los_chunks = []

# Compilar el patrón para validar nombres de partes
```

```
patron_parte_general = re.compile(
    r"""
    ^
    (
        (?:
            [A-Z]{2,}                # Palabra toda en mayúscula (ELEC)
            | [A-Z][a-z]+             # o capitalizada (Main, Rotor)
            | \d+[A-Za-z0-9\-]*         # o número con letras/códigos (3500LB, 704A41811035)
        )
        (?:
            (?:[\s\-\u2010]+           # incluye guion normal y guion unicode
            (?:
                [A-Z]{2,} |
                [A-Z][a-z]+ |
                [a-z]+ |
                \d+[A-Za-z0-9\-]*
            ))
        ){0,7}
          (?:\s+[sS])?                    # << acepta 's' o 'S' al final - quedan letras sueltas por problemas con el tokenizador
          (?:\s+[yY])?                    # << acepta 'y' o 'Y' al final - quedan letras sueltas por problemas con el tokenizador
    )
    $
    """,
    re.VERBOSE
)

for parte in partes_affected:
    # Limpiar texto ANTES de tokenizar
    parte_limpia = re.sub(r"\([^)]*\)", "", parte)  # elimina contenido entre paréntesis
    parte_limpia = re.sub(r"\s*-\s*", "-", parte_limpia)  # elimina espacios con guiones

    # Tokenizar y etiquetar
     parte_limpia = re.sub(r"[\u2010-\u2015]", "-", parte_limpia) #limpia algunos guiones mal interpretados por el tokenizador
    tokens_aff = regexp_tokenize(parte_limpia, r"\w+(?:-\w+)*|\S")
    tags_aff = pos_tag(tokens_aff)

    # Chunking sobre los bloques de Partes Afectadas - CANDIDATO_PARTE
     chunk_parser = RegexpParser(chunk_grammar)   # chunking con una expresión regular que estamos definiendo en "chunk_grammar"
    chunked_aff = chunk_parser.parse(tags_aff)
    todos_los_chunks.append((parte_limpia, chunked_aff))

# ==========================
# Extracción de partes afectadas + P/N
# ==========================

partes_detectadas = []
```

```
# Buscar chunks con etiqueta CANDIDATO_PARTE
for parte_limpia, chunked in todos_los_chunks:
    for subtree in chunked.subtrees():
        if subtree.label() == "CANDIDATO_PARTE":
            palabras = [token for token, tag in subtree]
            if len(palabras) < 2:
                continue

            texto_parte = " ".join(palabras).strip()

                                                                    if re.search(r"\b(part number|p/n|helicopters|manufacturer|applicable|document|status|european|appendix|affected|section|table|figure|installation|date|value|months|trimester|january|february|march|april|may|june|july|august|september|october|november|december)\b", texto_parte, re.IGNORECASE):# Ignorar si contiene términos genéricos o administrativos, como P/N, Part Number, Appendix, Section, etc.
                 continue
            if re.fullmatch(r"[A-Z0-9\-\/_]+", texto_parte):                # Ignorar si es un código puro (ej: 704A41811035 o ABC-1234)
                continue

            texto_parte_normalizado = texto_parte.strip().title()

            # Aplicar patrón general de partes
            match = patron_parte_general.fullmatch(texto_parte_normalizado)
            if not match:
                continue
            texto_parte_limpio = match.group(1).strip()

              # Limpiar paréntesis, dobles espacios y normalizar - se obtiene el nombre de la Parte Afectada
            texto_parte_limpio = re.sub(r"\([^)]*\)", "", texto_parte_limpio)
            texto_parte_limpio_norm = re.sub(r"\s+", " ", texto_parte_limpio).strip()



            # Buscar contexto donde aparece la Parte Afectada para extraer los P/N próximos
            affected_norm = re.sub(r"\s+", " ", parte_limpia).strip()
                        contexto = re.search(re.escape(texto_parte_limpio_norm) + r".{0,300}", affected_norm, re.IGNORECASE)

            if contexto:
                fragmento = contexto.group()

