



ugr

Universidad
de Granada

TRABAJO FIN DE GRADO
INGENIERÍA EN INFORMÁTICA

Sistema experto para detectar el Síndrome de Cushing en perros mediante lógica difusa

Autor

Manuel Martínez Cobos

Directores

Javier Medina Quero

Marios Kountouris



ESCUELA TÉCNICA SUPERIOR DE INGENIERÍAS INFORMÁTICA Y DE
TELECOMUNICACIÓN

Granada, Junio de 2026

Sistema experto para detectar el Síndrome de Cushing en perros mediante lógica difusa

Manuel Martínez Cobos

Palabras clave: Síndrome de Cushing, hiperadrenocorticismismo, sistema experto, lógica difusa, apoyo a la decisión clínica

Resumen

El síndrome de Cushing (hiperadrenocorticismismo) es una endocrinopatía frecuente en perros cuya confirmación diagnóstica resulta compleja debido a la variabilidad clínica y a las limitaciones de las pruebas complementarias. En este Trabajo de Fin de Grado (TFG) se desarrolla un sistema basado en conocimiento que estima la probabilidad de que un perro padezca esta enfermedad a partir de signos clínicos y parámetros analíticos.

El sistema implementa un motor de inferencia basado en lógica difusa que evalúa un conjunto de reglas derivadas de una investigación y detenida lectura de la literatura científica sobre dicha enfermedad así como de sus síntomas. La arquitectura modular diseñada permite la integración de nuevas variables y facilita la revisión y validación futura del modelo.

El objetivo final es proporcionar una herramienta de apoyo a la decisión clínica que contribuya a mejorar la detección temprana del síndrome de Cushing en el ámbito veterinario.

Expert system to detect Cushing's Syndrome in dogs using fuzzy logic

Manuel Martínez Cobos

Keywords: Cushing's syndrome, hyperadrenocorticism, expert system, fuzzy logic, clinical decision support

Abstract

Cushing's syndrome (hyperadrenocorticism) is a common endocrine disorder in dogs whose diagnostic confirmation is often challenging due to clinical variability and the limitations of complementary diagnostic tests.

This Bachelor's Thesis presents the development of a knowledge-based system designed to estimate the probability that a dog suffers from this condition based on clinical signs and laboratory parameters. The system implements a fuzzy logic-based inference engine that evaluates a set of rules derived from a thorough review of the scientific literature on the disease and its associated clinical manifestations. The proposed modular architecture enables the integration of additional variables and facilitates future refinement and validation of the model.

The ultimate goal is to provide a clinical decision support tool that contributes to improving the early detection of Cushing's syndrome in veterinary practice.

Yo, **Manuel Martínez Cobos**, alumno de la titulación **INGENIERÍA INFORMÁTICA** de la **Escuela Técnica Superior de Ingenierías Informática y de Telecomunicación de la Universidad de Granada**, con DNI 77449069L, autorizo la ubicación de la siguiente copia de mi Trabajo Fin de Grado en la biblioteca del centro para que pueda ser consultada por las personas que lo deseen.

Fdo: Manuel Martínez Cobos

Granada a 14 de Junio de 2026.

D. **Javier Medina Quero**, Profesor del Departamento de Ingeniería de Computadores, Automática y Robótica de la Universidad de Granada.

D. **Marios Kountouris**, Profesor del Departamento de Ciencias de la Computación e Inteligencia Artificial de la Universidad de Granada.

Informan:

Que el presente trabajo, titulado *Sistema experto para detectar el Síndrome de Cushing en perros mediante lógica difusa*, ha sido realizado bajo su supervisión por **Manuel Martínez Cobos**, y autorizamos la defensa de dicho trabajo ante el tribunal que corresponda.

Y para que conste, expiden y firman el presente informe en Granada a 14 de Junio de 2026.

Los directores:

Javier Medina Quero

Marios Kountouris

Agradecimientos

A mis tutores, por la paciencia y el apoyo que me han brindado a lo largo de estos meses.

A todo aquel que estuvo, porque el simple hecho de estar ya aporta algo.

A mis amigos, por ser ese punto de vista distinto y esa vía de escape cuando hacía falta.

A mi familia, por apostar por mí, por apoyarme en cada pequeño paso, por estar en mis éxitos y en mis fracasos, por inculcarme unos valores y unos principios irremplazables y por enseñarme a ser mi mejor yo en cada momento.

A Coco, mi más fiel compañero. Te echo de menos .

Índice general

1. Introducción	1
1.1. Contexto	1
1.2. Objetivo	2
1.3. Estructura	2
1.3.1. Estructura del trabajo	2
1.3.2. Estructura de la memoria	3
1.4. Alcance y limitaciones	4
2. Estado del arte	7
2.1. Diagnóstico del hiperadrenocorticismo	7
2.2. Limitaciones del enfoque actual	8
2.3. Modelos predictivos en hiperadrenocorticismo	9
2.4. Contribución de este proyecto	10
3. Análisis del sistema	13
3.1. Formalización del problema	13
3.2. Selección y estructuración de variables	15
3.2.1. Variables demográficas	16
3.2.2. Síntomas	16
3.2.3. Variables laboratoriales	17
3.3. Definición del universo de conocimiento	17
3.3.1. Variables lingüísticas	18
3.3.2. Funciones de pertenencia	18
3.3.3. Fuzzificación de las variables de entrada	19
3.3.4. Variable de salida	19
3.3.5. Variables demográficas	20
3.3.6. Variables clínicas	22
3.3.7. Variables laboratoriales	23
3.3.8. Defuzzificación	24
3.4. Diseño del modelo de inferencia difusa	24
3.4.1. Reglas diseñadas y criterios de selección	25
3.4.2. Diseño de la caja negra del sistema	25
3.4.3. Motor de inferencia y flujo de procesamiento	25

3.5. Modularización del conocimiento y evolución hacia una plataforma extensible	26
3.6. Metodología de desarrollo	27
3.6.1. Principios aplicados	27
3.6.2. Fases del proyecto	27
3.7. Planificación temporal	28
3.7.1. Distribución de horas por fase	28
3.8. Presupuesto	29
3.8.1. Costes de personal	29
3.8.2. Costes de hardware	30
3.8.3. Costes de <i>software</i> y servicios externos	30
3.8.4. Resumen del presupuesto total	31
4. Implementación	33
4.1. Arquitectura del sistema	33
4.1.1. Capa de presentación	34
4.1.2. Capa de aplicación	34
4.1.3. Capa de lógica difusa	35
4.1.4. Base de conocimiento	35
4.2. Tecnologías y herramientas	35
4.2.1. Justificación de las elecciones tecnológicas	36
4.3. Desarrollo del motor de inferencia	37
4.3.1. Diagrama de clases del motor de inferencia	37
4.3.2. Módulos de datos de entrada	38
4.3.3. Carga dinámica de la base de conocimiento	39
4.3.4. Construcción del sistema fuzzy	39
4.3.5. Validación de entradas y <i>clipping</i>	40
4.3.6. Pipeline de predicción	40
4.3.7. Métrica de confianza compuesta	40
4.3.8. Servicio de extracción con inteligencia artificial	41
4.4. Desarrollo de la interfaz de usuario	42
4.4.1. Componentes de la biblioteca de UI	42
4.4.2. Página de análisis (<i>/analyze</i>)	42
4.4.3. Página de resultados (<i>/results</i>)	44
4.4.4. Animaciones y experiencia de usuario	45
4.5. Integración y despliegue automático	45
4.5.1. <i>Job 1</i> — Tests del backend	45
4.5.2. <i>Job 2</i> — Build del frontend	46
4.5.3. <i>Job 3</i> — Despliegue en GitHub Pages	46
4.5.4. <i>Job 4</i> — Redespiegue del backend en Render	47
4.5.5. Gestión de secretos	47
4.5.6. Resultado del proceso de despliegue	47

5. Pruebas y validación	49
5.1. Metodología de pruebas	49
5.1.1. Tests unitarios	50
5.1.2. Tests de componente	51
5.1.3. Tests de integración	54
5.1.4. Resumen de la cobertura	57
5.2. Validación clínica	57
5.2.1. Perfil del experto	57
5.2.2. Validación mediante casos clínicos representativos . .	57
5.2.3. Pruebas de usabilidad	59
6. Conclusiones y trabajo futuro	63
6.1. Conclusiones	63
6.1.1. Conclusiones técnico-científicas	63
6.1.2. Conclusiones de diseño de <i>software</i>	64
6.1.3. Conclusiones clínicas	66
6.2. Líneas de mejora	66
6.2.1. Corto plazo — Mejoras inmediatas sobre el sistema actual	66
6.2.2. Medio plazo — Extensión de la plataforma	67
6.2.3. Largo plazo — Evolución hacia un sistema clínico completo	67
6.2.4. Privatización de los datos.	68
A. Sistema de lógica difusa aplicado al proyecto	71
A.1. Arquitectura general del paquete	71
A.2. Variables lingüísticas	72
A.3. Funciones de pertenencia	73
A.4. Reglas difusas	78
A.5. Motor de inferencia (FuzzySystem)	79
A.5.1. Pipeline de inferencia	79
A.5.2. Resultado devuelto	80
A.6. Defuzzificación	80
A.7. Métrica de confianza compuesta	81
A.8. Etiquetado lingüístico del resultado	82
A.9. Resumen aplicado al TFG	83
B. Guía de uso de la plataforma C.O.C.O.	85
B.1. Análisis de un caso clínico	85
B.1.1. Acceso a la plataforma	85
B.1.2. Flujo general de uso	86
B.1.3. Paso 1 — Entrada de datos	86
B.1.4. Paso 2 — Lanzar el análisis	88
B.1.5. Paso 3 — Interpretación de resultados	88

B.2. Implementación de un predictor para nueva patología	90
B.2.1. Paso 1 — Crear la base de conocimiento	90
B.2.2. Paso 2 — Clase de predicción Python	93
B.2.3. Paso 3 — Registrar el endpoint en la API	94
B.2.4. Paso 4 — Registrar la enfermedad en el frontend . . .	95
B.2.5. Resumen del proceso de extensión	96
Glosario	96

Índice de figuras

3.1. Proporción condicional de casos dentro de cada grupo etario en perros con sospecha clínica de HAC. Se observa un incremento marcado a partir de los 7 años y estabilización en edades avanzadas.	21
3.2. Diagrama de Gantt del proyecto C.O.C.O.	28
4.1. Arquitectura en capas de la plataforma C.O.C.O.	33
4.2. Diagrama de clases del <i>framework</i> de lógica difusa	37
4.3. Diagrama de clases de la capa de sistema e integración.	38
4.4. Pipeline CI/CD de GitHub Actions para C.O.C.O.	46
5.1. Distribución de tests por archivo.	57
5.2. Resultados del análisis y predicción para el primer caso de estudio (riesgo moderado).	58
5.3. Resultados del análisis y predicción para el segundo caso de estudio (riesgo alto mediante lectura de PDF).	59
A.1. Dependencias entre los módulos del paquete <code>logicaDifusa</code>	72
A.2. Funciones de pertenencia para la variable lingüística Edad	74
A.3. Funciones de pertenencia para la variable lingüística Peso relativo	75
A.4. Funciones de pertenencia para la variable lingüística ALP	75
A.5. Funciones de pertenencia para la variable lingüística USG	76
A.6. Funciones de pertenencia para la variable lingüística ALT	76
A.7. Funciones de pertenencia para la variable lingüística Colesterol	77
A.8. Funciones de pertenencia comunes a las variables clínicas binarias: polidipsia , poliuria , abdomen pendular , alopecia , debilidad muscular , piel fina , polifagia y jadeo	77
A.9. Funciones de pertenencia para la variable lingüística de salida Riesgo	78
B.1. Flujo de uso de la plataforma C.O.C.O.	86

B.2. Pasos para incorporar un predictor de nueva patología a C.O.C.O.	96
--	----

Índice de cuadros

3.1. Asociación multivariable (OR ajustadas) de variables demográficas con el diagnóstico de HAC. OR derivadas de los coeficientes del modelo final. Datos adaptados de [23]).	16
3.2. Comparación de signos clínicos entre casos y no casos en sospecha de HAC. Se indica retención en el modelo multivariable final y una interpretación clínica contextualizada. Datos adaptados de [23].	16
3.3. Variables laboratoriales evaluadas en sospecha de HAC y su papel dentro del modelo predictivo. Datos adaptados de [23].	17
3.4. Distribución de horas de trabajo por fase del proyecto.	29
3.5. Costes de personal del proyecto.	30
3.6. Costes de hardware amortizados.	30
3.7. Costes de <i>software</i> y servicios externos.	30
3.8. Resumen del presupuesto total del proyecto C.O.C.O.	31
4.1. Endpoints de la API REST de C.O.C.O.	34
4.2. Dependencias principales del backend.	35
4.3. Dependencias principales del frontend.	36
4.4. Componentes de la biblioteca de UI.	43
4.5. Paleta de colores por nivel de riesgo en la página de resultados.	45
4.6. Secretos de repositorio usados en el pipeline CI/CD.	47
5.1. Resumen de la suite de tests automatizados.	49
5.2. Casos de prueba de <code>test_variables.py</code>	50
5.3. Casos de prueba de <code>test_reglas.py</code>	51
5.4. Casos de prueba de <code>test_sistema.py</code>	52
5.5. Perfil del caso de alto riesgo usado en los tests de integración.	54
5.6. Casos de prueba de <code>test_api.py</code>	56
5.7. Resultados de la prueba de usabilidad desglosados por tareas evaluadas.	61
A.1. Funciones de pertenencia implementadas en <code>funcionesPertenencia.py</code>	74
A.2. Métodos de defuzzificación disponibles.	81

B.1. Campos demográficos del formulario.	87
B.2. Signos clínicos disponibles en el formulario.	88
B.3. Campos de laboratorio del formulario.	88
B.4. Niveles de riesgo y umbrales del valor crisp.	89
B.5. Campos obligatorios y opcionales de una regla JSON.	93

Capítulo 1

Introducción

1.1. Contexto

El síndrome de Cushing, o hiperadrenocorticismo, es un término general que engloba los cuadros clínicos derivados de una producción excesiva de hormonas adrenocorticales, principalmente cortisol. En perros, la forma más frecuente es el hiperadrenocorticismo dependiente de hipófisis, originado por una secreción excesiva de ACTH secundaria a una hiperplasia o adenoma hipofisario funcional [12]. Esta enfermedad representa un desafío debido a que sus manifestaciones clínicas son poco específicas y se superponen con múltiples patologías, como la diabetes, distintas enfermedades renales o la piometra. Asimismo, pruebas de laboratorio como el aumento de las transaminasas séricas son también propias de otras enfermedades, es por ello que tampoco sirven como marca clínica que asegure la detección de la enfermedad. Todas estas dificultades obligan a establecer un protocolo clínico y paraclínico estructurado para su confirmación. En muchos casos, los primeros motivos de consulta están relacionados con alteraciones dermatológicas; sin embargo, el hiperadrenocorticismo representa únicamente alrededor del 2 % de los casos dermatológicos en la práctica clínica veterinaria [26].

Además, el síndrome de Cushing puede comprometer gravemente la supervivencia de los animales afectados si no es diagnosticado y tratado de forma precoz. En estudios poblacionales amplios, se ha observado que aproximadamente el 25.9 % de los perros diagnosticados fallecieron como consecuencia directa de la enfermedad. [15].

En este contexto, resulta fundamental optimizar las estrategias de detección temprana para mejorar el pronóstico y la calidad de vida de los pacientes afectados.

1.2. Objetivo

El objetivo principal de este Trabajo Fin de Grado es desarrollar una herramienta aplicable en un entorno real que permita estimar, mediante un sistema basado en reglas clínicas, la probabilidad de que un perro padezca hiperadrenocorticismio.

La herramienta se plantea como un sistema de apoyo a la decisión clínica, destinado a complementar la valoración profesional. De acuerdo con la legislación vigente en materia de sistemas automatizados en el ámbito sanitario, su uso estaría limitado a funciones de asesoramiento, requiriendo siempre la supervisión y validación final por parte de un profesional cualificado, en este caso, un veterinario.

Para alcanzar el objetivo principal, se establecen los siguientes objetivos específicos:

- OE1. Revisión sistemática de la literatura científica y extracción del conocimiento clínico relevante.
- OE2. Definición y formalización de reglas clínicas a partir de combinaciones de sintomatología y parámetros analíticos.
- OE3. Diseño e implementación de un motor de inferencia capaz de evaluar las reglas definidas a partir de distintos formatos de entrada de datos.
- OE4. Desarrollo de una interfaz de usuario que permita al profesional introducir los datos de forma intuitiva y eficiente.
 - OE4.1. Implementación de un lector de archivos PDF/XLSX/CSV con formato estandarizado.
 - OE4.2. Introducción manual de variables mediante teclado.
 - OE4.3. Implementación de formularios de respuesta binaria (sí/no) para variables categóricas.

1.3. Estructura

1.3.1. Estructura del trabajo

El presente trabajo se ha desarrollado siguiendo una metodología estructurada y coherente con los principios de la investigación científica, procurando mantener un enfoque riguroso y alineado con la literatura clínica actual sobre el hiperadrenocorticismio canino.

Desde el punto de vista técnico, el sistema ha sido diseñado bajo una arquitectura modular, con el objetivo de facilitar su mantenimiento, revisión y futura ampliación. Esta organización permite la actualización independiente de cada componente sin comprometer el funcionamiento global de la herramienta.

Los módulos principales desarrollados son los siguientes:

- **Módulo clínico:** Encargado de integrar y formalizar el conocimiento médico extraído de la literatura científica, incluyendo la definición de reglas clínicas basadas en sintomatología y parámetros analíticos.
- **Módulo de inferencia:** Responsable de procesar los datos introducidos por el usuario y evaluar las reglas definidas, generando una estimación de probabilidad de hiperadrenocorticismos.
- **Módulo de interfaz:** Destinado a la interacción con el usuario, permitiendo la introducción de datos mediante distintos formatos (archivos estructurados o entrada manual).

1.3.2. Estructura de la memoria

El documento se organiza en distintos capítulos que describen el marco teórico, la metodología empleada, el desarrollo técnico del sistema, los resultados obtenidos y las conclusiones del trabajo. La estructura del documento es la siguiente:

- **Introducción:** Presenta el proyecto y expone el contexto científico en el que fue desarrollado. Se explican los objetivos perseguidos, la estructura del trabajo y del documento, así como el alcance y las limitaciones de la herramienta desarrollada.
- **Estado del arte:** Recoge una revisión de la literatura científica relevante sobre el diagnóstico del hiperadrenocorticismos canino. Se analizan los métodos de diagnóstico existentes, sus limitaciones, los modelos predictivos previos desarrollados en este ámbito y la contribución diferencial de este proyecto respecto a los trabajos anteriores.
- **Análisis del sistema:** Describe la formalización del problema abordado y el proceso de selección y estructuración de las variables de entrada del sistema, agrupadas en variables demográficas, síntomas clínicos y parámetros laboratoriales. Asimismo, se define el universo de conocimiento empleado y se presenta el diseño del modelo de inferencia difusa, incluyendo las reglas elaboradas y la modularización del conocimiento.

- **Implementación:** Detalla la arquitectura software del sistema, compuesta por la capa de presentación, la capa de aplicación, la capa de lógica difusa y la base de conocimiento. Se describen las tecnologías y herramientas utilizadas, el desarrollo del motor de inferencia y de la interfaz de usuario, así como el proceso de integración continua y despliegue automático.
- **Pruebas y validación:** Expone la metodología de pruebas seguida, que abarca tests unitarios, de componente y de integración, junto con un resumen de la cobertura alcanzada. Se incluye además una validación clínica mediante casos representativos y una prueba de usabilidad.
- **Presupuesto y planificación:** Recoge la metodología de desarrollo empleada, la planificación temporal del proyecto y el presupuesto estimado.
- **Conclusiones:** Resume los resultados obtenidos, evalúa el grado de cumplimiento de los objetivos planteados y propone posibles líneas de trabajo futuro.

1.4. Alcance y limitaciones

La herramienta desarrollada en este trabajo tiene como finalidad servir como sistema de apoyo a la decisión clínica en la sospecha de hiperadrenocorticismos canino. Su función principal es estimar la probabilidad de presencia de la enfermedad a partir de reglas clínicas definidas con base en la literatura científica actual.

El sistema permite la introducción de datos mediante distintos formatos y ha sido diseñado bajo una arquitectura modular que facilita su mantenimiento y posible ampliación futura.

No obstante, esta herramienta no constituye un método diagnóstico definitivo ni sustituye las pruebas confirmatorias recomendadas en la práctica clínica. Asimismo, no reemplaza el criterio clínico del profesional veterinario.

El modelo implementado se basa exclusivamente en reglas deterministas y no incorpora técnicas de aprendizaje automático. Asimismo, no ha sido validado mediante estudios clínicos prospectivos en población real, por lo que su utilidad debe entenderse como complementaria y orientativa.

Su rendimiento depende directamente de la calidad y completitud de los datos introducidos por el usuario. La ausencia de un conjunto de datos clínicos estructurado —que incluyera resultados analíticos y observaciones profesionales correspondientes a perros diagnosticados, sanos

y casos sospechosos— ha limitado la posibilidad de desarrollar y entrenar un modelo basado en aprendizaje automático.

En caso de disponer de dicho conjunto de datos, podría haberse implementado un módulo adicional que funcionara mediante un entrenamiento sistemático y desarrollo de aprendizaje automático destinado a comparar sus predicciones con las del sistema basado en reglas. Este módulo habría permitido ajustar o matizar la estimación probabilística en situaciones atípicas, así como identificar posibles casos fuera de los patrones clínicos habituales.