                # Buscar múltiples P/N en el fragmento
                pn_matches = re.findall(

r"\b(?:P\s*/\s*N|P\s*N|Part\s*N\s*umber|Part\s*Number)\s*[:\-]?\s*([A-Z0-9][A-Z0-9\-\/_]{2,})(?=\b|[.,\s])",
                fragmento,
```

```
            re.IGNORECASE
        )

            if pn_matches:
                pn_list = list({pn.upper() for pn in pn_matches})  # eliminar duplicados
            elif re.search(r"\ball\b", fragmento, re.IGNORECASE):
                pn_list = ["ALL"]
            else:
                pn_list = ["UNKNOWN"]

            # Agregar parte detectada con su(s) P/N
            partes_detectadas.append((texto_parte_limpio_norm.strip(), pn_list))

# ==========================
# Mostrar resultados
# ==========================
print("\nPartes afectadas detectadas:")
for parte, pn_list in partes_detectadas:
    print(f"- PART: {parte} | P/Ns: {', '.join(pn_list)}")

# ==========================
# Comparar los resultados con la BBDD importada
# ==========================
# Modelos, S/N y P/N extraídos previamente
modelo_detectado = modelos_detectados
sn_detectado = seriales_detectados
pn_detectado = [pn for _, pn_list in partes_detectadas for pn in pn_list]

coincidencias_encontradas = []

# Aseguramos que pn_detectado sea una lista
if isinstance(pn_detectado, str):
    pn_detectado = [pn_detectado]

# Filtrar base de datos por coincidencia de modelo
for modelo_detectado in modelos_detectados:
    df_modelo = df[df['modelo'].str.upper() == modelo_detectado.upper()]

    if df_modelo.empty:
        continue

    for sn_detectado in seriales_detectados:
        sn_encontrado = False

        for _, row in df_modelo.iterrows():
            sn_bbdd = str(row['sn']).strip()
            pn_bbdd = str(row['pn']).strip().upper()
            matricula_bbdd = str(row['matricula']).strip()
            coincidencia_sn = False
```

```
        # Verificación del número de serie
            if sn_detectado.upper() == "ALL SERIAL NUMBERS" or sn_bbdd.upper() in ["ALL SERIAL NUMBERS", "ALL"]:
                coincidencia_sn = True
            elif "-" in sn_detectado:
                try:
                    inicio, fin = map(int, sn_detectado.replace("RANGE:", "").replace("TO", "-").replace("to", "-").split("-"))
                    if inicio <= int(sn_bbdd) <= fin:
                        coincidencia_sn = True
                except ValueError:
                    print(f"⚠️ Rango inválido en el número de serie detectado: {sn_detectado}")
            elif sn_detectado == sn_bbdd:
                coincidencia_sn = True

            # Si el número de serie coincide
            if coincidencia_sn:
                sn_encontrado = True

                # Comparar con todos los P/N detectados
                pn_coincide = False
                for pn in pn_detectado:
                    if pn_bbdd.upper() == "ALL" or pn_bbdd.upper() == pn.upper():
                        pn_coincide = True
                        coincidencias_encontradas.append({
                            "Matrícula": matricula_bbdd,
                            "Modelo": modelo_detectado,
                            "S/N directiva": sn_detectado,
                            "S/N BBDD": sn_bbdd,
                            "P/N directiva": pn,
                            "P/N BBDD": pn_bbdd
                        })
                        break  # Ya encontramos coincidencia de P/N

# ==============================
# Mostrar resumen en tabla
# ==============================
if coincidencias_encontradas:
    df_resultado = pd.DataFrame(coincidencias_encontradas)
        print("\n📋 Coincidencia encontrada en el listado de aeronaves gestionadas por la Organización CAMO:")
    print(df_resultado.to_string(index=False))
     print("\n La AD afecta a las aeronaves con las matrículas anteriormente indicadas. Revisar acciones a programar")
else:
    print("\n❌ No se encontró ninguna coincidencia completa entre modelo, número de serie y número de parte en las " \
    "aeronaves gestionadas por la Organización CAMO.")
```