Capítulo 2

Estado del arte

2.1. Diagnóstico del hiperadrenocorticismo

El diagnóstico del hiperadrenocorticismo canino, de ahora en adelante abreviado como HAC, constituye en la actualidad un proceso clínico complejo que combina la evaluación de los síntomas compatibles y la interpretación de pruebas complementarias de laboratorio e imagen. No existe una prueba única que permita establecer el diagnóstico de forma absolutamente concluyente, por lo que la confirmación requiere la integración de datos clínicos, analíticos y, en determinados casos, pruebas de imagen. [12]

Para diagnosticar un caso de Síndrome de Cushing en un perro, lo más habitual es realizar la interpretación de análisis de orina y sangre, junto con las observaciones clínicas realizadas por el veterinario. En la fase inicial de sospecha, suelen detectarse alteraciones inespecíficas como elevación de la fosfatasa alcalina (ALP), incremento de enzimas hepáticas, hipercolesterolemia o cambios en la densidad urinaria. No obstante, estos hallazgos no son definitivos y pueden estar presentes en múltiples enfermedades sistémicas. [12]

Las pruebas dinámicas hormonales constituyen el pilar fundamental del diagnóstico confirmatorio. Entre ellas, el test de supresión con dexametasona a baja dosis (LDDST) es considerado la prueba de cribado de elección debido a su elevada sensibilidad. Por su parte, la prueba de estimulación con ACTH presenta una mayor especificidad, aunque con menor sensibilidad diagnóstica. Ambas pruebas pueden generar resultados falsos positivos con relativa facilidad, lo que obliga a interpretar los resultados dentro del contexto clínico global del paciente. [3]

Además, las técnicas de imagen, especialmente la ecografía abdominal, permiten evaluar el tamaño y la situación de las glándulas suprarrenales,

facilitando la diferenciación entre hiperadrenocorticismo dependiente de la hipófisis y de origen adrenal. En centros especializados, pueden emplearse técnicas avanzadas como la resonancia magnética para la valoración de adenomas hipofisarios.[3]

En consecuencia, el diagnóstico del síndrome de Cushing en perros se basa en un proceso interpretativo que requiere experiencia clínica y correlación multidisciplinar, lo que introduce cierto grado de variabilidad en la toma de decisiones.

2.2. Limitaciones del enfoque actual

A pesar de tratarse de una enfermedad ampliamente estudiada, el diagnóstico del hiperadrenocorticismo canino continúa presentando importantes limitaciones. La ausencia de un **gold standard** diagnóstico que permita confirmar de forma definitiva la presencia de la enfermedad implica una elevada dependencia del criterio clínico del veterinario, quien debe integrar e interpretar resultados procedentes de pruebas que, de forma aislada, carecen de especificidad absoluta.

Asimismo, el proceso diagnóstico puede implicar la realización de múltiples pruebas dinámicas hormonales, lo que incrementa tanto el coste económico como el tiempo necesario para alcanzar una confirmación del diagnóstico de esta enfermedad. Esta situación puede generar incertidumbre clínica, especialmente en casos con resultados limítrofes o presentaciones atípicas.

La similitud sintomática con otras enfermedades endocrinas, particularmente la diabetes, constituye otra limitación relevante. Diversos estudios han evidenciado una alta tasa de coexistencia entre ambas patologías. En un estudio realizado en la consulta de endocrinología del Hospital Veterinario de la Universidad de Zaragoza entre 2009 y 2015, se observó que un 27.47% de los casos correspondía a diabetes mellitus, un 28.57% al síndrome de Cushing y un 12.08% presentaba ambas patologías simultáneamente. De ese 12.08%. Además, un porcentaje significativo de perros diagnosticados con hiperadrenocorticismo desarrolló posteriormente diabetes mellitus, lo que sugiere una relación clínica estrecha y refuerza la importancia de una detección temprana y precisa [2]. Un correcto diagnóstico temprano de esos casos de HAC podría haber evitado el desarrollo de diabetes en los pacientes y sus posteriores complicaciones.

La ausencia de esa prueba definitiva también supone una situación estresante y económicamente comprometida para los dueños del perro sospechoso de portar la enfermedad. Según E.N. Behrend en su artículo para el JVIM de 2012 (artículo de bastante peso en la comunidad); el

camino a seguir pasa por un análisis de orina 2 días después de conseguir la muestra a analizar. En caso de sospecha, realizar los ya mencionados test de estimulación de ACTH y test LDDST. En caso de que de nuevo el veterinario encargado dictamine que el perro es sospechoso de sufrir de Síndrome de Cushing, se debería hacer un retest en 3-6 meses para tener la mayor garantía posible, que aún está alejada del 100 %.[3] Vemos entonces como se puede retrasar el diagnóstico de la enfermedad durante meses.

En conjunto, estas limitaciones evidencian que el diagnóstico del hiperadrenocorticismismo no depende exclusivamente de una prueba concreta, sino de un proceso integrador complejo que puede generar variabilidad en la toma de decisiones clínicas.

2.3. Modelos predictivos en hiperadrenocorticismismo

Ante las limitaciones del enfoque diagnóstico tradicional, en los últimos años han comenzado a explorarse métodos basados en modelos estadísticos y técnicas de aprendizaje automático como herramientas de apoyo a la decisión clínica en el ámbito del hiperadrenocorticismismo canino. Estos enfoques no pretenden sustituir las pruebas endocrinas convencionales, sino complementar la interpretación clínica mediante el análisis simultáneo de múltiples variables demográficas, analíticas y sintomáticas, con el objetivo de identificar patrones predictivos que puedan mejorar la precisión y rapidez del diagnóstico.

Para nuestro caso, destaca en especial el artículo "Machine-learning based prediction of Cushing's syndrome in dogs attending UK primary-care veterinary practice" de Imogen Schofield [22]. En este proyecto, con un presupuesto mucho mayor al estimado para nuestro sistema experto, se utilizan los datos de 905,544 perros bajo atención veterinaria en Reino Unido comparando perros finalmente diagnosticados de HAC con perros en los que la enfermedad fue descartada. Esos datos se evalúan con 4 algoritmos:

- LASSO (Least Absolute Shrinkage and Selection Operator): Modelo de regresión penalizada que realiza selección automática de variables, reduciendo el sobreajuste y facilitando la interpretabilidad del resultado.
- Random Forest: Método de aprendizaje basado en conjuntos de árboles de decisión que permite modelar relaciones no lineales y complejas entre variables clínicas.
- SVM lineal (Support Vector Machine): Algoritmo que busca el hiperplano óptimo que separa ambas clases, maximizando el margen de clasificación.

- SVM no lineal (kernel RBF): Extensión del SVM que emplea una función kernel radial para capturar relaciones no lineales entre las variables predictoras.

El modelo con mejor rendimiento fue LASSO, alcanzando un AUROC de 0.85, con una sensibilidad del 71 % y una especificidad del 82 %, resultados comparables al rendimiento de algunas pruebas endocrinas utilizadas en clínicas veterinarias.

Este trabajo demuestra que el HAC ofrece algunos patrones diagnósticos que, pese a poder ser confundidos con los correspondientes a otras enfermedades, como la ya mencionada diabetes, pueden ser estudiados y analizados mediante modelos estadísticos para facilitar la labor del veterinario, reforzando y apoyando sus diagnósticos.

El buen desempeño de los modelos de Aprendizaje Automático, en este caso utilizando el algoritmo LASSO; también nos hace pensar que, de haber contado con un dataset apropiado, la inclusión de un módulo de validación mediante aprendizaje automático a nuestro sistema experto habría permitido incorporar un mecanismo adicional de validación del resultado del sistema.

2.4. Contribución de este proyecto

El análisis del estado del arte ha evidenciado las dificultades existentes en el diagnóstico del Síndrome de Cushing en perros. La variabilidad e inespecificidad de los síntomas, junto con la superposición clínica con otras enfermedades y las limitaciones de las pruebas endocrinas actuales, generan un contexto de incertidumbre diagnóstica que puede conducir tanto a errores como a retrasos en la confirmación de la enfermedad. Estos retrasos no solo afectan al tratamiento temprano, sino que pueden favorecer la aparición de complicaciones secundarias que agraven el cuadro clínico y comprometan la supervivencia del animal.

En este contexto, las principales contribuciones de este Trabajo de Fin de Grado son las siguientes:

- **Formalización de un sistema experto orientado al apoyo diagnóstico:** Se propone una herramienta basada en reglas que integra síntomas clínicos y resultados analíticos para estimar la probabilidad de hiperadrenocorticismos, proporcionando al veterinario un apoyo estructurado en la toma de decisiones.
- **Reducción de la incertidumbre clínica inicial:** El sistema permite ofrecer una estimación razonada de la probabilidad de HAC en fases

tempranas del proceso diagnóstico, ayudando a priorizar pruebas complementarias o a orientar la estrategia terapéutica inicial.

- **Integración conceptual con modelos predictivos existentes:** Aunque desarrollado sin un módulo completo de aprendizaje automático por limitaciones de datos, el diseño del sistema contempla la posibilidad de incorporar mecanismos de validación y ajuste basados en datos reales, alineándose con las tendencias actuales en modelos predictivos aplicados a la medicina veterinaria.
- **Base para desarrollos futuros:** La arquitectura propuesta sienta las bases para futuras ampliaciones, como la integración de datasets clínicos reales, la incorporación de algoritmos de aprendizaje automático para calibración automática o la adaptación a otras enfermedades endocrinas con una problemática diagnóstica similar.

Estas contribuciones permiten situar el presente trabajo dentro del ámbito de los sistemas de apoyo a la decisión clínica en medicina veterinaria, ofreciendo un enfoque estructurado que combina conocimiento experto con fundamentos formales de modelado; abriendo la puerta a futuras mejoras basadas en datos empíricos.

Capítulo 3

Análisis del sistema

3.1. Formalización del problema

El problema abordado en este trabajo puede formalizarse como un problema de estimación bajo incertidumbre. Dado un conjunto de variables clínicas y analíticas observadas en un paciente canino, se desea obtener una estimación del grado de sospecha de sufrir hiperadrenocorticismio (HAC).

En la práctica clínica real, el veterinario no dispone de una única variable decisiva que confirme o descarte la enfermedad. En su lugar, debe integrar múltiples evidencias de distinta naturaleza: datos demográficos, signos clínicos observables, parámetros bioquímicos y resultados urinarios. Cada uno de estos elementos aporta información parcial, pero ninguno resulta concluyente de forma aislada. El proceso diagnóstico es, por tanto, acumulativo, interpretativo y progresivo.

Desde un punto de vista formal, el sistema puede describirse como una función:

$$f : X \rightarrow [0, 1]$$

Donde X representa el espacio de entrada compuesto por variables demográficas, clínicas y de laboratorio, y el valor de salida corresponde a un grado continuo de pertenencia al conjunto “sospecha de HAC”. El intervalo $[0, 1]$ permite expresar la estimación como una probabilidad normalizada, que posteriormente puede transformarse en un porcentaje para facilitar su interpretación.

A diferencia de un clasificador binario tradicional, el objetivo no es emitir un diagnóstico categórico (HAC/no HAC), sino proporcionar una estimación gradual que refleje la naturaleza progresiva e incierta del proceso diagnóstico real. Este planteamiento permite modelar situaciones intermedias, casos

límites y combinaciones parcialmente compatibles con la enfermedad, que en la práctica clínica constituyen una parte relevante de los casos sospechosos.

El conocimiento disponible sobre el HAC no se expresa en forma de relaciones deterministas exactas, sino mediante asociaciones clínicas y patrones frecuentes. Por ejemplo, determinados signos aumentan significativamente la sospecha, mientras que otros la reducen o simplemente aportan contexto. Además, pueden existir combinaciones de variables que refuercen mutuamente su relevancia diagnóstica o, por el contrario, generen incoherencias que deban ser ponderadas.

Este tipo de conocimiento no es fácilmente representable mediante lógica clásica, en la que las condiciones se evalúan como verdaderas o falsas de forma estricta. En el contexto clínico, conceptos como “ALP elevada”, “edad avanzada” o “distensión abdominal marcada” no poseen límites rígidos y definidos universalmente, sino que presentan transiciones graduales y dependen del contexto individual del paciente.

Por tanto, el modelo requiere una representación capaz de integrar información imprecisa, incompleta o parcialmente contradictoria, permitiendo además expresar los conceptos médicos en términos lingüísticos cercanos a la práctica veterinaria habitual.

En consecuencia, el problema se aborda mediante un sistema de inferencia basado en lógica difusa. Este enfoque permite:

- Representar variables continuas mediante etiquetas lingüísticas (por ejemplo, normal, moderadamente elevada y muy elevada).
- Modelar transiciones graduales entre estados clínicos.
- Combinar múltiples evidencias de forma acumulativa.
- Mantener la interpretabilidad del razonamiento, evitando que el sistema actúe como una “caja negra”.
- Obtener una salida continua que refleje el grado global de sospecha.

De esta manera, el sistema reproduce de forma estructurada el razonamiento clínico humano, formalizando el proceso de integración de evidencias sin sustituir el juicio profesional del veterinario, sino apoyándolo mediante una herramienta coherente, explicativa y matemáticamente definida.

3.2. Selección y estructuración de variables

A partir del análisis de la literatura científica reciente sobre modelos predictivos y estudios del hiperadrenocorticismo canino, se identificaron aquellas variables que presentan mayor relevancia estadística y coherencia fisiopatológica en la sospecha diagnóstica de la enfermedad.

La selección no se realizó únicamente en función de la frecuencia clínica, sino que también se consideró su significación estadística, su independencia respecto a otras variables y su papel dentro de modelos multivariados previamente publicados.

En el estudio de referencia, los resultados del modelo de regresión logística multivariable se reportan en forma de coeficientes β , que representan el efecto de cada variable sobre el logaritmo de las odds del diagnóstico. Con el objetivo de facilitar la interpretación clínica y comparativa de los efectos, dichos coeficientes fueron transformados a odds ratio (OR) mediante la relación $OR = e^{\beta}$. Esta transformación no modifica la estructura ni el ajuste del modelo original, sino que expresa los mismos efectos en una escala directamente interpretable como razón de probabilidades. Del mismo modo, los intervalos de confianza fueron exponenciados para mantener coherencia en la presentación de los resultados.

Con el objetivo de estructurar adecuadamente el conocimiento, las variables se organizaron en tres categorías funcionales.

3.2.1. Variables demográficas

Incluyen factores estructurales asociados al riesgo poblacional de la enfermedad:

Variable	Categoría (vs. referencia)	OR ajustada	IC 95 %	p
Sexo-estado reprod.	Hembra esterilizada (vs hembra intacta)	0.53	0.31-0.89	0.02
Sexo-estado reprod.	Macho intacto (vs hembra intacta)	0.71	0.38-1.34	0.29
Sexo-estado reprod.	Macho esterilizado (vs hembra intacta)	0.55	0.32-0.93	0.03
Edad (años)	7 a <11 (vs <7)	1.90	1.14-3.16	0.01
Edad (años)	≥11 (vs <7)	1.79	1.06-2.97	0.03
Raza	Bichon frise (vs mestizo)	1.97	1.01-3.86	0.05
Raza	Border terrier (vs mestizo)	1.84	0.77-4.39	0.17
Raza	Jack Russell terrier (vs mestizo)	1.12	0.63-1.99	0.72
Raza	Labrador retriever (vs mestizo)	0.25	0.10-0.66	0.005
Raza	West Highland white terrier (vs mestizo)	0.31	0.15-0.64	0.001
Raza	Schnauzer (vs mestizo)	0.36	0.13-1.01	0.05
Raza	Yorkshire terrier (vs mestizo)	1.09	0.50-2.41	0.82
Raza	Staffordshire bull terrier (vs mestizo)	1.05	0.53-2.08	0.47
Raza	Otros pura raza (vs mestizo)	0.96	0.64-1.43	0.84

Cuadro 3.1: Asociación multivariable (OR ajustadas) de variables demográficas con el diagnóstico de HAC. OR derivadas de los coeficientes del modelo final. Datos adaptados de [23]).

3.2.2. Síntomas

Corresponden a signos observables durante la exploración o reportados por el propietario:

Signo clínico	Casos (%)	No casos (%)	p	Modelo final	Interpretación clínica
Polidipsia	70.1	48.2	¡0.001	Sí	Signo muy frecuente y fuerte predictor positivo
Poliuria	58.8	36.0	¡0.001	No	Asociada, pero colineal con polidipsia
Polifagia	24.6	14.2	¡0.001	No	Asociación moderada; valor complementario
Vómitos	4.8	10.9	0.001	Sí	Su presencia reduce probabilidad de HAC
Diarrea	6.5	10.5	0.03	No	Asociación débil; probable ruido clínico
Distensión abd./Hepatomegalia	49.5	21.4	¡0.001	Sí	Mayor peso predictivo individual
Piel fina/seca	24.1	18.5	0.04	No	Asociación leve; bajo poder discriminativo
Alopecia	29.7	15.0	¡0.001	Sí	Signo endocrino típico; buen predictor
Prurito	3.8	8.3	0.005	Sí	Reduce probabilidad (menos típico en HAC)
Atrofia muscular	13.6	8.3	0.01	No	Asociación moderada; no independiente
Letargo	18.3	20.7	0.37	No	No discriminativo
Jadeo	20.1	18.3	0.49	No	No asociado
Signos neurológicos	4.5	5.7	0.41	No	No relevante en diagnóstico inicial

Cuadro 3.2: Comparación de signos clínicos entre casos y no casos en sospecha de HAC. Se indica retención en el modelo multivariable final y una interpretación clínica contextualizada. Datos adaptados de [23].

Nota: La significación mostrada corresponde a la comparación bivalente (casos vs no casos). La retención en el modelo final indica asociación independiente tras ajustar por el resto de predictores; algunas variables

clínicamente frecuentes (p.ej., poliuria) no fueron retenidas por colinealidad con otras variables, como por ejemplo la polidipsia.

3.2.3. Variables laborales

Incluyen parámetros bioquímicos y urinarios que aportan evidencia fisiopatológica:

Variable	Modelo final	Impacto sobre la probabilidad de HAC	p	Interpretación clínica y rol en el sistema
Fosfatasa Alcalina (ALP)	Sí	Impacto fuerte	≤ 0.001	Variable central. Elevación marcada es altamente sugestiva de HAC. Factor clave del módulo laboratorio.
Densidad urinaria (USG)	Sí	Impacto moderado	0.001	Refuerza o contradice la clínica de poliuria/polidipsia. Coherencia veterinaria-laboratorio.
ALT / AST	No	Impacto contextual	≤ 0.001	Elevación frecuente pero inespecífica. Variable contextual, no decisiva.
Glucosa	No	Impacto complementario	–	Alteración metabólica secundaria al cortisol. Aumenta sospecha si coincide con polifagia/obesidad.
Colesterol	No	Impacto complementario metabólico	–	Alteración metabólica consistente en HAC. Variable complementaria.
Proteinuria	No	Impacto bajo	0.08	Valor bajo como predictor independiente.

Cuadro 3.3: Variables laborales evaluadas en sospecha de HAC y su papel dentro del modelo predictivo. Datos adaptados de [23].

3.3. Definición del universo de conocimiento

El sistema desarrollado se basa en un modelo de inferencia difusa de tipo Mamdani, donde las variables clínicas, demográficas y laborales se representan mediante variables lingüísticas. Este enfoque permite modelar la incertidumbre inherente al razonamiento clínico veterinario y transformar observaciones numéricas o categóricas en conceptos interpretables por el sistema experto.

El universo de conocimiento del sistema está constituido por el conjunto de variables seleccionadas en el apartado anterior, formalizadas como variables lingüísticas dentro del marco de la lógica difusa. Cada variable queda definida por un dominio de valores, un conjunto de etiquetas lingüísticas y sus correspondientes funciones de pertenencia.

El objetivo de esta formalización no es reproducir de forma exacta los coeficientes estadísticos de los modelos de regresión logística descritos en la literatura, sino traducir su significado clínico en estructuras interpretables que permitan modelar el razonamiento veterinario bajo incertidumbre.

3.3.1. Variables lingüísticas

Siguiendo la definición propuesta por Zadeh, una variable lingüística puede definirse como el cuádruplo:

$$V = (X, T, U, M)$$

donde:

- X representa el nombre de la variable.
- T es el conjunto de términos lingüísticos.
- U es el universo de discurso.
- M es el conjunto de funciones de pertenencia asociadas a cada término.

Por ejemplo, para la variable *Edad*:

$$Edad = (Edad, \{Joven, Adulto, Senior\}, [0, 18], M)$$

donde el universo de discurso corresponde al intervalo de edades observables en perros y cada término lingüístico se representa mediante una función de pertenencia difusa.

3.3.2. Funciones de pertenencia

Cada término lingüístico se modela mediante una función de pertenencia:

$$\mu_A : U \rightarrow [0, 1]$$

que asigna a cada valor x un grado de pertenencia al conjunto difuso A . Un valor cercano a 1 indica alta compatibilidad con el concepto lingüístico, mientras que un valor próximo a 0 indica incompatibilidad.

Por ejemplo, para un perro de 9 años, el sistema podría obtener:

$$\mu_{Joven}(9) = 0.1 \quad \mu_{Adulto}(9) = 0.8 \quad \mu_{Senior}(9) = 0.3$$

Obsérvese que un mismo valor puede pertenecer simultáneamente a varios conjuntos difusos con distintos grados de pertenencia, reproduciendo la naturaleza gradual del razonamiento clínico.

3.3.3. Fuzzificación de las variables de entrada

La etapa de fuzzificación transforma las observaciones clínicas introducidas por el usuario en grados de pertenencia difusos. Sea:

$$x_i \in U_i$$

una observación correspondiente a una variable de entrada. La fuzzificación consiste en calcular:

$$\mu_{A_j}(x_i)$$

para cada término lingüístico A_j asociado a dicha variable.

Por ejemplo, si se introduce una fosfatasa alcalina (ALP) equivalente a 4 veces el límite superior de referencia ($ALP = 4.0$), el sistema podría obtener:

$$\mu_{Normal}(4.0) = 0 \quad \mu_{Moderada}(4.0) = 0.35 \quad \mu_{Marcada}(4.0) = 0.85$$

lo que refleja que el valor es altamente compatible con una elevación marcada de ALP, con compatibilidad parcial con la categoría moderada y nula con el rango normal.

3.3.4. Variable de salida

La variable de salida del sistema se define como el grado de sospecha de hiperadrenocorticismo:

$$Sospecha_HAC \in [0, 1]$$

Este intervalo representa un grado continuo de pertenencia al conjunto “sospecha de HAC”. Para facilitar la interpretación clínica, se definen tres etiquetas lingüísticas:

- **Baja**
- **Media**
- **Alta**

Cada regla activa parcial o totalmente uno o varios de estos conjuntos de salida. Tras la agregación de todas las reglas, se obtiene un conjunto

difuso global de sospecha diagnóstica, que posteriormente se transforma en un valor numérico único mediante defuzzificación.

Dicho valor continuo podrá transformarse en un porcentaje y agruparse en tres niveles interpretativos:

- Bajo riesgo: $[0, 0.25)$
- Riesgo intermedio: $[0.25, 0.75)$
- Alto riesgo: $[0.75, 1]$

3.3.5. Variables demográficas

Edad

La edad se modela como una variable continua:

$$Edad \in [0, 18]$$

Dado que el riesgo de HAC no presenta un comportamiento lineal y aumenta de forma marcada a partir de los 7–9 años, se definen tres conjuntos difusos solapados:

- **Joven**
- **Adulto**
- **Senior**

Las funciones de pertenencia presentan transiciones suaves para evitar discontinuidades artificiales en la evaluación del riesgo.

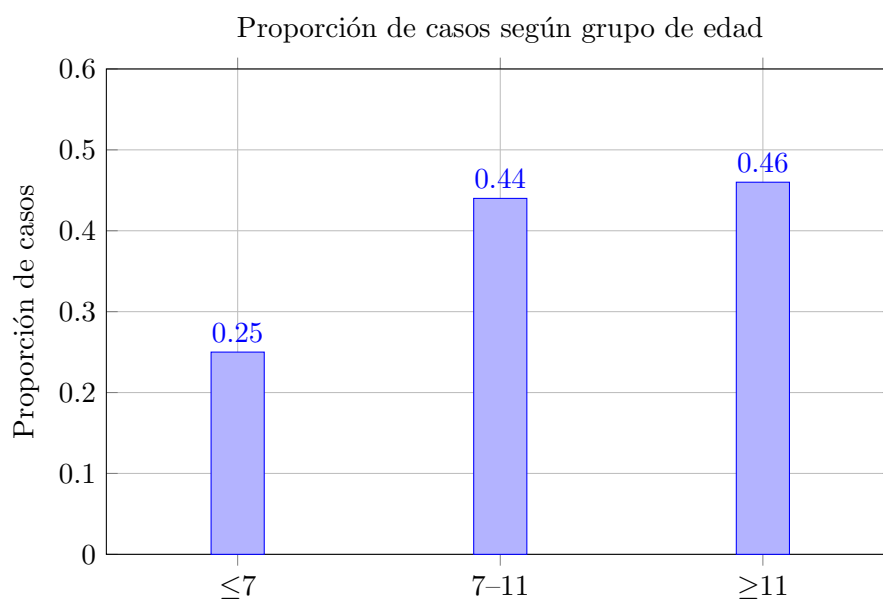


Figura 3.1: Proporción condicional de casos dentro de cada grupo etario en perros con sospecha clínica de HAC. Se observa un incremento marcado a partir de los 7 años y estabilización en edades avanzadas.

Raza

La raza, originalmente categórica, se transforma en una variable lingüística basada en su asociación epidemiológica con la enfermedad. Se definen tres categorías basadas en su OR en comparación con el perro de raza mestiza:

- **Protectora** ($OR \leq 1$)
- **Neutra** ($OR = 1$)
- **Con tendencia** ($OR \geq 1$)

Esta clasificación se fundamenta en la dirección y magnitud de las odds ratio ajustadas descritas en modelos multivariantes.

Peso relativo a la raza-sexo

Para evitar la dependencia directa del peso respecto a la raza y el sexo, se emplea una variable comparativa que evalúa la desviación respecto a la media adulta de su grupo de referencia. Se definen tres categorías:

- **Bajo** (por debajo de $\pm 15\%$ de la media)
- **Estándar** (dentro de $\pm 15\%$)
- **Alto** (por encima de $\pm 15\%$)

La categoría “Alto” se asocia a un incremento moderado del riesgo.

Seguimiento médico

El estado de aseguramiento se incorpora como variable contextual:

- **Asegurado**
- **No asegurado**

Esta variable no modifica la probabilidad fisiopatológica de desarrollar la enfermedad, sino el contexto de detección diagnóstica y el nivel de confianza interpretativa.

3.3.6. Variables clínicas

Los signos clínicos se modelan como variables binarias con posibilidad de estado desconocido:

- **Ausente**
- **Presente**
- **No evaluado**

Entre los signos incluidos se encuentran:

- Polidipsia
- Poliuria
- Polifagia
- Vómitos
- Diarrea
- Distensión abdominal / hepatomegalia
- Alopecia endocrina

- Prurito
- Letargo

Cada uno de ellos será posteriormente ponderado dentro del sistema de reglas según su relevancia clínica y su fuerza de asociación descrita en la literatura.

3.3.7. Variables laboratoriales

Fosfatasa alcalina (ALP)

Se define como la variable central del módulo de laboratorio. Su dominio corresponde a valores normalizados respecto al rango de referencia del laboratorio. Se establecen tres etiquetas lingüísticas:

- Normal
- Moderadamente elevada
- Marcadamente elevada

La elevación marcada constituye uno de los elementos con mayor peso inferencial dentro del sistema.

Densidad urinaria (USG)

Se modela como variable categórica:

- Diluida (≤ 1.020)
- No diluida (> 1.020)

Esta variable actúa como elemento de coherencia entre la clínica de poliuria/polidipsia y la evidencia analítica.

Transaminasas (ALT/AST)

Se incluyen como variables complementarias con dos etiquetas:

- Normal
- Elevada

Su papel es contextual y no decisivo, dado su carácter inespecífico.

Glucosa y colesterol

Ambas se modelan como variables metabólicas auxiliares:

- Normal
- Elevada

Incrementan la coherencia cuando coinciden con las variables clínicas compatibles y ayudan a hacer el sistema más explicativo, pero no constituyen variables nucleares del sistema.

3.3.8. Defuzzificación

Para generar un resultado interpretable por el veterinario es necesario transformar el conjunto difuso de salida en un valor numérico único. El sistema emplea el método del centroide (*Center of Gravity*):

$$y^* = \frac{\int y \mu(y) dy}{\int \mu(y) dy}$$

donde $\mu(y)$ representa el conjunto difuso agregado e y^* es el valor *crisp* final devuelto por el sistema. Este valor se transforma posteriormente a porcentaje:

$$Probabilidad = 100 \cdot y^*$$

y se clasifica según los niveles de riesgo definidos en la variable de salida, facilitando así la interpretación clínica del resultado.

En conjunto, la definición del universo de conocimiento establece una estructura formal capaz de representar información heterogénea —demográfica, clínica y laboratorial— bajo un marco común de incertidumbre gradual. Esta formalización permite que el sistema de inferencia difusa integre las distintas evidencias de manera acumulativa y coherente con el razonamiento clínico veterinario.

3.4. Diseño del modelo de inferencia difusa

En este apartado se describe la arquitectura funcional del sistema difuso propuesto, con énfasis en tres componentes clave: la base de reglas,

el comportamiento global como caja negra y el motor de inferencia que transforma evidencias clínicas en un grado continuo de sospecha diagnóstica.

3.4.1. Reglas diseñadas y criterios de selección

En este subapartado se detallan las reglas difusas finalmente incorporadas al sistema, así como los criterios empleados para su diseño y depuración. Se priorizan reglas clínicamente interpretables, consistentes con la evidencia bibliográfica y con capacidad de discriminar entre combinaciones de variables frecuentes en la práctica veterinaria.

3.4.2. Diseño de la caja negra del sistema

En este subapartado se presenta el sistema como una caja negra de entrada-salida, especificando con precisión qué información recibe, cómo se transforma internamente y qué tipo de salida produce. Esta visión permite analizar el comportamiento global del modelo sin depender inicialmente del detalle interno de cada regla.

Se describen las entradas demográficas, clínicas y laboratoriales, el proceso de agregación de evidencias y la salida final en términos de sospecha normalizada de HAC. También se discute cómo esta representación facilita la validación funcional, la trazabilidad y la integración del sistema en un flujo de apoyo a la decisión clínica.

3.4.3. Motor de inferencia y flujo de procesamiento

Este subapartado desarrolla el funcionamiento interno del motor de inferencia difusa, incluyendo las etapas de fuzzificación, evaluación de reglas, implicación, agregación y defuzzificación. Se explican las decisiones de diseño adoptadas en cada fase y su impacto sobre la estabilidad e interpretabilidad de la salida.

Igualmente, se describe el flujo completo desde la lectura de variables de entrada hasta la obtención del valor final de sospecha, destacando cómo el modelo mantiene coherencia clínica en presencia de información imprecisa, incompleta o parcialmente contradictoria.

3.5. Modularización del conocimiento y evolución hacia una plataforma extensible

A lo largo del proceso de análisis descrito en este capítulo se fue haciendo evidente que la naturaleza del problema abordado invitaba a una reflexión más amplia sobre el alcance del sistema. El síndrome de Cushing es una de las endocrinopatías más frecuentes en perros, pero no es la única. Enfermedades como el hipotiroidismo, la diabetes mellitus canina o la enfermedad de Addison comparten con el HAC una característica fundamental desde el punto de vista del modelado: su diagnóstico se apoya en la combinación de variables demográficas, signos clínicos y parámetros analíticos, y su confirmación se ve dificultada por la gradualidad e imprecisión de esas evidencias. En otras palabras, la estructura del problema es la misma; solo cambian las variables, sus rangos y las reglas que las relacionan.

Esta observación, unida al hecho de que el diseño adoptado ya separaba de forma natural el conocimiento clínico —codificado en ficheros JSON independientes— del motor de inferencia que lo procesa, llevó a plantear una decisión de diseño con un alcance superior al inicialmente previsto: en lugar de construir una herramienta exclusiva para la detección del HAC, se optó por diseñar el sistema como una plataforma abierta y extensible, a la que cualquier miembro de la comunidad pudiera incorporar nuevos predictores de enfermedades siguiendo la misma estructura. Esta plataforma recibe el nombre de C.O.C.O. (Cuidado y Observación Canina Optimizada), y el predictor de hiperadrenocorticismismo desarrollado en este trabajo constituye su primera implementación.

Es importante subrayar que esta decisión no supone una ampliación del alcance del TFG ni una desviación respecto a sus objetivos originales. El trabajo sigue centrado íntegramente en el síndrome de Cushing: la revisión bibliográfica, la selección de variables, la definición del universo de conocimiento y el diseño de las reglas de inferencia son exclusivamente para esa patología. Lo que cambia es la forma en que se empaqueta e implementa el sistema resultante: en lugar de hacerlo como una aplicación de propósito único, se hace de manera que la arquitectura sea reutilizable desde el primer momento, sin coste adicional de desarrollo y con una ganancia significativa en términos de mantenibilidad, extensibilidad y valor para la comunidad. Los detalles concretos de cómo se materializa esta arquitectura se describen en el Capítulo 4.

3.6. Metodología de desarrollo

El carácter individual del TFG y la necesidad de iterar frecuentemente sobre la base de conocimiento clínica determinaron la elección de una metodología **Kanban personal** [1]. Frente a marcos ágiles más formales como Scrum [24], Kanban no requiere cadencias fijas de *sprint* ni reuniones de equipo, lo que resulta adecuado para un proyecto unipersonal con una carga de trabajo variable.

3.6.1. Principios aplicados

- **Visualización del flujo de trabajo:** Se dividieron las tareas en tres grupos: Planificadas, en proceso y finalizadas. Ha sido obligatorio que todas las tareas realizadas pasen por todas las fases del flujo.
- **Limitación del trabajo en progreso (WIP):** Como máximo dos tareas activas simultáneamente, evitando el cambio excesivo entre el backend difuso y el frontend.
- **Integración continua desde el inicio:** El pipeline de CI/CD (sección 4.5) se configuró en la primera semana del proyecto, garantizando que cualquier cambio en la rama principal pasase los tests automáticos antes de llegar a producción.
- **Revisión iterativa de la base de conocimiento:** Los archivos JSON de variables y reglas se refinaron en varias iteraciones tras contrastar los resultados del motor de inferencia con la literatura clínica de referencia.

3.6.2. Fases del proyecto

El desarrollo se articuló en seis fases secuenciales, aunque con solapamiento parcial:

1. **Análisis y diseño:** Revisión bibliográfica sobre HAC canino y lógica difusa; definición de la arquitectura en capas; elección de tecnologías.
2. **Motor de inferencia:** Diseño de múltiples sistemas con sus correspondientes motores de inferencia hasta llegar a la versión final.
3. **Base de conocimiento:** Codificación en JSON de las variables y reglas del sistema de predicción de Cushing, con validación de esquemas.

4. **Lógica Difusa:** Diseño del wrapper que envuelve `skfuzzy` para ganar un mayor control sobre el sistema.
5. **API REST y módulos:** Desarrollo de `api.py` con FastAPI, módulos de datos e integración con Gemini para extracción de documentos y voz.
6. **Interfaz de usuario:** Implementación de la aplicación Next.js con los componentes de análisis, resultados y visualización de reglas activadas.
7. **Pruebas, despliegue y documentación** — Suite de tests, configuración de CI/CD con GitHub Actions y redacción final de la memoria.

3.7. Planificación temporal

El proyecto se desarrolló durante los meses de **noviembre de 2025 a junio de 2026**, con una dedicación media estimada de 20 horas semanales, compatible con la carga lectiva del último curso del grado. La figura 3.2 muestra el diagrama de Gantt.

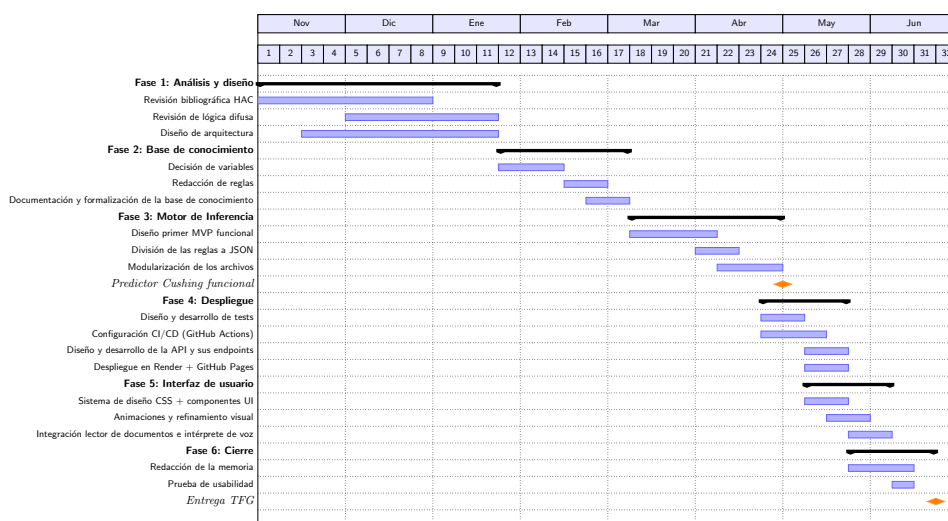


Figura 3.2: Diagrama de Gantt del proyecto C.O.C.O.

3.7.1. Distribución de horas por fase

La tabla 3.4 resume la dedicación estimada por fase, calculada a partir del registro de *commits* en el repositorio y del tiempo dedicado a la redacción de la memoria. El total asciende a aproximadamente **375 horas**.

Fase	Horas estimadas	%
Fase 1 — Análisis y diseño	70	18.7 %
Fase 2 — Motor de inferencia	75	20.0 %
Fase 3 — Base de conocimiento	65	17.3 %
Fase 4 — Despliegue	45	12.0 %
Fase 5 — Interfaz de usuario	40	10.7 %
Fase 6 — Cierre	80	21.3 %
Total	375	100 %

Cuadro 3.4: Distribución de horas de trabajo por fase del proyecto.

Las fases con mayor carga fueron las del motor de inferencia y la de cierre, lo que refleja la complejidad matemática del sistema difuso y el tiempo invertido en la redacción técnica de la memoria. La fase de base de conocimiento, aunque conceptualmente intensa por la revisión bibliográfica, se extendió más en el tiempo debido a la necesidad de asimilar la información de un nuevo campo de estudio.

3.8. Presupuesto

Debido a las limitaciones obvias por el carácter de ser un trabajo de final de grado, el presupuesto de este proyecto ha sido muy bajo, limitado a usar las herramientas y dispositivos que se han utilizado a lo largo de mi experiencia en el grado, así como a licencias gratuitas por ser estudiante.

Es por ello que en este apartado vamos a **simular** los costes si este proyecto se quisiese llevar a cabo en un entorno de producción profesional con más margen económico.

3.8.1. Costes de personal

Siguiendo los baremos habituales para un Trabajo de Fin de Grado, se considera la figura del ingeniero junior como categoría de referencia para el trabajo de implementación y la de tutor académico para las horas de supervisión.

La tarifa del desarrollador junior (aprox. 15€/h) es coherente con el rango habitual para puestos de prácticas en empresas tecnológicas de España. Las 10 horas de los tutores incluyen las reuniones de seguimiento, la revisión de la memoria y la orientación en las decisiones de diseño.

Rol	Tarifa (€/h)	Horas	Coste (€)
Desarrollador (autor del TFG)	15,00	375	5.625,00
Tutor académico	60,00	10	600,00
Cotutor académico	60,00	10	600,00
Subtotal personal			6.825,00

Cuadro 3.5: Costes de personal del proyecto.

3.8.2. Costes de hardware

Equipo	Precio (€)	Vida útil (años)	Uso (meses)	Coste amortizado (€)
Portátil de desarrollo	900,00	4	6	112,50
Subtotal hardware				112,50

Cuadro 3.6: Costes de hardware amortizados.

La amortización se calcula de forma lineal como:

$$\text{Coste amortizado} = \frac{\text{Precio compra}}{\text{Vida útil (meses)}} \times \text{Meses de uso}$$

3.8.3. Costes de *software* y servicios externos

En el proyecto actual todos los servicios utilizados son sobre su versión gratuita. En un entorno de producción en el que se buscase un alto impacto de nuestra plataforma sería necesario realizar los siguientes ajustes. Se calcula el precio en formato trimestral.

Servicio / Herramienta	Plan utilizado	Coste/mes (€)	Total (€)
GitHub (repositorio + Actions)	Gratis (2000 min/mes CI)	0,00	0,00
GitHub Pages (hosting frontend)	Gratis	0,00	0,00
Render	Pro (Pasa de 512MB a 4GBX2CPUs de RAM)	95	285
Gemini API (1M tokens)	De pago	≈2,80	≈8.40
Overleaf (redacción de memoria)	Gratis	0,00	0,00
VS Code (IDE)	Gratis	0,00	0,00
Python 3.12	Open Source	0,00	0,00
Librerías de frontend	Open Source	0,00	0,00
Librerías de backend	Open Source	0,00	0,00
Subtotal software			≈293.5

Cuadro 3.7: Costes de *software* y servicios externos.

Con la mejora de plan de la API de Gemini no daríamos la información de los perros examinados a Google para que mejore sus modelos, aumentando la privacidad de los mismos. Con la mejora del plan de Render, pasaríamos a poder soportar una carga de solicitudes mucho mayor, conseguiríamos que

nuestro servicio estuviera operativo de manera inmediata durante todo el día y nos permitiría crecer y añadir funcionalidades al proyecto.

Por último, cabe mencionar la posibilidad de hacer acuerdos con bases de datos veterinarias, opción que se contempló durante el desarrollo del proyecto, y usar esos datos para mejorar el modelo. Si se decidiese realizar dicha mejora, más allá del acuerdo institucional, habría que realizar un pago oscilante entre 300€-800€ según la cantidad de datos que se quisiera obtener. Se ha hecho mención de dicha opción, pero no se incluirá en el cálculo del coste final.

3.8.4. Resumen del presupuesto total

Categoría	Coste (€)
Personal	6.825,00
Hardware (amortizado)	112,50
<i>Software</i> y servicios externos	293.50
Subtotal	7.231
IVA (21 %)	1.518,51
TOTAL	8749.51

Cuadro 3.8: Resumen del presupuesto total del proyecto C.O.C.O.

El presupuesto total asciende a aproximadamente **9203.26 €** IVA incluido.

Capítulo 4

Implementación

4.1. Arquitectura del sistema

Se decidió aprovechar la modularización del sistema para darle una estructura que fuera extensible por la comunidad. De tal manera que, a la hora de darle forma al proyecto, se ha optado por crear C.O.C.O. (Cuidado y Observación Canina Optimizada), una plataforma desde la cual se puede estimar el riesgo de enfermedades en perros. La primera implementación será la que se detalla en este trabajo, el Síndrome de Cushing.

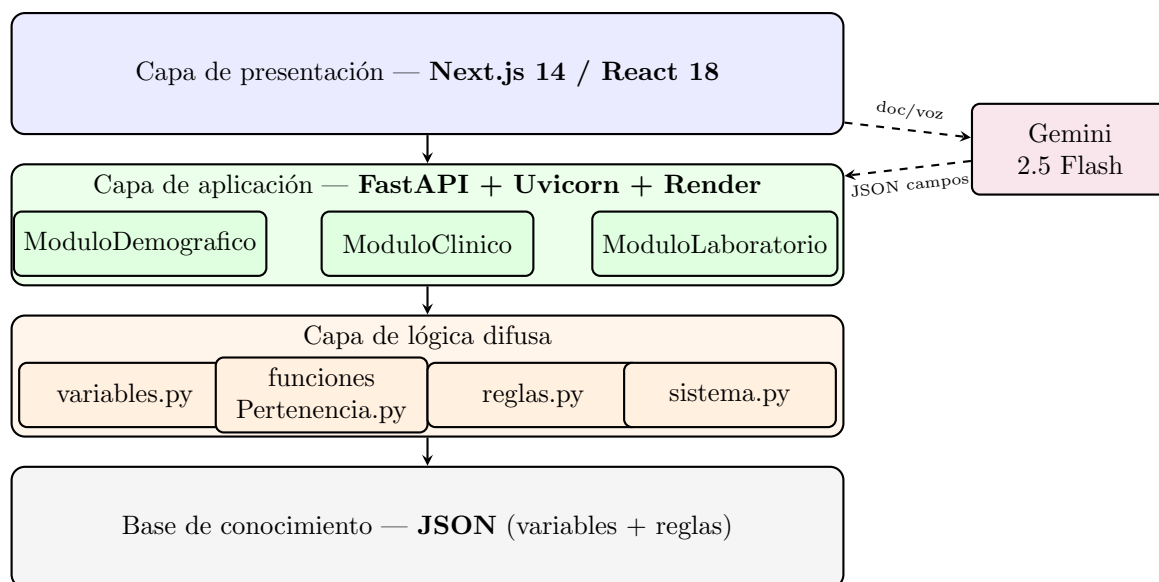


Figura 4.1: Arquitectura en capas de la plataforma C.O.C.O.

4.1.1. Capa de presentación

La interfaz de usuario se implementa como una **Single Page Application** con Next.js 14[27] en modo *static export* (salida HTML/CSS/JS pura, sin servidor Node.js en producción). Consta de tres rutas principales:

- `/`: Página de inicio con información de la metodología.
- `/analyze`: Formulario de análisis clínico con soporte de extracción por IA.
- `/results`: Visualización del resultado difuso y las reglas activadas.

4.1.2. Capa de aplicación

El backend expone una API REST con **FastAPI 0.136** [19] servido por **Uvicorn** [5]. Sus responsabilidades son:

1. Recibir y validar el payload clínico mediante modelos Pydantic[7].
2. Construir los módulos de datos (`ModuloDemografico`, `ModuloClinico`, `ModuloLaboratorio`).
3. Delegar la predicción al sistema de lógica difusa.
4. Actuar como intermediario con la API de Gemini [11] para la extracción de documentos y voz.
5. Serializar y devolver el resultado al frontend.

Los endpoints disponibles se resumen en la tabla 4.1.2.

Método	Ruta	Función
POST	<code>/predict/cushing</code>	Ejecuta inferencia difusa completa para HAC
POST	<code>/api/extraer-documento</code>	Extrae campos clínicos de un archivo PDF/CSV/Excel usando Gemini
POST	<code>/api/extraer-voz</code>	Extrae campos clínicos de un dictado de voz usando Gemini

Cuadro 4.1: Endpoints de la API REST de C.O.C.O.

4.1.3. Capa de lógica difusa

El paquete `backend/logicaDifusa` implementa el motor de inferencia de Mamdani como un *framework* propio sobre `scikit-fuzzy` [28]. Sus cinco módulos actúan de forma desacoplada y son completamente reutilizables para cualquier predictor (véase el apéndice A).

4.1.4. Base de conocimiento

El conocimiento clínico se almacena en **archivos JSON** dentro de `backend/conocimiento/cushing/`. Esta separación entre lógica e información permite modificar variables y reglas sin tocar código Python, y facilita la extensión a nuevas patologías.

4.2. Tecnologías y herramientas

La tabla 4.2 y la tabla 4.3 recogen las dependencias principales con sus versiones exactas tal como aparecen en `requirements.txt` y `package.json`.

Librería	Versión	Propósito
Python	3.12	Lenguaje base del backend
FastAPI	0.136.3	Framework REST, validación, documentación OpenAPI automática
Uvicorn	0.48.0	Servidor ASGI de alto rendimiento
Pydantic	2.13.4	Modelos de datos y validación de esquemas JSON
scikit-fuzzy	0.5.0	Funciones de pertenencia y defuzzificación
NumPy	2.4.5	Álgebra vectorial para universos y arrays de pertenencia
google-generativeai	≥0.8.0	Cliente oficial de la API de Gemini
python-dotenv	≥1.0.0	Gestión de variables de entorno
python-multipart	≥0.0.9	Soporte para subida de archivos multipart
pytest	9.0.3	Suite de tests automatizados
matplotlib	3.10.9	Visualización de funciones de pertenencia (desarrollo)

Cuadro 4.2: Dependencias principales del backend.

Librería / Herramienta	Versión	Propósito
Next.js	14	Framework React con enrutamiento, SSG y exportación estática
React	18	Biblioteca de interfaz de usuario basada en componentes
TypeScript	5	Tipado estático sobre JavaScript
Framer Motion	—	Animaciones declarativas y transiciones de página
Vanilla CSS	—	Sistema de diseño propio, sin preprocesador
Node.js	20	Entorno de ejecución para el proceso de build

Cuadro 4.3: Dependencias principales del frontend.

4.2.1. Justificación de las elecciones tecnológicas

FastAPI sobre Django/Flask : FastAPI genera documentación OpenAPI automática, soporta validación mediante Pydantic y tiene rendimiento comparable a frameworks asíncronos más complejos. Su sintaxis declarativa resulta más concisa que Flask para APIs con modelos de datos bien definidos.

Next.js en modo estático : Permite desplegar el frontend en GitHub Pages (alojamiento gratuito) sin necesidad de un servidor Node.js en producción. El enrutamiento basado en sistema de archivos simplifica la estructura del proyecto.

scikit-fuzzy como base : Es la librería de lógica difusa más madura del ecosistema Python (más de 1.3 M de descargas), con soporte activo y funciones contrastadas. El *wrapper* propio proporciona la abstracción necesaria sin perder acceso a las funciones nativas cuando se necesitan.

Gemini 2.5 Flash : Modelo multimodal capaz de leer PDFs, hojas de cálculo y texto libre de forma nativa, con soporte para respuesta estructurada (`response_schema`) que garantiza la coherencia del JSON devuelto.

JSON como base de conocimiento : Frente a bases de datos relacionales o formatos binarios, los archivos JSON son legibles, versionables con Git y modificables por personal clínico sin conocimientos de programación.

4.3. Desarrollo del motor de inferencia

El motor de inferencia es el núcleo del sistema. Se desarrolló de forma incremental, partiendo de la abstracción de variables y funciones de pertenencia hasta llegar a la métricas de confianza compuesta.

4.3.1. Diagrama de clases del motor de inferencia

La figura 4.2 muestra la jerarquía de clases del *framework* logicaDifusa y la figura 4.3 muestra cómo se integra con la capa de sistema y la API REST.

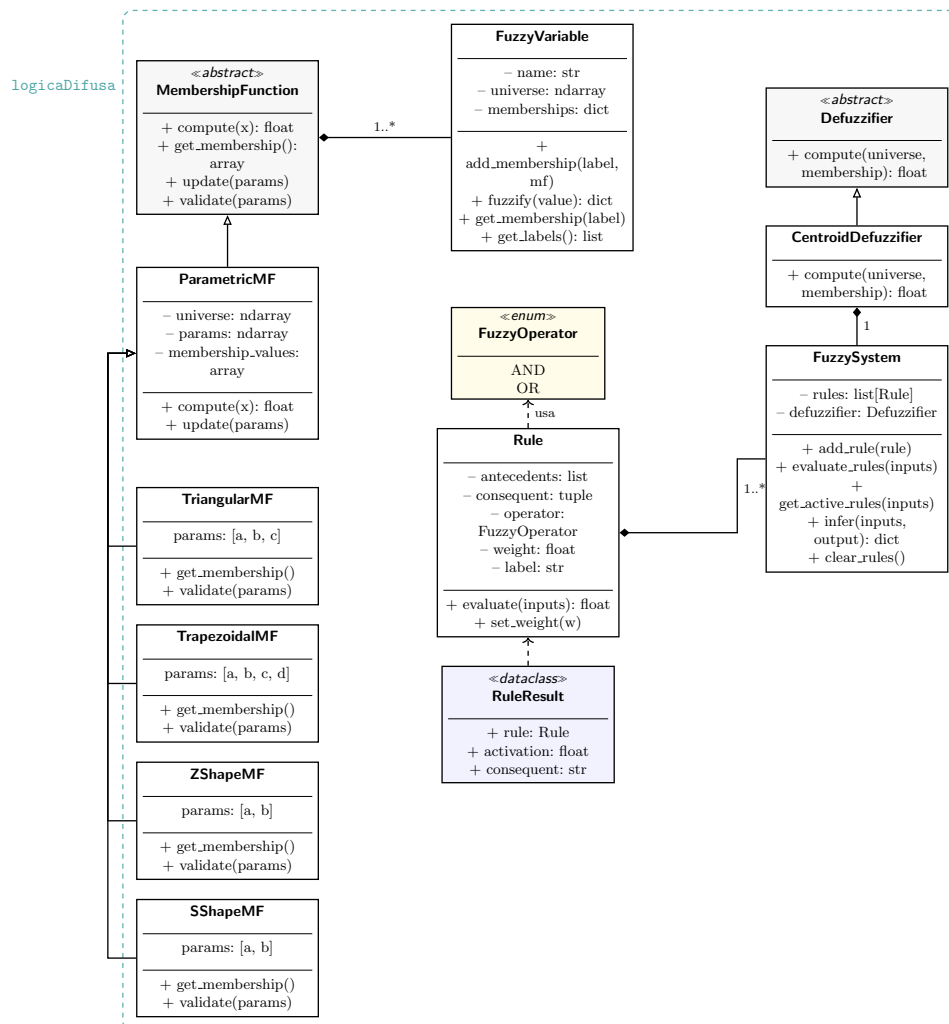


Figura 4.2: Diagrama de clases del *framework* de lógica difusa

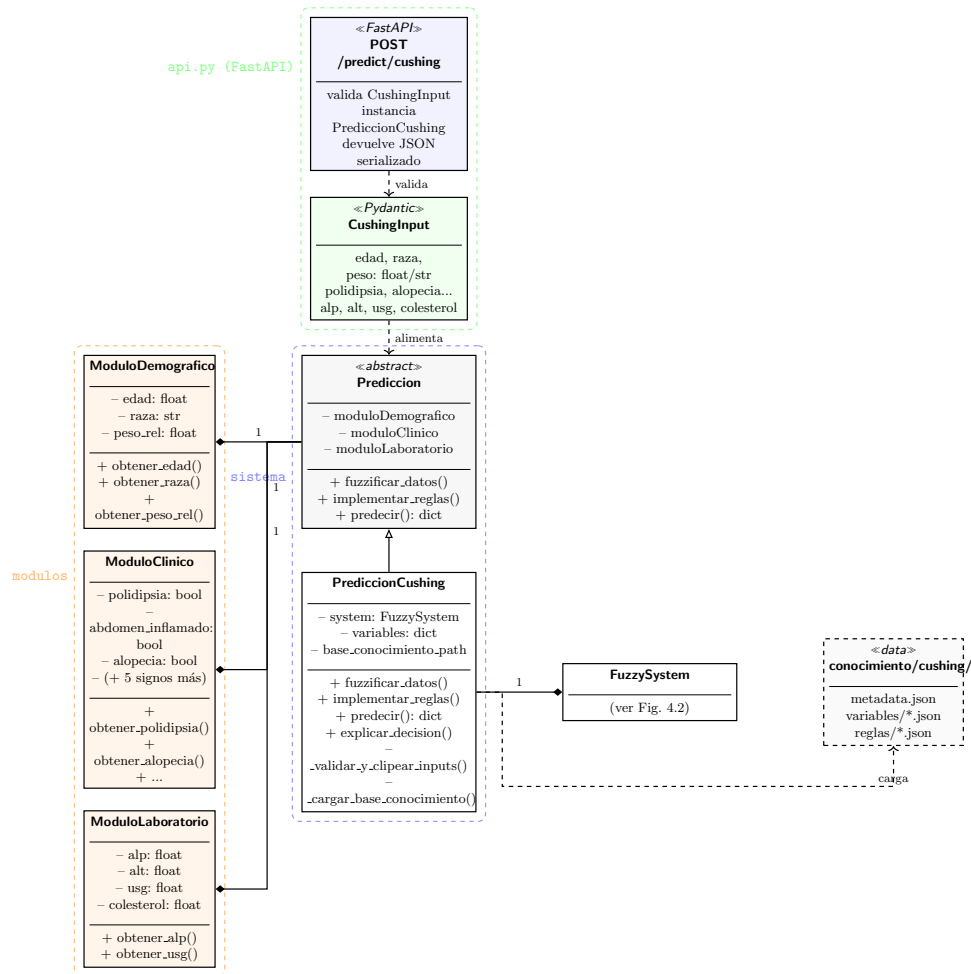


Figura 4.3: Diagrama de clases de la capa de sistema e integración.

4.3.2. Módulos de datos de entrada

Antes de que los datos lleguen al motor difuso se empaquetan en tres módulos de datos que encapsulan y validan la información del paciente:

- **ModuloDemografico** — Almacena *edad* (años), *raza* y *peso_rel* (porcentaje relativo respecto a la media de la raza).
- **ModuloClinico** — Almacena los ocho signos clínicos binarios: *polidipsia*, *abdomen_inflamado*, *alopecia*, *polifagia*, *poliuria*, *debilidad_muscular*, *piel_fina* y *jadeo*.
- **ModuloLaboratorio** — Almacena *alp* (U/L), *alt* (U/L), *usg*

(adimensional) y `colesterol` (mg/dL).

Todos ellos son simples contenedores con getters/setters que desacoplan la fuente de datos (API REST, CLI, tests) del motor difuso.

4.3.3. Carga dinámica de la base de conocimiento

La clase `PrediccionCushing` carga la base de conocimiento: El método `_cargar_base_conocimiento()` sólo ejecuta la lectura de disco la primera vez que se invoca y cachea el resultado en `self._base_conocimiento`. El proceso de carga recorre tres directorios:

1. `metadata.json`: Metadatos de la base (versión, autor, descripción), con validación de campos obligatorios.
2. `variables/*.json`: Todos los archivos JSON del directorio. Los archivos cuyo nombre contiene "consecuente" se clasifican como variables de salida; el resto, como antecedentes.
3. `reglas/*.json`: Arrays o objetos de reglas. Al cargar la clase valida el esquema de cada regla (campos `antecedentes`, `consecuente`, restricciones sobre `peso`).

4.3.4. Construcción del sistema fuzzy

A partir de la base de conocimiento cargada, `fuzzificar_datos()` instancia objetos `FuzzyVariable` con sus funciones de pertenencia correspondientes. El tipo de variable determina la clase de función de pertenencia a usar:

- Variables numéricas continuas (ALP, ALT, USG, colesterol, edad, peso relativo): funciones `trimf`, `zmf` o `smf` según el tipo de transición.
- Variables binarias de signos clínicos: dos términos (*no/si*) con funciones `trimf` en los extremos del universo $[0, 1]$.
- Variables categóricas (raza): clase interna `CategoricalMF` que devuelve $\mu = 1$ si la raza está en la lista de valores aceptados, $\mu = 0$ en caso contrario. Esto implementa una pertenencia *crisp* para información discreta sin universo numérico natural.

4.3.5. Validación de entradas y *clipping*

Antes de ejecutar la inferencia, `_validar_y_clipear_inputs()` comprueba que cada valor numérico esté dentro del universo declarado en el JSON. Si un valor supera el rango $[u_{\min}, u_{\max}]$, se recorta al límite más cercano con `numpy.clip` y se emite un `warnings.warn` para trazabilidad. Las variables categóricas (tipo `str`) se omiten de esta comprobación.

4.3.6. Pipeline de predicción

El método público `predecir()` orquesta el pipeline completo delegando en `FuzzySystem.infer()`, que implementa el ciclo de Mamdani en seis etapas (véase el apéndice A para la descripción matemática detallada):

1. **Fuzzificación** — `mf.compute(x)` interpola μ para cada entrada.
2. **Evaluación de reglas** — cada `Rule.evaluate(inputs)` aplica el operador lógico (AND = min, OR = max) y multiplica por el peso w .
3. **Filtrado** — se retienen solo las reglas con $\alpha > 0$.
4. **Implicación MIN** — la función de pertenencia del consecuente se recorta a α_k .
5. **Agregación MAX** — las salidas recortadas se combinan punto a punto.
6. **Defuzzificación centroide** — `fuzz.defuzz(..., 'centroid')` devuelve el centro de gravedad de la distribución agregada.

4.3.7. Métrica de confianza compuesta

El sistema calcula, además del valor crisp, una métrica de **confianza compuesta** formada por dos componentes:

Fuerza (f): Intensidad de activación Evita que reglas débiles diluyan reglas fuertes mediante una media autoponderada cuadrática:

$$f = \frac{\sum_i \alpha_i^2 \cdot w_i}{\sum_i \alpha_i \cdot w_i}$$

Consenso (c): Coherencia entre reglas Mide qué fracción de la activación ponderada total apunta al término consecuente dominante:

$$c = \frac{\max_t \sum_{i: \text{term}_i=t} \alpha_i \cdot w_i}{\sum_i \alpha_i \cdot w_i}$$

Confianza final

$$\text{confidence} = f \cdot c$$

Esta métrica solo es alta cuando las reglas activan con fuerza *y* están de acuerdo entre sí, lo que proporciona al clínico una segunda dimensión de interpretación más allá del valor de riesgo estimado.

4.3.8. Servicio de extracción con inteligencia artificial

El módulo `gemini.service.py` integra la API de **Gemini 2.5 Flash** para reducir la fricción en la entrada de datos clínicos. Soporta dos modalidades:

Extracción desde documento

La función `extract_document_data(file_bytes, mime_type)` recibe el contenido binario de un archivo (PDF, CSV o Excel) y lo envía a Gemini como *inline part* junto con el prompt de extracción. El modelo está configurado con:

- `response\_mime\_type="application/json"`: Fuerza respuesta JSON pura.
- `response\_schema=ExtractedDocument`: Esquema Pydantic que Gemini usa como contrato de salida, garantizando que los tipos sean correctos y que los campos no encontrados se devuelvan como `null`.
- `temperature=0.0`: Máximo determinismo para extracción factual.

Extracción desde voz

La función `extract_voice_data(transcript)` recibe la transcripción literal del dictado del veterinario. El prompt `VOICE_PROMPT` incluye una guía

de interpretación clínica que mapea expresiones coloquiales a campos del formulario. Por ejemplo:

- «*bebe mucho*», «*siempre en el bebedero*» → `polidipsia: true`
- «*ALP 400*», «*fosfatasa alcalina 400*» → `alp: 400.0`
- «*panza caída*», «*barrigón*» → `abdomen_inflamado: true`

En ambos casos, `get_missing_fields()` analiza el JSON devuelto e identifica los campos con valor `null`, que la interfaz resalta en ámbar para revisión manual.

4.4. Desarrollo de la interfaz de usuario

La interfaz sigue un sistema de diseño propio basado en **Vanilla CSS** con variables personalizadas (`--accent`, `--foreground`, `--muted`, etc.) y glassmorphism como estética principal. Los componentes se construyeron como unidades reutilizables en TypeScript con tipado estricto.

4.4.1. Componentes de la biblioteca de UI

La carpeta `frontend/src/components/ui/` contiene los siguientes componentes:

4.4.2. Página de análisis (/analyze)

Esta es la página central de la aplicación. Su estado se gestiona con `useState` de React [18] y coordina tres flujos de entrada de datos:

Flujo de extracción por documento

El componente `DocumentUploader` acepta archivos PDF, CSV y Excel mediante arrastrar y soltar o selección de archivo. Al recibir un archivo llama a `extractDocumentData(file)`, que hace `POST /api/extraer-documento` con un `FormData`. La respuesta actualiza el formulario mediante la función `handleDocumentExtracted(data, missing)`, que:

- Rellena los campos reconocidos convirtiendo tipos (booleanos directamente, números como cadenas para los `<input>`).
- Actualiza `missingFields`, un array de nombres de campo usados por la función `cls(field)` para aplicar el estilo ámbar (`inputMissingCls`) en los inputs no extraídos.

Componente	Responsabilidad
Button / ButtonLink	Botón primario, secundario y ghost; enlace estilizado; estado de carga
Card	Contenedor con borde suave y glassmorphism
Modal	Diálogo accesible con fondo semitransparente; bloqueable
ProgressBar	Barra de progreso animada para el estado de análisis
Loader	Indicador de carga con etiqueta
DiseaseBadge	Pastilla de etiqueta para la enfermedad activa
ResultCard	Tarjeta principal del resultado con nivel de riesgo y confianza
DocumentUploader	Zona de arrastrar/soltar con integración de extracción IA
VoiceRecorder	Grabación de audio, transcripción y envío al backend
UploadZone	Primitiva de zona de carga reutilizada por DocumentUploader

Cuadro 4.4: Componentes de la biblioteca de UI.

Flujo de dictado por voz

El componente `VoiceRecorder` utiliza la API `Web MediaRecorder` [21] para capturar audio del micrófono. Una vez el usuario detiene la grabación, el audio se transcribe mediante la API de reconocimiento de voz del navegador (`SpeechRecognition` [25]) y la transcripción resultante se envía a `POST /api/extraer-voz`. La respuesta se procesa con la misma función `handleDocumentExtracted`.

Flujo de introducción manual

El formulario está dividido en tres secciones con separadores visuales:

1. **Datos demográficos:** Tres campos numéricos/texto con validación inline.
2. **Signos clínicos:** Ocho tarjetas interactivas con checkbox; un contador en tiempo real (`activeSignsCount`) muestra cuántos signos están activos.
3. **Laboratorio:** Cuatro campos numéricos con etiquetas que incluyen las unidades (U/L, mg/dL).

El panel lateral «*Listo para analizar*» muestra tres indicadores de

progreso que verifican en tiempo real si cada bloque está completo, proporcionando retroalimentación visual antes de lanzar el análisis.

Lanzamiento del análisis

Al hacer clic en *Analizar ahora*, la función `startAnalysis()` valida que todos los campos obligatorios sean finitos y no vacíos. Si la validación pasa:

1. Se muestra un modal de progreso con barra animada.
2. Se llama a `analyzeCushing({diseaseId, payload})` del cliente API.
3. El resultado se serializa en `sessionStorage` con la clave `"analysisResult"`.
4. El router navega a `/results?d={diseaseId}`.

4.4.3. Página de resultados (/results)

La página recupera el resultado de `sessionStorage` (o genera un resultado de ejemplo si no existe). Presenta dos columnas:

Columna izquierda

- **ResultCard:** Muestra el valor $\text{crisp} \in [0, 1]$, el nivel de riesgo (etiqueta lingüística) y la confianza compuesta.
- **Recomendaciones:** Lista de acciones de seguimiento devuelta por el backend.
- **Aviso legal:** Recordatorio explícito de que el resultado no es un diagnóstico.

Columna derecha — Reglas activadas

Las reglas se ordenan de mayor a menor activación y se renderizan con una paleta de colores que depende del nivel de riesgo del consecuente (tabla 4.5).

Para cada regla se muestra su `label` descriptivo (campo del JSON de reglas), el nivel de consecuente con *badge* coloreado, una barra de activación proporcional a α , y el peso w de la regla. Esta sección constituye la **capa de explicabilidad** del sistema: el clínico puede identificar exactamente qué patrones clínicos han contribuido al nivel de riesgo estimado.

Nivel	Clave interna	Color de barra
Muy bajo	muy_bajo	Verde esmeralda (#10b981)
Bajo	bajo	Verde azulado (#14b8a6)
Medio	medio	Ámbar (#f59e0b)
Alto	alto	Naranja (#f97316)
Muy alto	muy_alto	Rosa/rojo (#f43f5e)

Cuadro 4.5: Paleta de colores por nivel de riesgo en la página de resultados.

4.4.4. Animaciones y experiencia de usuario

Para mejorar la percepción de calidad y dinamismo se emplean:

- **Framer Motion** [8]: Animaciones de entrada (`initial/animate`) con `whileInView` para activar transiciones al hacer scroll. Los componentes de la página de inicio aparecen con un efecto de *fade-up* coordinado.
- **CSS transitions**: Cambios de color y borde en las tarjetas de signos clínicos al marcarlas, y en los campos con estado ámbar.
- **Contador en tiempo real**: El número de signos activos se actualiza instantáneamente sin necesidad de enviar datos al servidor.

4.5. Integración y despliegue automático

C.O.C.O. implementa un pipeline de **Integración Continua y Despliegue Continuo** (CI/CD) mediante **GitHub Actions** [10]. El archivo `.github/workflows/deploy.yml` define cuatro *jobs* que se ejecutan secuencialmente con dependencias explícitas. La figura 4.4 muestra el flujo completo.

4.5.1. Job 1 — Tests del backend

Se ejecuta en **todas** las ramas (push y pull request). Levanta un runner `ubuntu-latest` con Python 3.12, instala las dependencias desde `backend/requirements.txt` con caché de pip, y ejecuta:

```
1 cd backend && python -m pytest tests/ -v
```

Si algún test falla, el pipeline se detiene y los *jobs* posteriores no se ejecutan. Esto garantiza que ningún código defectuoso llegue a producción.

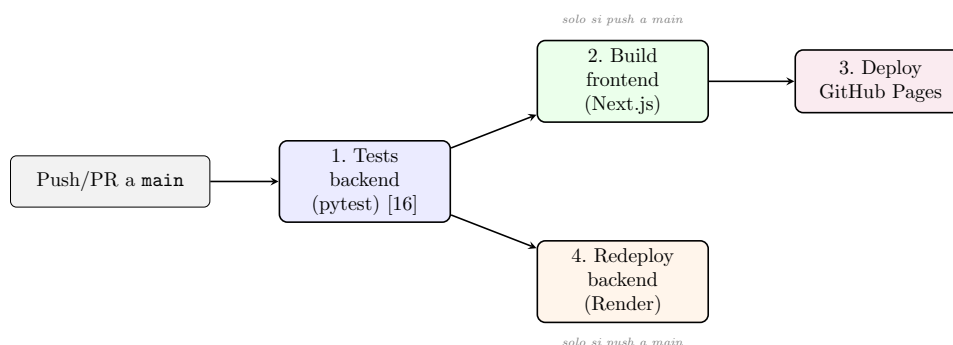


Figura 4.4: Pipeline CI/CD de GitHub Actions para C.O.C.O.

4.5.2. Job 2 — Build del frontend

Solo se ejecuta en push a `main`, y solo si el `job` de tests ha pasado (`needs: test`). El proceso:

1. Instala Node.js 20 con caché de npm.
2. Ejecuta `npm ci` para una instalación reproducible a partir de `package-lock.json`.
3. Inyecta la variable de entorno `NEXT_PUBLIC_API_BASE` desde los secretos del repositorio de GitHub. Esta variable apunta a la URL del backend en Render, garantizando que el build estático incluya la URL correcta sin exponerla en el código.
4. Ejecuta `npm run build`, que genera el directorio `frontend/out/` con la exportación estática de Next.js.
5. Sube `frontend/out/` como artefacto de Pages con `actions/upload-pages-artifact@v3`.

4.5.3. Job 3 — Despliegue en GitHub Pages

Depende del `job` de build (`needs: build-frontend`) y usa el artefacto subido en el paso anterior para desplegarlo en GitHub Pages mediante `actions/deploy-pages@v4`. El entorno `github-pages` expone la URL final del despliegue como variable de salida.

El control de concurrencia (`concurrency: group: "deploy", cancel-in-progress: false`) evita despliegues solapados: si hay un despliegue en curso, el nuevo espera hasta que el anterior termine.

4.5.4. Job 4 — Redespliegue del backend en Render

En paralelo con el build del frontend, este *job* notifica a **Render** [20] que debe redesplegar el servicio backend. Render es una plataforma PaaS que mantiene el servidor FastAPI activo de forma continua con HTTPS automático.

La notificación se realiza mediante un *deploy hook*: una URL secreta almacenada en los secretos del repositorio (`RENDER_DEPLOY_HOOK_URL`) que se activa con una petición POST usando `curl`:

```
1 curl -s --fail-with-body \  
2   "${{ secrets.RENDER_DEPLOY_HOOK_URL }}" \  
3   -X POST
```

El uso de `--fail-with-body` hace que el *job* falle si Render rechaza el hook, proporcionando visibilidad sobre errores de redespliegue.

4.5.5. Gestión de secretos

La tabla 4.6 lista los secretos de repositorio empleados en el pipeline.

Cuadro 4.6: Secretos de repositorio usados en el pipeline CI/CD.

Secreto	Uso
<code>NEXT_PUBLIC_API_BASE</code>	URL base del backend (ej. <code>https://coco.onrender.com</code>) inyectada en el build de Next.js
<code>RENDER_DEPLOY_HOOK_URL</code>	URL privada del deploy hook de Render
<code>GEMINI_API_KEY</code>	Clave de API de Google Gemini, inyectada en el entorno del backend en Render

4.5.6. Resultado del proceso de despliegue

El resultado final del pipeline es:

- **Frontend** publicado en `https://mamarco13.github.io/coco-vet-engine` como sitio estático, servido directamente por la CDN de GitHub Pages con HTTPS.
- **Backend** ejecutándose en Render con un servidor Uvicorn persistente, accesible desde el frontend sin política de CORS restrictiva (los orígenes permitidos están declarados explícitamente en `api.py`).

Cada *push* a la rama **main** desencadena el pipeline completo de forma automática, garantizando que la versión desplegada siempre corresponda al estado de la rama principal del repositorio y que ningún cambio llegue a producción sin pasar por la suite de tests.

Capítulo 5

Pruebas y validación

La verificación del correcto funcionamiento de C.O.C.O. se abordó desde dos perspectivas complementarias: una **perspectiva técnica**, mediante una suite de tests automatizados que cubre unitariamente cada componente del sistema y verifica la integración de extremo a extremo; y una **perspectiva clínica**, mediante la validación con casos reales y una prueba de usabilidad con usuarios del perfil objetivo.

5.1. Metodología de pruebas

La suite de tests se implementó con **pytest 9.0.3** y se organiza en cinco archivos dentro del directorio `backend/tests/`. El pipeline de CI/CD (véase la sección 4.5) ejecuta automáticamente toda la suite en cada push y pull request a la rama principal, garantizando que ningún cambio rompa el comportamiento esperado del sistema.

La tabla 5.1 ofrece una visión cuantitativa de la cobertura por archivo.

Archivo	Módulo bajo test	Tipo	Tests
<code>test_variables.py</code>	<code>FuzzyVariable</code>	Unitario	12
<code>test_reglas.py</code>	<code>Rule</code> , <code>FuzzyOperator</code>	Unitario	11
<code>test_sistema.py</code>	<code>FuzzySystem</code>	Componente	15
<code>test_prediccion_cushing.py</code>	<code>PrediccionCushing</code>	Componente + Integración	27
<code>test_api.py</code>	API REST (<code>/predict/cushing</code>)	Integración	5
Total	—	—	70

Cuadro 5.1: Resumen de la suite de tests automatizados.

Los tests se agrupan en tres niveles según la granularidad del componente verificado, siguiendo la pirámide de testing clásica: una base amplia de tests unitarios, una capa intermedia de tests de componente y un conjunto más reducido de tests de integración que verifican la comunicación entre capas[6].

5.1.1. Tests unitarios

Los tests unitarios verifican el comportamiento de las clases más atómicas del *framework* de lógica difusa de forma aislada, sin dependencias externas ni acceso a disco.

Tests de FuzzyVariable (test_variables.py)

Este archivo verifica la clase **FuzzyVariable**, responsable de almacenar el universo de discurso y el diccionario de funciones de pertenencia.

Los tests usan tres funciones de pytest:

```

1 @pytest.fixture
2 def universo():
3     return np.arange(0, 1.01, 0.01)
4
5 @pytest.fixture
6 def variable(universo):
7     return FuzzyVariable("test", universo)
8
9 @pytest.fixture
10 def mf_medio(universo):
11     return TriangularMF(universo, 0.2, 0.5, 0.8)

```

Listing 5.1: Fixtures de test_variables.py.

Clase de test	Aspectos verificados
TestAddGet	Añadir y recuperar una MF; acceso por [] (<code>__getitem__</code>); <code>__contains__</code> ; error al registrar etiqueta duplicada; error al acceder a etiqueta inexistente
TestFuzzify	Que <code>fuzzify()</code> devuelve todas las etiquetas como claves del dict; que el grado en el pico es 1,0; que fuera del soporte es 0,0; que en el flanco ascendente es 0,5 ($\pm 0,05$)
TestManagement	<code>get_labels()</code> ; <code>len()</code> con 0 y con múltiples MFs; <code>remove_membership()</code> ; error al eliminar etiqueta inexistente; <code>clear_memberships()</code>

Cuadro 5.2: Casos de prueba de test_variables.py.

Tests de Rule y FuzzyOperator (test_reglas.py)

Este archivo verifica la evaluación correcta de reglas difusas con operadores AND y OR, la aplicación del peso y los errores de validación.

```

1 def crear_antecedente(nombre, a, b, c):
2     universo = np.arange(0, 1.01, 0.01)
3     mf = TriangularMF(universo, a, b, c)
4     return nombre, mf

```

Listing 5.2: Helper `crear_antecedente()` de `test_reglas.py`.

Clase de test	Aspectos verificados
TestAND	AND devuelve el mínimo de las activaciones; AND con ambas variables en el pico devuelve 1,0; AND con una variable fuera del soporte devuelve 0,0
TestOR	OR devuelve el máximo de las activaciones; OR con ambas variables fuera del soporte devuelve 0,0
TestPeso	<code>weight=0,5</code> escala la activación a la mitad; <code>weight=2,0</code> duplica la activación; <code>set_weight()</code> actualiza correctamente; peso negativo lanza <code>ValueError</code>
TestErrores	Input faltante lanza <code>ValueError</code> con mensaje « <i>Falta input</i> »; regla sin antecedentes lanza excepción (comportamiento de <code>min([])</code> en NumPy)

Cuadro 5.3: Casos de prueba de `test_reglas.py`.

5.1.2. Tests de componente

Los tests de componente verifican subsistemas más amplios que integran varias clases, pero sin llegar a la capa HTTP ni al acceso real a disco.

Tests de FuzzySystem (test_sistema.py)

Este archivo verifica el motor de inferencia completo: estructura del resultado, corrección del etiquetado lingüístico, y comportamiento de la métrica de confianza compuesta.

Para evitar dependencia con los archivos JSON reales, los tests definen en memoria una variable `riesgo` idéntica al `consecuente.json` de producción:

```

1 def output_riesgo():
2     u = np.arange(0, 1.01, 0.01)
3     var = FuzzyVariable("riesgo", u)
4     var.add_membership("muy_bajo", ZShapeMF(u, 0.0, 0.06))
5     var.add_membership("bajo", TriangularMF(u, 0.02, 0.12, 0.24))
6     var.add_membership("medio", TriangularMF(u, 0.28, 0.50, 0.72))
7     var.add_membership("alto", TriangularMF(u, 0.76, 0.87, 0.94))
8     var.add_membership("muy_alto", SShapeMF(u, 0.96, 1.0))
9     return var

```

Listing 5.3: Variable de salida definida in-memory en `test_sistema.py`.

Clase de test	Aspectos verificados
TestInferEstructura	Que <code>infer()</code> devuelve las 8 claves esperadas (<code>crisp</code> , <code>label</code> , <code>etiqueta</code> , <code>confidence</code> , <code>fuerza</code> , <code>consenso</code> , <code>rules</code> , <code>aggregated</code>); <code>crisp</code> $\in [0, 1]$; <code>confidence</code> , <code>fuerza</code> , <code>consenso</code> $\in [0, 1]$; <code>rules</code> es lista no vacía
TestGetLabel	Que <code>_get_label()</code> devuelve "muy_bajo" en $x = 0,0$, "bajo" en $x = 0,12$, "medio" en $x = 0,5$, "alto" en $x = 0,87$, "muy_alto" en $x = 1,0$; clipping silencioso para valores fuera del universo ($x = 1,5$ y $x = -0,5$); que la etiqueta empieza por mayúscula
TestConfianza	Consenso = 1 cuando todas las reglas activas apuntan al mismo término; <code>consenso</code> < 1 cuando dos reglas apuntan a términos distintos (<i>alto/muy_alto</i>); <code>fuerza</code> > 0,8 cuando la única regla activa está en su pico; que una regla con <code>weight=2</code> domina sobre una con <code>weight=1</code>
TestErrores	Sin reglas activas: <code>confidence</code> , <code>fuerza</code> y <code>consenso</code> son 0, <code>rules</code> es lista vacía, y el resultado contiene <code>crisp</code> y <code>label</code> ; <code>clear_rules()</code> vacía el sistema

Cuadro 5.4: Casos de prueba de `test_sistema.py`.

Tests de PrediccionCushing (test_prediccion_cushing.py)

Este archivo es el más extenso de la suite. Verifica en profundidad la clase `PrediccionCushing`, que actúa como fachada del sistema completo. Se organiza en cuatro bloques:

Bloque 1 — Clipping de entradas (TestClipping) Verifica que `_validar_y_clipear_inputs()` recorta correctamente los valores numéricos fuera del universo declarado y emite un `warnings.warn` con la cadena «*Clipping*» en el mensaje. Los casos cubiertos son:

- Valor dentro del universo $\rightarrow$ no se modifica.
- Valor por encima del máximo ($4000 > 3000$) $\rightarrow$ se recorta a 3000 y se emite exactamente 1 warning.
- Valor por debajo del mínimo ($-50 < 0$) $\rightarrow$ se recorta a 0 y se emite exactamente 1 warning.
- Variable de tipo `str` (raza) $\rightarrow$ pasa intacta sin intentar recorte numérico.
- Variable no registrada en `self.variables` $\rightarrow$ pasa intacta.
- Múltiples variables, solo una fuera $\rightarrow$ exactamente 1 warning, la otra variable no se modifica.

Bloque 2 — Validación de metadatos (TestValidarMetadata) Verifica que `_validar_metadata()` acepta un diccionario completo y lanza `ValueError` si falta cualquiera de los tres campos obligatorios (`autor`, `version`, `descripcion`), con un mensaje de error que contiene el nombre del campo faltante.

Bloque 3 — Validación de variables (TestValidarVariable) Verifica la validación del esquema JSON de una variable difusa. Cubre 9 casos de error:

Listing 5.4: Ejemplo de caso de prueba de validación de variable.

```
1 def test_funcion_desconocida_lanza_error(self):
2     defn = self._def_valida()
3     defn["terminos"]["bajo"]["funcion"] = "gaussmf"
4     with pytest.raises(ValueError, match="gaussmf"):
5         self.p._validar_variable("alp", defn, "vars.json")
```

También cubre la validación de variables categóricas (presencia de "terminos" con valores de tipo lista) y el rechazo de variables categóricas sin "terminos".

Bloque 4 — Validación de reglas (TestValidarRegla) Verifica 9 casos de error sobre el esquema JSON de una regla: ausencia de **antecedentes** o **consecuente**, antecedentes no lista, antecedente sin **variable** o sin **termino**, consecuente sin **termino**, peso negativo y peso no numérico.

5.1.3. Tests de integración

Los tests de integración verifican el sistema de extremo a extremo, desde la entrada del usuario hasta la respuesta final, atravesando todas las capas.

Tests *end-to-end* de predicción (TestIntegracion)

La clase `TestIntegracion` en `test_prediccion_cushing.py` ejecuta predicciones reales sobre la base de conocimiento JSON de producción usando dos perfiles de paciente contrastantes.

Definido por la función auxiliar `predictor_completo()` con los valores por defecto:

Módulo	Variable	Valor
Demográfico	Edad	10 años
	Raza	Bichón Frisé
	Peso relativo	100 %
Clínico	Polidipsia	Sí
	Abdomen inflamado	Sí
	Alopecia	Sí
	Poliuria	Sí
	Polifagia	Sí
	Debilidad muscular	Sí
	Jadeo	Sí
Laboratorio	ALP	780 U/L
	ALT	220 U/L
	USG	1,012
	Colesterol	410 mg/dL

Cuadro 5.5: Perfil del caso de alto riesgo usado en los tests de integración.

El test `test_predecir_caso_alto_riesgo` verifica que para este perfil el sistema devuelve `label` $\in \{ \text{"alto"}, \text{"muy_alto"} \}$.

Perfil de bajo riesgo

```

1 def test_predecir_caso_bajo_riesgo(self):
2     p = predictor_completo(
3         demo={"edad": 3, "raza": "otro", "peso_rel": 100},
4         clin=dict(
5             polidipsia=False, abdomen_inflamado=False, alopecia=False,
6             poliuria=False, polifagia=False, debilidad_muscular=False,
7             piel_fina=False, jadeo=False
8         ),
9         lab={"alp": 80, "alt": 40, "usg": 1.030, "colesterol": 180}
10    )
11    result = p.predecir()
12    assert result["label"] in {"muy_bajo", "bajo", "medio"}

```

Listing 5.5: Caso de bajo riesgo en `test_prediccion_cushing.py`.

Por otro lado contamos también con el test contrario, que estudia un perfil de bajo riesgo: un paciente joven (3 años), sin ningún signo clínico, con analítica completamente dentro de los rangos normales (ALP 80 U/L, USG 1,030) debe obtener un nivel de riesgo *muy bajo*, *bajo* o, como máximo, *medio*, nunca *alto* ni *muy alto*.

Los tests `test_predecir_devuelve_claves_esperadas`, `test_predecir_crisp_en_rango`, `test_predecir_label_valido` y `test_predecir_confidence_entre_0_y_1` verifican que el diccionario devuelto por `predecir()` cumple el contrato de interfaz con el frontend: claves presentes, tipos correctos y rangos válidos.

Tests de la API HTTP (`test_api.py`)

Este archivo usa el `TestClient` de FastAPI, que permite enviar peticiones HTTP reales a la aplicación sin levantar un servidor de red, asegurando que la capa de serialización y validación Pydantic funciona correctamente.

```

1 def test_predict_cushing_ok_retorna_json_esperado(self):
2     response = client.post("/predict/cushing", json=payload_completo())
3
4     assert response.status_code == 200
5     assert response.headers["content-type"].startswith("application/json")
6
7     data = response.json()
8     claves = {"crisp", "label", "etiqueta", "confidence",
9              "fuerza", "consenso", "rules", "aggregated"}
10    assert claves.issubset(data.keys())
11    assert 0.0 <= data["crisp"] <= 1.0
12    assert 0.0 <= data["confidence"] <= 1.0
13    # Cada regla tiene activation, consequent y weight
14    for item in data["rules"]:
15        assert {"activation", "consequent", "weight"}.issubset(item.keys())

```

Listing 5.6: Test principal de la API (`test_api.py`).

Test	Payload	Resultado esperado
<code>test_predict_cushing_ok_retorna_json_esperado</code>	Completo con todos los campos	HTTP 200, JSON con 8 claves, rangos válidos
<code>test_predict_cushing_acepta_booleanos_omitidos</code>	Solo campos obligatorios (sin booleanos)	HTTP 200, rules como lista
<code>test_predict_cushing_rechaza_payload_incompleto</code>	Sin campo <code>alp</code>	HTTP 422 (Unprocessable Entity)
<code>test_predict_cushing_rechaza_tipo_invalido</code>	<code>alp = "no_es_numero"</code>	HTTP 422
<code>test_predict_cushing_devuelve_json_serializable</code>	Completo	HTTP 200, rules y aggregated son listas

Cuadro 5.6: Casos de prueba de `test_api.py`.

Los tests HTTP también verifican que el array **aggregated** (distribución difusa agregada) se serializa correctamente a lista de floats, ya que NumPy[14] arrays no son directamente serializables a JSON y requieren la conversión explícita implementada en la función `serialize_result()` de `api.py`.

5.1.4. Resumen de la cobertura

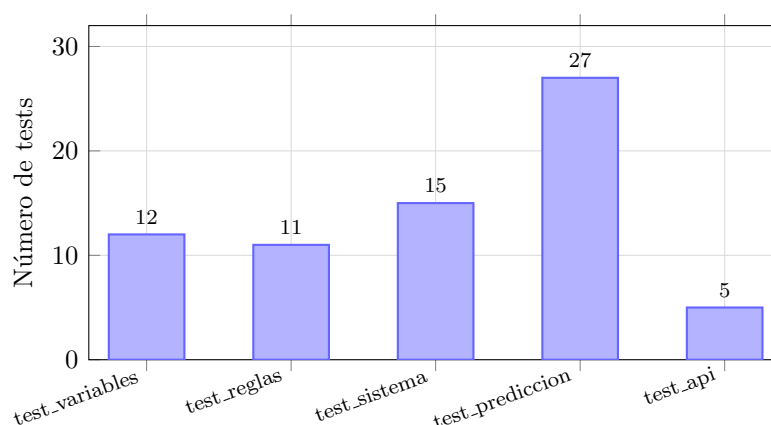


Figura 5.1: Distribución de tests por archivo.

5.2. Validación clínica

Además de las pruebas automatizadas, se llevó a cabo una validación clínica cualitativa con el propósito de evaluar el rendimiento y la viabilidad del sistema en un entorno operativo real. Para ello, se concertó y realizó una entrevista semiestructurada con un profesional del sector.

5.2.1. Perfil del experto

La evaluación externa fue realizada por Joaquín López, veterinario licenciado por la Universidad de Córdoba en 1995. El especialista cuenta con más de tres décadas de trayectoria profesional y actualmente ejerce como director de la Clínica Veterinaria de La Zubia (Granada), centro donde se desarrolló la sesión de validación y en el cual desempeña también las funciones de responsable de las áreas de medicina interna, hospitalización y cirugía.

5.2.2. Validación mediante casos clínicos representativos

El proceso de validación manual se articuló en torno a dos escenarios clínicos diferenciados, correspondientes a pacientes caninos con distintos niveles de sospecha y probabilidad de padecer Hiperadrenocorticismismo (HAC).

Caso 1 — Paciente de riesgo moderado

En el primer escenario, se introdujeron manualmente a través del formulario de la interfaz los datos de un paciente de raza Yorkshire Terrier de edad media, con sobrepeso leve y sintomatología compatible (jadeo), junto con los resultados de una analítica laboratorial estándar. El sistema determinó un nivel de riesgo medio (55 %) con un índice de confianza del 67 %. De acuerdo con el criterio del experto, esta estimación se adecuó estrictamente al diagnóstico presuntivo esperado.

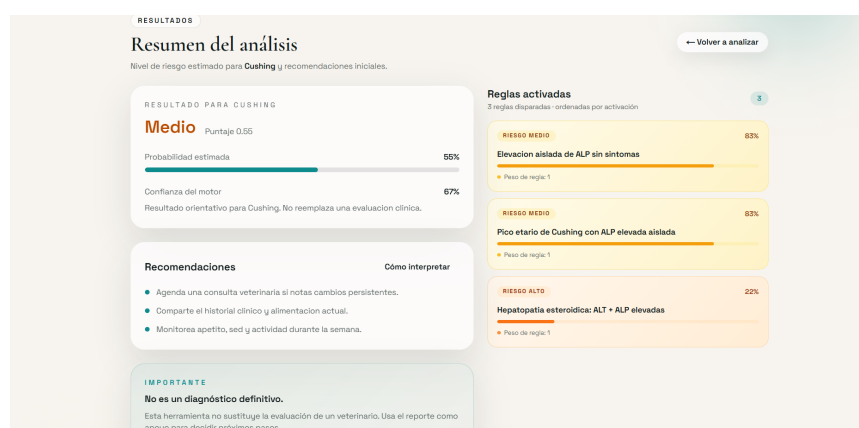


Figura 5.2: Resultados del análisis y predicción para el primer caso de estudio (riesgo moderado).

Caso 2 — Paciente de riesgo muy alto

Para el segundo escenario, se utilizó el módulo lector de archivos PDF para procesar un informe simulado. Este documento compilaba el historial de un paciente de raza Golden Retriever de edad avanzada que manifestaba un cuadro clínico completo de HAC y alteraciones laboratoriales severas. La plataforma arrojó un diagnóstico de riesgo alto (86 %) con un grado de confianza del 95 %. El especialista ratificó la coherencia y precisión de dicha valoración.

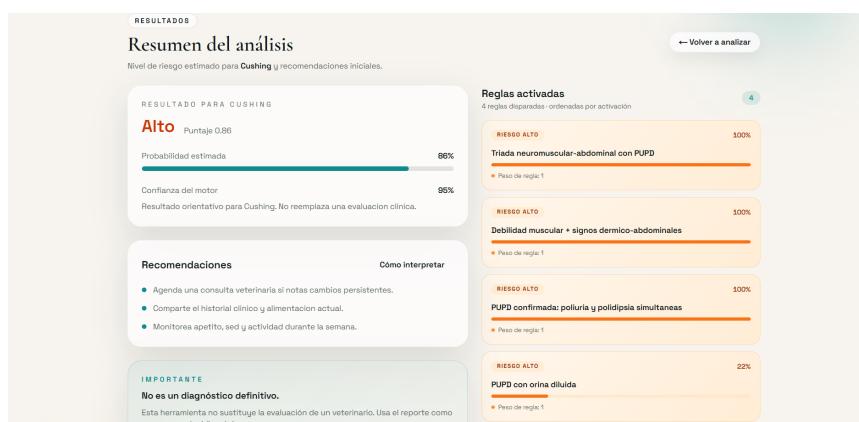


Figura 5.3: Resultados del análisis y predicción para el segundo caso de estudio (riesgo alto mediante lectura de PDF).

5.2.3. Pruebas de usabilidad

Se llevó a cabo una prueba de usabilidad con el propósito principal de evaluar si el personal clínico es capaz de interactuar con la plataforma de forma autónoma, intuitiva y sin necesidad de formación previa, garantizando asimismo que los resultados e inferencias generados por el sistema sean fácilmente interpretables.

Metodología

La sesión de pruebas se estructuró con antelación mediante un protocolo definido para optimizar el tiempo de la entrevista. Con el fin de mitigar el sesgo de aprendizaje y garantizar la espontaneidad y validez de las observaciones, el especialista no tuvo acceso previo a la interfaz ni conocimiento de los escenarios de prueba hasta el momento exacto de la evaluación.

Tareas evaluadas

El participante ejecutó de manera secuencial el siguiente conjunto de tareas predefinidas:

1. **Tarea 1 — Navegación inicial:** Reconocimiento de la interfaz de usuario, comprensión de sus funcionalidades principales y validación de los mecanismos de acceso a los diferentes módulos.
2. **Tarea 2 — Evaluación de las variables:** Verificación de la

idoneidad y rigor de las variables clínicas seleccionadas, así como de la idoneidad de sus unidades de medida correspondientes.

3. **Tarea 3 — Análisis manual:** Evaluación de la ergonomía y usabilidad en la inserción directa de datos clínicos a través del formulario, comprobando la fluidez del flujo de trabajo asociado.
4. **Tarea 4 — Análisis mediante documentos:** Validación de la funcionalidad de carga y procesamiento automático de archivos, evaluando la precisión del reconocimiento óptimo de datos clínicos y la flexibilidad del sistema para su posterior edición manual.
5. **Tarea 5 — Interpretación de resultados:** Evaluación de la claridad, fidelidad y comprensibilidad de las métricas de riesgo y los intervalos de confianza generados en la predicción del Síndrome de Cushing (HAC).
6. **Tarea 6 — Evaluación del motor de reglas:** Inspección y comprobación directa de las reglas lógicas que componen la base de conocimiento para verificar la cobertura técnica y la coherencia clínica ante los diferentes escenarios.

Resultados observados

Los resultados cualitativos y cuantitativos recopilados durante la ejecución de las tareas analizadas se sintetizan en la Tabla 5.7.

Tarea	Resultado general	Observaciones
1	Correcto	El usuario centró su atención en el flujo de contenido clínico, obviando la sección técnica orientada a desarrolladores para la configuración de nuevas patologías.
2	Parcialmente correcto	Se sugirió modificar las unidades de medida de las enzimas hepáticas ALP y ALT a mg/dL, por ser la métrica convencional de referencia en el ámbito clínico cotidiano.
3	Correcto	El evaluador destacó positivamente la agilidad del flujo de trabajo y la baja latencia en el procesamiento del formulario.
4	Correcto	Se constató un alto grado de satisfacción respecto a la versatilidad de la función de parseo y la posibilidad crítica de enmendar los datos extraídos a posteriori.
5	Correcto	El experto ratificó la coherencia diagnóstica y la lógica probabilística de las predicciones arrojadas en ambos casos clínicos de estudio.
6	Correcto	Se confirmó la idoneidad, rigor y correcta correspondencia de las reglas lógicas seleccionadas por el sistema.

Cuadro 5.7: Resultados de la prueba de usabilidad desglosados por tareas evaluadas.

Conclusiones de la prueba

A partir de los resultados expuestos en la Tabla 5.7, es posible concluir que la plataforma presenta un alto índice de usabilidad y madurez en su interfaz de usuario, cumpliendo satisfactoriamente los objetivos de autonomía y accesibilidad técnica planteados inicialmente.

La respuesta positiva en tareas críticas como el análisis manual (Tarea 3) y la ingesta automatizada de documentos (Tarea 4) pone de manifiesto que el sistema no solo reduce la carga cognitiva del profesional, sino que además mitiga las barreras de entrada tecnológicas gracias a un flujo de trabajo fluido y ágil. Asimismo, la total consonancia en la interpretación de resultados y reglas (Tareas 5 y 6) valida la solidez clínica del motor de inferencia ante profesionales experimentados.

Por último, el comportamiento observado en la Tarea 1 demuestra la correcta separación del diseño entre la lógica de desarrollo y la práctica médica. Como aspecto fundamental de mejora de cara a futuras iteraciones,

se ha identificado la necesidad de flexibilizar la parametrización de unidades del sistema (Tarea 2). Adaptar las métricas de las enzimas hepáticas a los estándares más extendidos del sector (como mg/dL) supondrá el paso definitivo para maximizar la interoperabilidad de la herramienta e impulsar su adopción definitiva en entornos clínicos reales.

Capítulo 6

Conclusiones y trabajo futuro

6.1. Conclusiones

El presente Trabajo de Fin de Grado proponía una solución para el diagnóstico del hiperadrenocorticismo (HAC) canino haciendo uso de la lógica difusa. Para ello se ha abordado el diseño, implementación y despliegue de C.O.C.O., una plataforma open-source y extensible por la comunidad, de apoyo al diagnóstico en enfermedades caninas. A lo largo del proyecto se han alcanzado los objetivos planteados inicialmente y se han obtenido conclusiones de distinta naturaleza que se exponen a continuación.

6.1.1. Conclusiones técnico-científicas

La lógica difusa es un formalismo adecuado para el razonamiento clínico veterinario.

La incertidumbre inherente al diagnóstico clínico —signos que se presentan con distintos grados de intensidad, valores de laboratorio en rangos limítrofes, razas con predisposición variable— no es bien capturada por los sistemas binarios tradicionales. El motor de inferencia de Mamdani implementado en C.O.C.O. modela esta gradualidad de forma natural: un perro con ALP de 210 U/L puede pertenecer simultáneamente a los conjuntos «normal» y «elevada» con grados de pertenencia proporcionales, y el sistema integra esa ambigüedad en la estimación final de riesgo en lugar de descartarla.

La arquitectura basada en JSON desacopla el conocimiento de la lógica.

La separación entre la base de conocimiento (`conocimiento/cushing/*.json`) y el motor de inferencia (`logicaDifusa/`) es la decisión de diseño con mayor impacto en la mantenibilidad del sistema. Las reglas y variables pueden ser revisadas por personal clínico familiarizado con JSON sin necesidad de modificar código Python, y la validación de esquemas garantiza que los cambios no introducen inconsistencias en tiempo de ejecución. Esta arquitectura ha demostrado ser correcta: el sistema de HAC se implementó y refinó en múltiples iteraciones sin necesidad de alterar el motor de inferencia.

6.1.2. Conclusiones de diseño de *software*

El framework `logicaDifusa` es reutilizable y extensible.

Los cinco módulos del paquete (`variables`, `funcionesPertenencia`, `reglas`, `sistema`, `defuzzification`) constituyen un *framework* de propósito general que puede utilizarse para modelar cualquier problema de inferencia difusa, independientemente del dominio. La jerarquía de herencia de `MembershipFunction` y el patrón de fachada de `FuzzySystem` son abstracciones sólidas que permiten añadir nuevas funciones de pertenencia o nuevos métodos de defuzzificación sin afectar al código existente.

La integración con Gemini 2.5 Flash reduce significativamente la fricción de entrada.

La posibilidad de subir un informe analítico en PDF o dictar los datos en lenguaje natural y obtener el formulario pre-llenado automáticamente es la característica con mayor impacto práctico en la experiencia del clínico. El uso del parámetro `response_schema` de la API garantiza que la extracción es determinista y estructuralmente correcta, eliminando la necesidad de *parsing* frágil de texto libre.

El pipeline CI/CD garantiza la fiabilidad del despliegue continuo.

La configuración de GitHub Actions con cuatro *jobs* secuenciales asegura que cualquier cambio en la rama principal pasa por los 70 tests automatizados antes de llegar a producción. Esta práctica, habitual en entornos profesionales, resulta especialmente valiosa en un proyecto unipersonal donde no existe revisión de código por pares. Los tests actúan como red de seguridad frente a regresiones involuntarias.

Los patrones de diseño orientados a objetos son el sustento estructural del sistema.

La solidez de C.O.C.O. no reside únicamente en los algoritmos de inferencia difusa, sino en la forma en que éstos se organizan. A lo largo del proyecto se aplicaron de forma consciente y coherente cuatro patrones de diseño clásicos que, en conjunto, han hecho posible que el sistema sea extensible, testable y mantenible.

El patrón **Método Plantilla** (*Template Method*) [9] vertebró toda la capa de predicción: la clase abstracta `Prediccion` define el algoritmo de tres pasos —`fuzzificar_datos()`, `implementar_reglas()`, `predecir()`— que cualquier predictor debe respetar, delegando en las subclases los detalles específicos de cada patología. Gracias a este patrón, añadir un predictor nuevo no requiere reescribir el flujo de control; basta con implementar los tres métodos en una clase concreta.

El patrón **Estrategia** (*Strategy*) [9] es quizás el más importante del sistema; gobierna las funciones de pertenencia, el uso de los predictores y la defuzzificación: `ParametricMF` y `Defuzzifier` son abstracciones intercambiables cuyos algoritmos concretos —`TriangularMF`, `ZShapeMF`, `SShapeMF`, `CentroidDefuzzifier`— pueden sustituirse en tiempo de ejecución sin modificar el código que los consume. Esta separación fue clave en la fase de diseño de la base de conocimiento: cambiar la forma de una función de pertenencia en el JSON equivale a cambiar de estrategia de forma transparente.

El patrón **Fachada** (*Facade*) [9] encapsula toda la complejidad del motor de inferencia detrás de `FuzzySystem.infer()`: el código cliente —tanto el endpoint de FastAPI como los tests de integración— obtiene un resultado completo (`crisp`, `confidence`, `rules`) con una única llamada, sin necesidad de conocer los detalles del ciclo de fuzzificación, implicación MIN, agregación MAX ni defuzzificación centroide.

En definitiva, los patrones de diseño no fueron una elección académica, sino una necesidad práctica: sin la abstracción del *Template Method* no habría extensibilidad, sin la *Strategy* no habría flexibilidad en las MFs ni posibilidad de elegir con tanta facilidad la patología a examinar, sin la *Facade* los tests serían frágiles, y sin los módulos de datos la arquitectura no sería testable de forma aislada. Su aplicación sistemática es, en buena medida, la razón por la que el sistema puede crecer hacia nuevas patologías sin acumular deuda técnica.

6.1.3. Conclusiones clínicas

El sistema es coherente con la literatura científica de referencia.

Las reglas difusas codificadas en la base de conocimiento se fundamentan en estudios publicados con alta relevancia clínica [4], [17], [13]. Los casos de prueba extremos —perfil de alto riesgo con todos los signos activos y analítica alterada, perfil de bajo riesgo sin signos y analítica normal— producen resultados en los niveles de riesgo esperados, lo que valida la coherencia global del sistema con el conocimiento clínico experto.

C.O.C.O. no sustituye al veterinario.

Es fundamental subrayar que el sistema es una herramienta de **apoyo a la decisión**, no un sistema de diagnóstico autónomo. El resultado difuso es un indicador de sospecha clínica que debe integrarse en la evaluación completa del paciente: exploración física, historia clínica detallada y, en última instancia, pruebas diagnósticas específicas (test de supresión con dexametasona, estimulación con ACTH). El aviso legal presente en la interfaz y el carácter probabilístico de los resultados refuerzan esta distinción.

6.2. Líneas de mejora

Las siguientes líneas de trabajo se identificaron durante el desarrollo y la prueba de usabilidad del sistema. Se organizan según su horizonte temporal estimado de implementación.

6.2.1. Corto plazo — Mejoras inmediatas sobre el sistema actual

Ampliación del vocabulario del dictado por voz.

La prueba de usabilidad detectó que expresiones muy coloquiales no incluidas en el prompt `VOICE_PROMPT` de `gemini_service.py` no se mapeaban correctamente a campos del formulario. Una mejora directa sería enriquecer la guía de interpretación clínica del prompt con más variantes lingüísticas y regionalismos, posiblemente mediante evaluación con un corpus de transcripciones reales.

Mensaje explicativo para los campos en ámbar.

Los participantes de la prueba de usabilidad solicitaron un texto descriptivo junto al indicador de color ámbar que explique que el campo no fue encontrado en el documento y requiere revisión manual. Esta mejora es trivial de implementar y mejoraría significativamente la experiencia de los usuarios menos familiarizados con la herramienta.

6.2.2. Medio plazo — Extensión de la plataforma**Sistema de historial de pacientes.**

Actualmente la plataforma no persiste información entre sesiones. Una mejora de alto valor clínico sería añadir un módulo de gestión de pacientes que permita almacenar el historial de análisis por paciente, registrar la evolución del nivel de riesgo a lo largo del tiempo y comparar resultados antes y después de un tratamiento. Esto requeriría introducir una base de datos (p. ej., PostgreSQL) y un sistema de autenticación básico.

Aprendizaje híbrido difuso-ML para calibración de parámetros.

Los parámetros de las funciones de pertenencia (puntos de soporte de las MFs) se definieron actualmente de forma experta a partir de la literatura clínica. Una línea de mejora sería aplicar técnicas de optimización —algoritmos genéticos o gradiente estocástico— sobre un conjunto de casos clínicos etiquetados para calibrar automáticamente los parámetros, manteniendo la interpretabilidad de la estructura de reglas pero mejorando la precisión de la estimación.

6.2.3. Largo plazo — Evolución hacia un sistema clínico completo**Integración con sistemas de gestión veterinaria.**

Las clínicas veterinarias utilizan software de gestión (GVET, PetSoft, Qvet, etc.) que almacena analíticas e historiales en formatos propietarios. La integración de C.O.C.O. con estas plataformas mediante APIs o importación directa de los ficheros de analítica eliminaría completamente la necesidad de introducir datos manualmente, convirtiendo el análisis difuso en un paso automático del flujo de trabajo clínico.

Evaluación clínica prospectiva.

La validación definitiva del sistema requeriría un estudio prospectivo con casos clínicos reales, anonimizados y con diagnóstico confirmado mediante pruebas de referencia (test de supresión con dexametasona de dosis bajas, estimulación con ACTH). Dicho estudio permitiría calcular métricas de rendimiento diagnóstico (sensibilidad, especificidad, área bajo la curva ROC) y comparar la estimación difusa con la del clínico, proporcionando una validación externa rigurosa del sistema.

6.2.4. Privatización de los datos.

Para que la plataforma pudiera funcionar en producción, sería recomendable evitar compartir los datos de los perros con Google. Para ello tenemos dos opciones, pasar a usar una API Gemini de pago (Pequeño gasto mensual, idóneo para empezar); o establecer una IA local en un servidor propio usando tecnologías como Ollama (Gasto grande pero un único pago, solo si el proyecto ya estuviera en producción y fuera autosostenible)

Bibliografía

- [1] D. J. Anderson. *Kanban: Successful Evolutionary Change for Your Technology Business*. Blue Hole Press, 2010.
- [2] J. M. Aznar Aznar and A. Loste Montoya. *Diabetes mellitus y Síndrome de Cushing. Problemática de la presencia conjunta en perros*. Trabajo de fin de grado, Universidad de Zaragoza, VET, Departamento de Patología Animal, Área de Medicina y Cirugía Animal, Zaragoza, España, 2015.
- [3] E. Behrend, H. Kooistra, R. Nelson, C. Reusch, and J. Scott-Moncrieff. Diagnosis of spontaneous canine hyperadrenocorticism: 2012 acvim consensus statement (small animal). *Journal of Veterinary Internal Medicine*, 27(6):1292–1304, 11 2013.
- [4] V. Boghian. Clinical biochemistry of cushing syndrome in dogs. *Revista de Chimie*, 71:320–324, 01 2001.
- [5] T. Christie et al. Unicorn: The lightning-fast asgi server, 2017.
- [6] M. Cohn. *Succeeding with Agile: Software Development Using Scrum*. Addison-Wesley Professional, 2009.
- [7] S. Colvin et al. Pydantic: Data validation using python type hints, 2017.
- [8] Framer B.V. Framer motion: Open source, production-ready animation and gesture library for react, 2019.
- [9] E. Gamma, R. Helm, R. Johnson, and J. Vlissides. *Design Patterns: Elements of Reusable Object-Oriented Software*. Addison-Wesley, 1994.
- [10] GitHub, Inc. Github actions: Automate your workflow from idea to production, 2018.
- [11] Google DeepMind. Gemini 2.5 flash, 2026. Multimodal AI Model.
- [12] D. S. Greco. Cushing disease (pituitary-dependent hyperadrenocorticism) in animals, 2024/09 2024.
- [13] D. S. Greco. Cushing disease (pituitary-dependent hyperadrenocorticism) in animals, 2024/09 2024.
- [14] C. R. Harris, K. J. Millman, S. J. van der Walt, et al. Numpy: The fundamental package for scientific computing with python, 2020. Artículo principal publicado en Nature 585, 357–362 (2020).

- [15] J. M. Hoffman, B. N. Lourenço, D. E. L. Promislow, and K. E. Creevy. Canine hyperadrenocorticism associations with signalment, selected comorbidities and mortality within north american veterinary teaching hospitals. *Journal of Small Animal Practice*, 59(11):681–690, 2018.
- [16] H. Krekel, B. Oliveira, R. Pfannschmidt, F. Bruynooghe, B. Laughner, and F. Bruhin. *pytest: helps you write better programs*, 2004.
- [17] F. Martins, G. Carvalho, L. Jesus, Pöppel, and F. González. Epidemiological, clinical, and laboratory aspects in a case series of canine hyperadrenocorticism: 115 cases (2010-2014). *Pesquisa Veterinária Brasileira*, 39:900–908, 11 2019.
- [18] Meta Platforms, Inc. *React: The library for web and native user interfaces*, 2013.
- [19] S. Ramírez. *Fastapi: Web framework for building apis with python*, 2018.
- [20] Render Services, Inc. *Render: Unified cloud to build and run all your apps*, 2019.
- [21] A. Roach et al. *Mediastream recording: W3c working draft*, 2017.
- [22] I. Schofield, D. Brodbelt, N. Kennedy, S. Niessen, D. Church, R. Geddes, and D. O'Neill. Machine-learning based prediction of cushing's syndrome in dogs attending uk primary-care veterinary practice. *Scientific Reports*, 11, 04 2021.
- [23] I. Schofield, D. C. Brodbelt, S. J. M. Niessen, D. B. Church, R. F. Geddes, N. Kennedy, and D. G. O'Neill. Development and internal validation of a prediction tool to aid the diagnosis of cushing's syndrome in dogs attending primary-care practice. *Journal of Veterinary Internal Medicine*, 34(6):2306–2318, 11 2020.
- [24] K. Schwaber and M. Beedle. *Agile Software Development with Scrum*. Prentice Hall, 2002.
- [25] G. Shires and H. Wennborg. *Web speech api specification*, 2012. W3C Community Group Draft Report.
- [26] B. Vasile. Clinical biochemistry of cushing syndrome in dogs. *Revista de Chimie*, 71(3):320–324, 2020.
- [27] Vercel Inc. *Next.js: The react framework for the web*, 2016.
- [28] J. Warner, X. Miskouridou, N. Twomey, J. Unpingco, et al. *scikit-fuzzy: Fuzzy logic toolkit for scipy*, 2019. Versión 0.4.2.

Apéndice A

Sistema de lógica difusa aplicado al proyecto

El objetivo de este apéndice es describir cómo se implementa la lógica difusa en el sistema de apoyo al diagnóstico de hiperadrenocorticismismo (HAC) canino. La meta es transformar conocimiento clínico impreciso en un procedimiento matemático interpretable, manteniendo la coherencia con el razonamiento veterinario.

Para ello se desarrolló un *framework* propio, ubicado en el paquete `backend/logicaDifusa`, que actúa como capa de abstracción sobre la librería `scikit-fuzzy` [28]. Esto permite definir variables, funciones de pertenencia, reglas y el motor de inferencia de forma modular, desacoplada y extensible, sin quedar ligado a la librería subyacente.

A.1. Arquitectura general del paquete

El paquete está organizado en cinco módulos con responsabilidades bien delimitadas (véase la figura A.1):

- `variables.py` — Define la clase `FuzzyVariable`, que representa una variable lingüística con su universo de discurso y su diccionario de funciones de pertenencia.
- `funcionesPertenencia.py` — Implementa la jerarquía de clases de funciones de pertenencia (*membership functions*).
- `reglas.py` — Define `Rule` y el enumerado `FuzzyOperator` para construir reglas difusas con antecedentes y consecuentes.

- `sistema.py` — Contiene `FuzzySystem`, el motor de inferencia que orquesta fuzzificación, evaluación de reglas, agregación y defuzzificación.
- `defuzzification.py` — Ofrece distintos defuzzificadores intercambiables mediante el patrón *Estrategia*.

Los patrones de diseño [9] aplicados de forma transversal son:

- **Estrategia:** Las funciones de pertenencia y los defuzzificadores son intercambiables en tiempo de ejecución.
- **Template Method parcial:** La clase abstracta `ParametricMF` implementa `compute()` y `update()` reutilizables; las subclases solo definen `get_membership()` y `validate()`.
- **Composición:** `FuzzyVariable` agrega funciones de pertenencia; `FuzzySystem` agrega reglas.
- **Fachada:** `FuzzySystem.infer()` ofrece un único punto de entrada al pipeline completo.

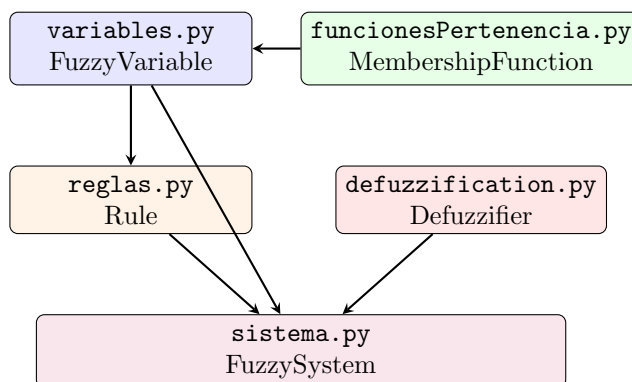


Figura A.1: Dependencias entre los módulos del paquete `logicaDifusa`.

A.2. Variables lingüísticas

En lógica difusa, cada variable de entrada (o salida) se modela como una *variable lingüística*, definida por la terna:

$$V = (U, T, \mu)$$

donde U es el universo de discurso (dominio numérico), T es el conjunto de términos lingüísticos y μ contiene las funciones de pertenencia asociadas a cada término.

La clase `FuzzyVariable` implementa esta abstracción:

- `add_membership(label, mf)`: Registra una función de pertenencia bajo una etiqueta lingüística, impidiendo duplicados.
- `fuzzify(value)`: Calcula el grado de pertenencia del valor `value` para *todas* las etiquetas, devolviendo un diccionario `{etiqueta:  $\mu$ }`.
- `get_membership(label)` / `__getitem__`: Acceso directo a una función de pertenencia concreta.
- `plot()`: Visualización de todas las funciones de pertenencia sobre el universo.

Las variables empleadas en el sistema de diagnóstico de HAC son, entre otras:

- **Edad** ($U = [0, 18]$ años).
- **ALP normalizada** (valor relativo al límite superior de referencia del laboratorio).
- **USG** (densidad urinaria, como evidencia de dilución o no dilución).
- **Signos clínicos** (polidipsia, alopecia endocrina, distensión abdominal, etc.).
- **Sospecha HAC** (variable de salida continua en $[0, 1]$).

La ventaja de esta formulación es que permite representar estados intermedios sin forzar decisiones binarias.

A.3. Funciones de pertenencia

Las funciones de pertenencia convierten un valor numérico en un grado de pertenencia:

$$\mu_A : U \rightarrow [0, 1]$$

La jerarquía de clases diseñada sigue el patrón *Template Method*: la clase abstracta `MembershipFunction` define la interfaz con cuatro

métodos obligatorios (`compute`, `get_membership`, `update`, `validate`). La clase intermedia `ParametricMF` proporciona implementaciones reutilizables de `compute()` —mediante interpolación con `fuzz.interp_membership`— y `update()`, delegando a `skfuzzy` la generación del array de valores. Las subclases concretas solo aportan la lógica específica de su forma:

Clase	Forma	Parámetros	Restricción
<code>TriangularMF</code>	Triángulo	a, b, c	$a \leq b \leq c$
<code>TrapezoidalMF</code>	Trapecio	a, b, c, d	$a \leq b \leq c \leq d$
<code>ZShapeMF</code>	Curva en Z	a, b	$a < b$
<code>SShapeMF</code>	Curva en S	a, b	$a < b$

Cuadro A.1: Funciones de pertenencia implementadas en `funcionesPertenencia.py`.

Todas las clases validan sus parámetros en el constructor y mantienen un **caché** interno del array de pertenencia, que se recalcula únicamente al invocar `update()`. Esto evita recomputaciones innecesarias en cada consulta.

A continuación, se adjuntan las figuras de las distintas funciones de pertenencia de todas las variables del proyecto:

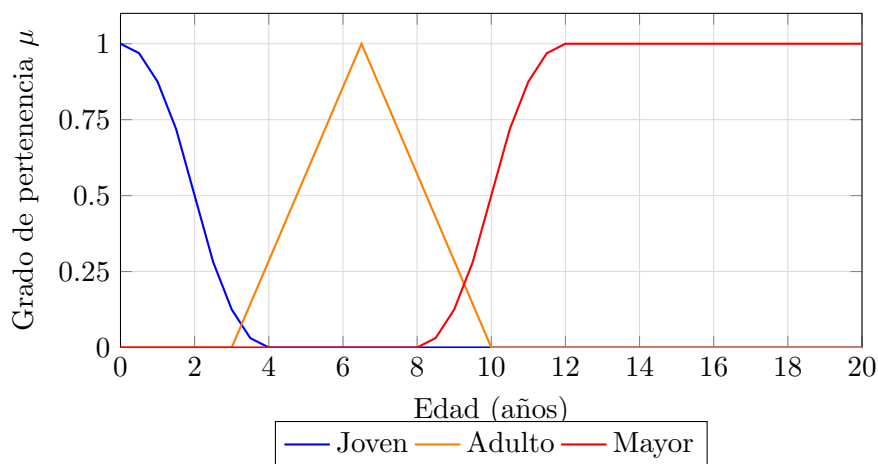


Figura A.2: Funciones de pertenencia para la variable lingüística **Edad**.

Tipos de funciones: elección y justificación

- **Trapezoidal:** Usada para términos extremos (*joven*, *senior*) o cuando la pertenencia máxima se mantiene durante un rango: un animal claramente joven sigue siendo joven aunque tenga dos o tres años.

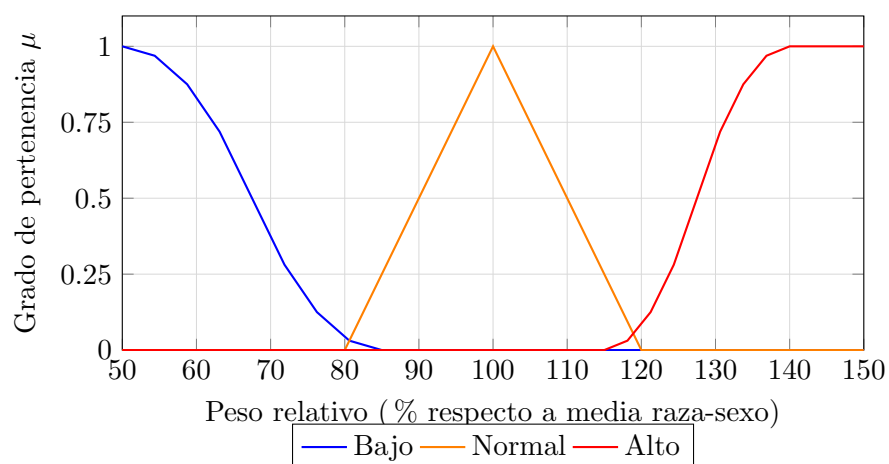


Figura A.3: Funciones de pertenencia para la variable lingüística **Peso relativo**.

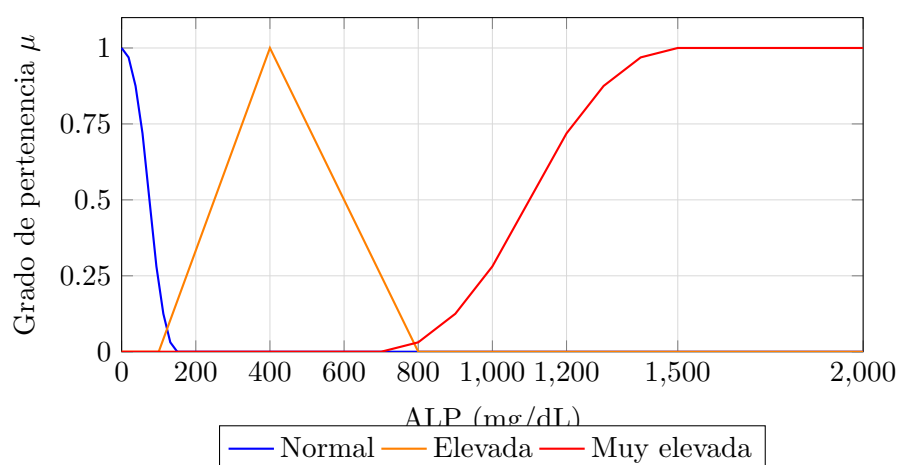


Figura A.4: Funciones de pertenencia para la variable lingüística **ALP**.

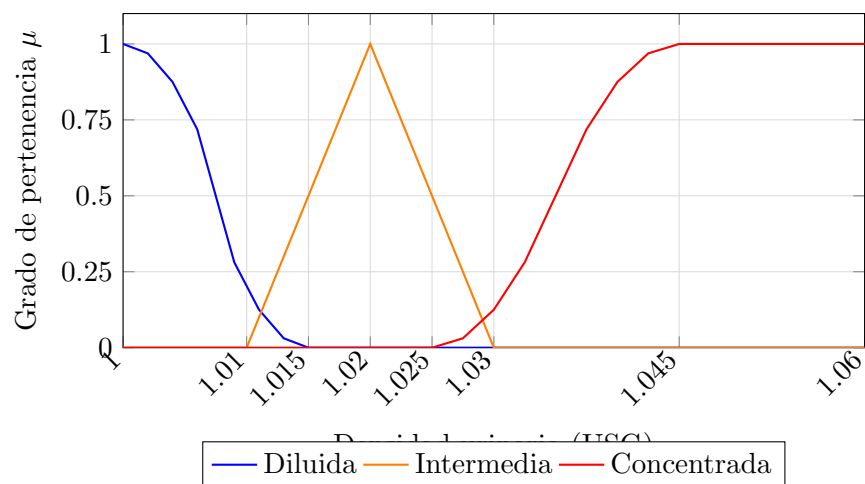


Figura A.5: Funciones de pertenencia para la variable lingüística **USG**.

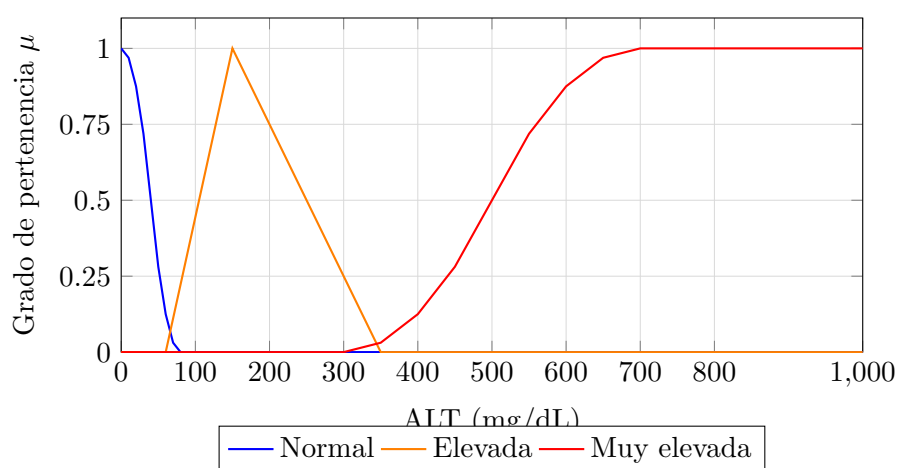


Figura A.6: Funciones de pertenencia para la variable lingüística **ALT**.

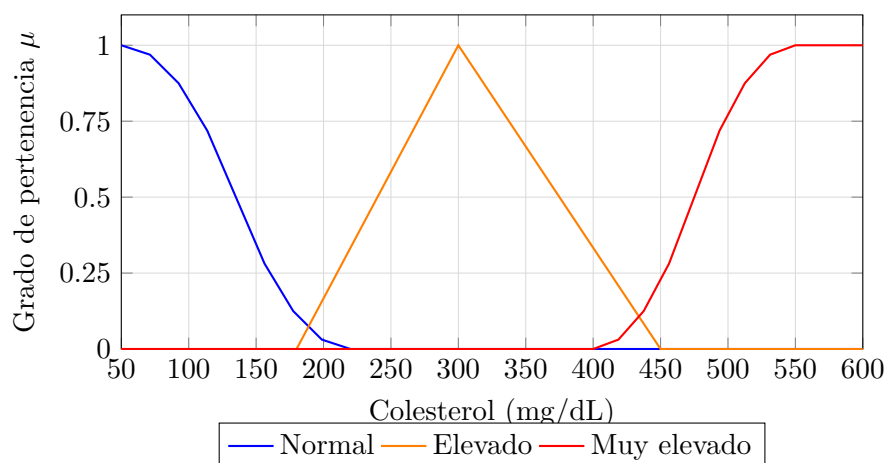


Figura A.7: Funciones de pertenencia para la variable lingüística **Colesterol**.

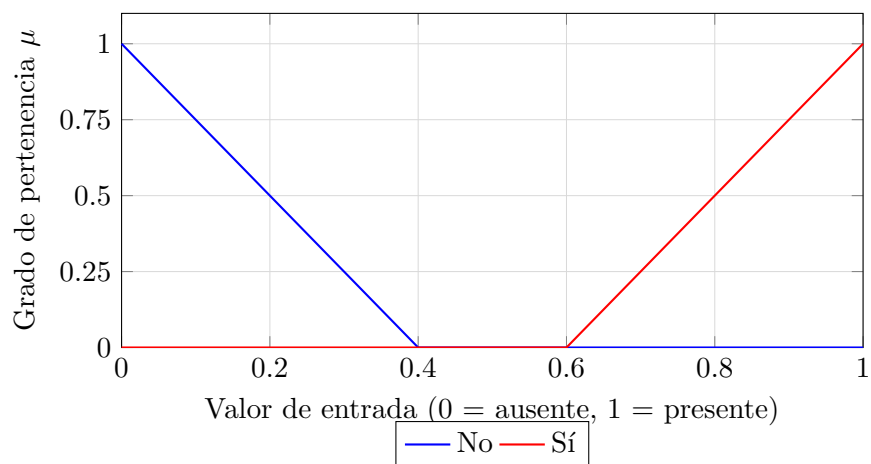


Figura A.8: Funciones de pertenencia comunes a las variables clínicas binarias: **polidipsia, poliuria, abdomen pendular, alopecia, debilidad muscular, piel fina, polifagia y jadeo**.

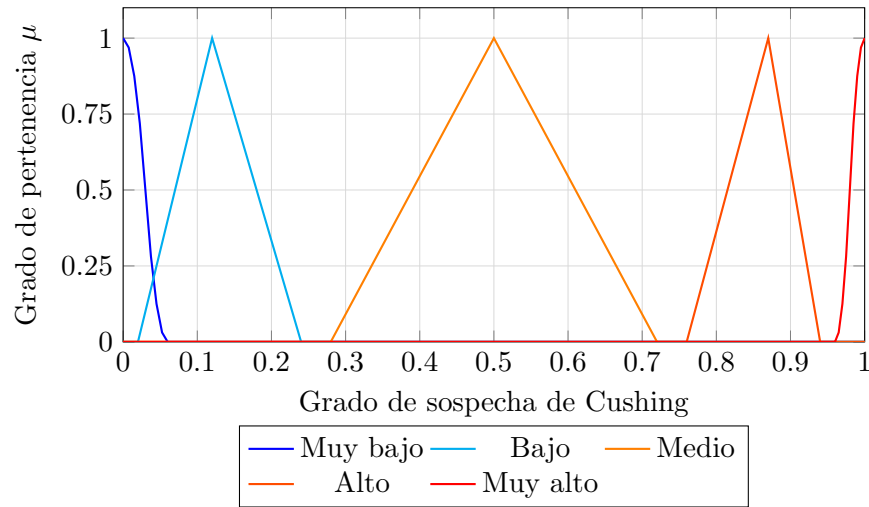


Figura A.9: Funciones de pertenencia para la variable lingüística de salida **Riesgo**.

- **Triangular:** Usada para términos intermedios (*adulto*), donde la transición es simétrica y el pico está bien definido.
- **Z-shape / S-shape:** Apropriadas para variables continuas con un único punto de transición suave (por ejemplo, la ALP normalizada en la frontera entre *normal* y *elevada*).

A.4. Reglas difusas

El conocimiento clínico se codifica en la clase `Rule` mediante reglas del tipo:

SI ⟨antecedente⟩ ENTONCES ⟨consecuente⟩

Cada regla tiene:

- **Antecedentes:** Lista de pares (`nombre_variable`, `función_de_pertenencia`).
- **Consecuente:** Par (`nombre_variable_salida`, `término_lingüístico`).
- **Operador:** `FuzzyOperator.AND` o `FuzzyOperator.OR` (enumerado Python).
- **Peso** ($w \geq 0$): Importancia relativa de la regla dentro del sistema.
- **Etiqueta** (`label`): Texto legible para trazabilidad clínica.

Evaluación de una regla

El método `Rule.evaluate(inputs)` computa la activación en dos pasos:

1. **Fuzzificación individual:** para cada antecedente se calcula $\mu_i = \text{mf.compute}(x_i)$.
2. **Combinación con operador lógico:**

$$\alpha_{\text{AND}} = \min(\mu_1, \mu_2, \dots, \mu_n) \quad \alpha_{\text{OR}} = \max(\mu_1, \mu_2, \dots, \mu_n)$$

3. **Aplicación del peso:**

$$\alpha_{\text{final}} = \alpha \cdot w$$

Este esquema sigue la semántica de Mamdani para conectivos lógicos difusos.

Ejemplos de reglas aplicadas al HAC:

- **R1:** Si *ALP es marcadamente elevada* **AND** *polidipsia es presente*, entonces *Sospecha_HAC es alta*.
- **R2:** Si *edad es senior* **AND** *distensión abdominal es presente*, entonces *Sospecha_HAC es media-alta*.
- **R3:** Si *vómitos son prominentes* **OR** *prurito es dominante*, entonces *Sospecha_HAC tiende a disminuir*.

A.5. Motor de inferencia (FuzzySystem)

`FuzzySystem` implementa el patrón *Fachada*: el método público `infer(inputs, output_variable)` ejecuta el pipeline completo y devuelve un diccionario estructurado con todos los resultados.

A.5.1. Pipeline de inferencia

El método `infer()` sigue seis etapas:

1. **Fuzzificación:** Se obtienen los grados de pertenencia de cada valor de entrada respecto a sus términos lingüísticos llamando a `mf.compute(x)`.
2. **Evaluación de reglas:** `evaluate_rules(inputs)` itera sobre todas las reglas y almacena cada activación en un `RuleResult` (*dataclass* que agrupa la regla, su activación y su consecuente).

3. **Filtrado de reglas activas:** `get_active_rules(inputs)` retiene únicamente aquellas con $\alpha > 0$, lo que evita procesamiento innecesario.
4. **Implicación (MIN):** Para cada regla activa, la función de pertenencia del consecuente se recorta al nivel de activación de la regla:

$$\mu'_k(y) = \min(\alpha_k, \mu_{\text{consecuente}_k}(y))$$

5. **Agregación (MAX):** Las salidas recortadas de todas las reglas se combinan punto a punto:

$$\mu_{\text{agg}}(y) = \max_k \mu'_k(y)$$

6. **Defuzzificación:** Se obtiene un valor crisp a partir de la distribución agregada (véase la sección A.6).

A.5.2. Resultado devuelto

El método `infer()` devuelve un diccionario con los siguientes campos:

- `crisp` — Valor numérico defuzzificado en $[0, 1]$.
- `label` — Clave interna del término dominante (ej. "muy_alto").
- `etiqueta` — Texto legible para interfaz clínica (ej. "Muy alto").
- `confidence` — Métrica de confianza compuesta (véase la sección A.7).
- `fuerza` — Componente de fuerza de activación.
- `consenso` — Componente de acuerdo entre reglas.
- `rules` — Lista de `RuleResult` activos (trazabilidad).
- `aggregated` — Array con la distribución difusa agregada.

A.6. Defuzzificación

El módulo `defuzzification.py` implementa el patrón *Estrategia*: todos los defuzzificadores heredan de la clase abstracta `Defuzzifier` y delegan el cálculo en `fuzz.defuzz()` de `scikit-fuzzy`.

En la implementación actual se emplea por defecto el método **centroide**:

Clase	Método	Descripción
CentroidDefuzzifier	Centroide	Centro de gravedad de μ_{agg}
BisectorDefuzzifier	Bisector	Punto que divide el área en dos mitades iguales
MOMDefuzzifier	Mean of Maximum	Promedio de los valores con μ máxima
SOMDefuzzifier	Smallest of Maximum	Menor valor con μ máxima
LOMDefuzzifier	Largest of Maximum	Mayor valor con μ máxima

Cuadro A.2: Métodos de defuzzificación disponibles.

$$y^* = \frac{\int_U y \cdot \mu_{agg}(y) dy}{\int_U \mu_{agg}(y) dy}$$

Este método es el más extendido en aplicaciones médicas y de diagnóstico porque suaviza el efecto de reglas con activaciones parciales y produce salidas continuas y estables.

Gracias a la abstracción **Defuzzifier**, el defuzzificador puede sustituirse en tiempo de ejecución sin modificar el motor de inferencia, lo que facilita la experimentación y el ajuste del sistema.

A.7. Métrica de confianza compuesta

Además del valor crisp, **FuzzySystem** calcula una **confianza compuesta** que cuantifica tanto la intensidad de activación del sistema como el grado de acuerdo entre las reglas activas. Esta métrica complementa la salida difusa y permite al sistema comunicar al clínico no solo *qué* sospecha, sino *con qué seguridad*.

Fuerza (f)

La fuerza mide cuán intensamente han activado las reglas activas en conjunto. Para evitar que reglas débiles diluyan reglas fuertes se usa una media autoponderada:

$$f = \frac{\sum_i \alpha_i^2 \cdot w_i}{\sum_i \alpha_i \cdot w_i}$$

donde α_i es la activación de la regla i y w_i su peso. Esta formulación eleva el peso relativo de las reglas más activas de forma cuadrática.

Consenso (c)

El consenso mide en qué medida las reglas activas apuntan al mismo término de salida (coherencia diagnóstica). Se calcula como la fracción de la activación ponderada total que corresponde al término consecuente dominante:

$$c = \frac{\max_t \sum_{i: \text{term}_i=t} \alpha_i \cdot w_i}{\sum_i \alpha_i \cdot w_i}$$

Un consenso bajo indica que reglas con consecuentes distintos están activándose simultáneamente con fuerzas similares, lo que refleja ambigüedad diagnóstica.

Confianza final

La confianza final es el producto de ambas componentes:

$$\boxed{\text{confidence} = f \cdot c}$$

Esta métrica solo es alta cuando las reglas activan *fuertemente y están de acuerdo* entre sí. Su valor puede mostrarse al clínico en porcentaje para facilitar la interpretación, y puede emplearse para decidir si el sistema requiere confirmación diagnóstica adicional.

A.8. Etiquetado lingüístico del resultado

Una vez obtenido el valor crisp, el método `_get_label()` determina la etiqueta lingüística dominante interpolando el valor en cada función de pertenencia de la variable de salida y seleccionando el término con mayor grado μ . Se aplica un recorte defensivo (*clipping*) para manejar valores ligeramente fuera del universo numérico sin lanzar errores.

La etiqueta se formatea automáticamente para la interfaz clínica: los guiones bajos se sustituyen por espacios y se aplica capitalización (ej. "muy_alto" $\rightarrow$ "Muy alto").

A.9. Resumen aplicado al TFG

En este proyecto, la lógica difusa permite pasar de observaciones heterogéneas —demográficas, clínicas y laboratoriales— a una estimación única, continua e interpretable de sospecha de HAC. El diseño arquitectónico del paquete garantiza que:

- Las **funciones de pertenencia** son intercambiables y parametrizables sin alterar el resto del sistema.
- Las **reglas** son trazables: el campo **rules** del resultado expone qué reglas han contribuido al diagnóstico y con qué activación.
- El **defuzzificador** puede sustituirse de forma transparente para experimentar con distintas estrategias.
- La **confianza compuesta** ofrece información adicional sobre la solidez del diagnóstico, lo que es esencial para una herramienta de apoyo a la decisión veterinaria.

Apéndice B

Guía de uso de la plataforma C.O.C.O.

Este apéndice está dirigido a dos perfiles de usuario:

- **Personal clínico** que desee analizar un caso de Cushing canino mediante la interfaz web (sección B.1).
- **Desarrolladores o investigadores** que quieran extender la plataforma con un predictor para una patología distinta (sección B.2).

Aviso legal

C.O.C.O. es una herramienta de **apoyo a la decisión**, no un diagnóstico definitivo. Cualquier resultado debe interpretarse en el contexto de la evaluación clínica completa por parte de un veterinario colegiado.

B.1. Análisis de un caso clínico

B.1.1. Acceso a la plataforma

La aplicación está disponible en <https://mamarco13.github.io/coco-vet-engine>. No requiere registro ni instalación; basta con un navegador moderno. La página principal ofrece un resumen de la metodología y el botón *Analizar caso* que da acceso al formulario.

B.1.2. Flujo general de uso

El proceso de análisis consta de tres etapas:

1. **Entrada de datos:** Introducción de los parámetros del paciente.
2. **Análisis difuso:** Ejecución del motor de inferencia en el servidor.
3. **Interpretación de resultados:** Lectura del nivel de riesgo, confianza y reglas activadas.

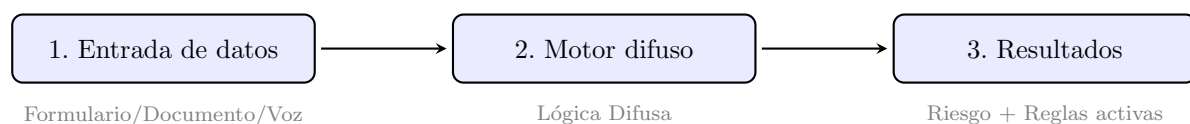


Figura B.1: Flujo de uso de la plataforma C.O.C.O.

B.1.3. Paso 1 — Entrada de datos

Al acceder a `/analyze` se muestra el formulario de análisis clínico. Existen **tres vías** para introducir los datos, que pueden combinarse libremente:

A. Extracción automática mediante documento

1. En la sección *Extracción automática con IA*, hacer clic en el área de carga o arrastrar el archivo.
2. Formatos aceptados: **PDF**, **CSV**, **Excel** (`.xlsx` / `.xls`).
3. El documento se envía al endpoint `POST /api/extraer-documento`, que lo procesa con Gemini y devuelve los campos extraídos.
4. Los campos reconocidos se rellenan automáticamente en el formulario. Los que no se pudieron extraer se marcan con un borde ámbar para indicar que requieren revisión manual.

Campos en ámbar

Un campo con borde ámbar indica que la IA no encontró ese dato en el documento. Revísalo e introdúcelo manualmente antes de continuar.

B. Dictado por voz

1. Hacer clic en *Dictar por voz* (micrófono).
2. Dictar los datos del paciente en lenguaje natural, por ejemplo:
«*Paciente Golden Retriever, ocho años, bebe mucha agua y sufre alopecia, ALP 400 U/L*».
3. La transcripción se envía al endpoint `POST /api/extraer-voz`; Gemini infiere los campos y los vuelca en el formulario de igual forma que con el documento.

C. Introducción manual

El formulario se divide en tres secciones, todas obligatorias salvo los signos clínicos (que por defecto se consideran ausentes):

Datos demográficos

Campo	Unidad / Formato	Ejemplo
Edad	Años decimales (≥ 0)	8 o 7.5
Raza	Texto libre	Golden Retriever
Peso relativo	% respecto a la media de la raza	120 (20 % sobrepeso)

Cuadro B.1: Campos demográficos del formulario.

Datos clínicos

Se presentan como tarjetas con casilla de verificación. Al marcar un signo, la tarjeta cambia de color para indicar que está activo. En la esquina superior derecha se muestra un contador en tiempo real del número de signos seleccionados.

Los signos disponibles para el predictor de HAC son:

Signo	Descripción breve
Polidipsia	Bebe más agua de lo habitual
Abdomen inflamado	Distensión abdominal visible
Alopecia	Pérdida de pelo simétrica
Polifagia	Apetito excesivo constante
Poliuria	Orina en mayor cantidad
Debilidad muscular	Dificultad para levantarse o moverse
Piel fina	Piel frágil o con hematomas
Jadeo	Respiración jadeante sin esfuerzo

Cuadro B.2: Signos clínicos disponibles en el formulario.

Datos de laboratorio

Campo	Unidad	Ejemplo
ALP (Fosfatasa Alcalina)	U/L	350
ALT (Alanina Aminotransferasa)	U/L	80
USG (Gravedad Específica Orina)	adimensional	1.008
Colesterol	mg/dL	320

Cuadro B.3: Campos de laboratorio del formulario.

B.1.4. Paso 2 — Lanzar el análisis

Antes de enviar, el panel lateral derecho (*Listo para analizar*) muestra el estado de completitud de cada sección con un indicador visual: ✓ cuando el bloque está completo y • cuando falta algún campo.

Una vez verificados los datos, hacer clic en **Analizar ahora**. Aparece un modal de progreso con una barra animada mientras el frontend llama al endpoint `POST /predict/cushing` del backend. El análisis tarda habitualmente menos de 2 segundos.

Si algún campo obligatorio (marcado con *) está vacío o no es un número válido, el formulario muestra un mensaje de error y no envía la solicitud.

B.1.5. Paso 3 — Interpretación de resultados

Tras el análisis, la aplicación navega automáticamente a `/results` y presenta dos columnas de información.

Columna izquierda — Nivel de riesgo y recomendaciones

La tarjeta principal (`ResultCard`) muestra:

- **Nivel de riesgo** en forma de etiqueta lingüística (*Muy bajo*, *Bajo*, *Medio*, *Alto*, *Muy alto*) junto con el valor numérico normalizado $\in [0, 1]$.
- **Confianza** de la predicción, que refleja la solidez conjunta de las reglas activadas (véase el apéndice A).
- **Recomendaciones** de seguimiento, presentadas como lista de acciones concretas.

La tabla B.4 resume la correspondencia entre el valor crisp y los niveles de riesgo.

Cuadro B.4: Niveles de riesgo y umbrales del valor crisp.

Nivel	Rango crisp	Color indicativo
Muy bajo	$[0, 0,2)$	Verde esmeralda
Bajo	$[0,2, 0,4)$	Verde azulado
Medio	$[0,4, 0,6)$	Ámbar
Alto	$[0,6, 0,8)$	Naranja
Muy alto	$[0,8, 1]$	Rosa/rojo

Columna derecha — Reglas activadas

Las reglas se presentan ordenadas de mayor a menor activación. Para cada regla activada se muestra:

- **Etiqueta descriptiva** (campo `label` de la regla JSON), que resume el patrón clínico detectado. Ejemplo: «*Tríada clínica principal*».
- **Nivel de consecuente** (*Riesgo alto*, *Riesgo medio*, etc.) con código de color.
- **Barra de activación** que representa gráficamente $\alpha \in [0, 1]$ de la regla.
- **Peso de la regla** (w), que indica su importancia relativa en el sistema.

Esta sección proporciona la **explicabilidad** del resultado: el clínico puede identificar exactamente qué combinación de signos y valores de laboratorio ha contribuido a la estimación de riesgo.

Botón «Cómo interpretar»

El botón *Cómo interpretar* abre un modal con una explicación resumida del significado del puntaje difuso, recordando que el resultado es orientativo y no reemplaza la evaluación profesional.

B.2. Implementación de un predictor para nueva patología

C.O.C.O. está diseñada para crecer. Añadir un predictor para una nueva patología —por ejemplo, *hipotiroidismo* o *diabetes mellitus*— implica:

1. Crear la base de conocimiento JSON en `backend/conocimiento/`.
2. Implementar la clase de predicción Python heredada de `Prediccion`.
3. Registrar el endpoint en `backend/api.py`.
4. Registrar la enfermedad en el frontend (`frontend/src/lib/diseases.ts`).

Los pasos siguientes describen cada uno de ellos con el mismo esquema que usa el predictor de Cushing como referencia.

B.2.1. Paso 1 — Crear la base de conocimiento

La base de conocimiento es un directorio con la siguiente estructura obligatoria:

```

1 backend/conocimiento/
2 \-- <patologia>/          # nombre en minúsculas
3   |-- metadata.json
4   |-- variables/
5   |   |-- demograficas.json
6   |   |-- clinicas.json
7   |   |-- laboratorio.json
8   |   \-- consecuente.json # nombre debe contener "consecuente"
9   \-- reglas/
10      |-- riesgo_bajo.json
11      |-- riesgo_medio.json
12      \-- riesgo_alto.json # tantos archivos como convenga

```

Listing B.1: Estructura de directorios de la base de conocimiento.

metadata.json

Campos obligatorios validados por `Prediccion<Patologia>._validar.metadata()` (tomando como referencia el caso base de `PrediccionCushing`):

```
1 {  
2   "metadata": {  
3     "version": "1.0",  
4     "fecha": "01/06/2026",  
5     "autor": "Nombre Apellido",  
6     "descripcion": "Base de conocimiento difusa para predicción de hipotiroidismo canino"  
7   }  
8 }
```

Listing B.2: Ejemplo de metadata.json.

Variables antecedentes (variables/*.json)

Cada archivo (excepto `consecuente.json`) declara variables numéricas o categóricas. El nombre del archivo no es relevante; el sistema carga todos los `/*.json` del directorio.

Variable numérica

```
1 {  
2   "t4_libre": {  
3     "universo": [0, 5, 0.01],  
4     "terminos": {  
5       "bajo": { "funcion": "zmf", "params": [0.5, 1.5] },  
6       "normal": { "funcion": "trimf", "params": [1.0, 2.5, 4.0] },  
7       "alto": { "funcion": "smf", "params": [3.5, 5.0] }  
8     }  
9   }  
10 }
```

Listing B.3: Ejemplo de variable numérica con función triangular.

Las funciones de pertenencia disponibles se listan en la tabla A.3.

Variable categórica

```
1 {  
2   "raza": {  
3     "tipo": "categorica",  
4     "terminos": {  
5       "predispuesta": ["beagle", "dachshund", "cocker_spaniel", "doberman"],  
6       "otra": ["labrador", "golden_retriever", "pastor_aleman"]  
7     }  
8   }  
9 }
```

Listing B.4: Ejemplo de variable categórica (raza).

Variable binaria (signo clínico) Los signos clínicos se modelan como variables numéricas en $[0, 1]$ con términos **no/si**:

```

1 {
2   "letargia": {
3     "tipo": "binaria",
4     "universo": [0, 1, 0.01],
5     "terminos": {
6       "no": { "funcion": "trimf", "params": [0, 0, 0.4] },
7       "si": { "funcion": "trimf", "params": [0.6, 1, 1] }
8     }
9   }
10 }

```

Listing B.5: Ejemplo de variable binaria para un signo clínico.

Variable consecuente (variables/consecuente.json)

El nombre del archivo debe contener la cadena "consecuente" para que el cargador lo clasifique correctamente. Define la variable de salida del sistema:

```

1 {
2   "riesgo": {
3     "universo": [0, 1, 0.01],
4     "terminos": {
5       "muy_bajo": { "funcion": "zmf", "params": [0.0, 0.2] },
6       "bajo": { "funcion": "trimf", "params": [0.1, 0.3, 0.5] },
7       "medio": { "funcion": "trimf", "params": [0.3, 0.5, 0.7] },
8       "alto": { "funcion": "trimf", "params": [0.5, 0.7, 0.9] },
9       "muy_alto": { "funcion": "smf", "params": [0.8, 1.0] }
10    }
11  }
12 }

```

Listing B.6: Ejemplo de consecuente.json.

Reglas (reglas/*.json)

Cada archivo puede contener una o varias reglas (array o objeto). Campos de cada regla:

```

1 [
2   {
3     "label": "Letargia con T4 bajo: perfil hipotiroideo clásico",
4     "antecedentes": [
5       { "variable": "letargia", "termino": "si" },
6       { "variable": "t4_libre", "termino": "bajo" }
7     ],
8     "conectiva": "AND",
9     "consecuente": { "variable": "riesgo", "termino": "alto" },

```

```

10     "peso": 1.0,
11     "fuente": "Scott-Moncrieff 2015: letargia presente en >80% hipotiroidismo canino"
12 }
13 ]

```

Listing B.7: Ejemplo de regla JSON con dos antecedentes AND.

Campo	Tipo	Obl.	Descripción
antecedentes	array	Sí	Lista de pares <i>variable/termino</i>
consecuente	objeto	Sí	Par <i>variable/termino</i> de salida
conectiva	string	No	"AND" (defecto) u "OR"
peso	número	No	$w \geq 0$; defecto 1,0
label	string	No	Texto descriptivo; aparece en la UI
fuentes	string	No	Referencia bibliográfica

Cuadro B.5: Campos obligatorios y opcionales de una regla JSON.

B.2.2. Paso 2 — Clase de predicción Python

Crea el archivo `backend/sistema/prediccionPatologia.py` heredando de la clase abstracta `Prediccion`:

```

1  #Realizar los imports necesarios
2
3  class PrediccionHipotiroidismo(Prediccion):
4      """Predictor difuso para hipotiroidismo canino."""
5
6      FUNCIONES_FUZZY = {
7          "trimf": TriangularMF,
8          "zmf": ZShapeMF,
9          "smf": SShapeMF,
10     }
11
12     def __init__(self, moduloDemografico=None,
13                  moduloClinico=None,
14                  moduloLaboratorio=None):
15         super().__init__(moduloDemografico, moduloClinico, moduloLaboratorio)
16
17         self.base_conocimiento_path = os.path.join(
18             os.path.dirname(__file__), '..', 'conocimiento', 'hipotiroidismo'
19         )
20         self._base_conocimiento = None
21         self.system = FuzzySystem()
22         self.variables = {}
23
24     # Delegación al mismo motor genérico que PrediccionCushing
25
26     def fuzzificar_datos(self):
27         # Copiar de PrediccionCushing: _cargar_base_conocimiento() +
28         # _crear_variable_fuzzy() para cada antecedente y consecuente
29         ...
30

```

```

31 def implementar_reglas(self):
32     # Copiar de PrediccionCushing: _construir_reglas()
33     ...
34
35 def predecir(self):
36     inputs = {
37         # Mapear los módulos a los nombres de variable declarados en los JSON
38         "t4_libre": self.moduloLaboratorio.obtener_t4_libre(),
39         "letargia": 1.0 if self.moduloClinico.obtener_letargia() else 0.0,
40         "raza": self.moduloDemografico.obtener_raza(),
41         # ... resto de variables
42     }
43     return self.system.infer(inputs, self.variables["riesgo"])

```

Listing B.8: Esqueleto mínimo de un nuevo predictor.

Consejo: reutilizar el motor genérico

Los métodos `_cargar_base_conocimiento()`, `_crear_variable_fuzzy()`, `implementar_reglas()` e incluso la validación de esquema de `PrediccionCushing` son independientes de la patología. Puedes extraerlos a una clase base intermedia `PrediccionJSON` y heredar de ella para evitar duplicación de código.

También deberás añadir al módulo correspondiente (por ejemplo `ModuloLaboratorio`) los atributos y getters necesarios para las nuevas variables de laboratorio.

B.2.3. Paso 3 — Registrar el endpoint en la API

En `backend/api.py`, añade el modelo de entrada Pydantic y el endpoint:

```

1 #Realizar los imports necesarios
2
3 class HipotiroidismoInput(BaseModel):
4     edad: float
5     raza: str
6     t4_libre: float
7     letargia: bool = False
8     # ... resto de campos
9
10 @app.post("/predict/hipotiroidismo")
11 def predict_hipotiroidismo(payload: HipotiroidismoInput):
12     modulo_demografico = ModuloDemografico(edad=payload.edad, raza=payload.raza, peso_re=
13     modulo_clinico = ModuloClinico(letargia=payload.letargia)
14     modulo_laboratorio = ModuloLaboratorio(t4_libre=payload.t4_libre)
15
16     predictor = PrediccionHipotiroidismo(
17         moduloDemografico=modulo_demografico,
18         moduloClinico=modulo_clinico,
19         moduloLaboratorio=modulo_laboratorio,
20     )
21     predictor.fuzzificar_datos()
22     predictor.implementar_reglas()

```

```
23  
24     result = predictor.predecir()  
25     return jsonable_encoder(serialize_result(result))
```

Listing B.9: Nuevo endpoint FastAPI para hipotiroidismo.

B.2.4. Paso 4 — Registrar la enfermedad en el frontend

La lista de enfermedades disponibles en el selector del formulario se gestiona en `frontend/src/lib/diseases.ts` (o el archivo equivalente `diseases.ts`):

```
1 export const diseases = [  
2   {  
3     id: "cushing",  
4     name: "Hiperadrenocorticismo (Cushing)",  
5     shortName: "Cushing canino",  
6     endpoint: "/predict/cushing",  
7   },  
8   //      Añadir aquí la nueva patología  
9   {  
10    id: "hipotiroidismo",  
11    name: "Hipotiroidismo canino",  
12    shortName: "Hipotiroidismo",  
13    endpoint: "/predict/hipotiroidismo",  
14  },  
15 ];
```

Listing B.10: Registro de una nueva enfermedad en el frontend.

La función `analyzeCushing()` del cliente API deberá generalizarse (o duplicarse) para que use el `endpoint` dinámicamente según el `diseaseId` seleccionado.

B.2.5. Resumen del proceso de extensión

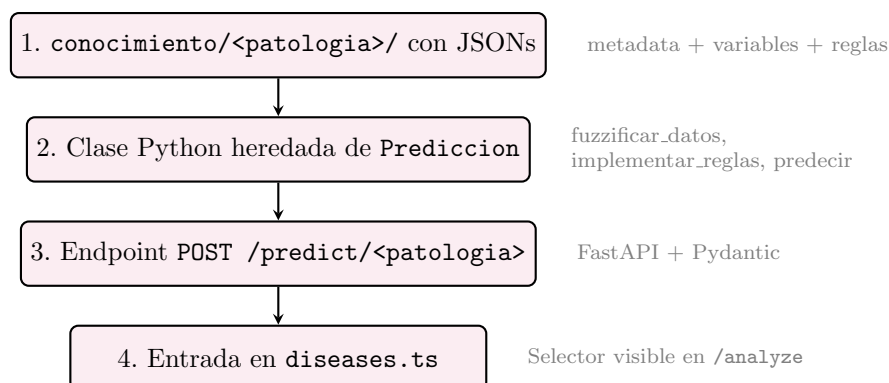


Figura B.2: Pasos para incorporar un predictor de nueva patología a C.O.C.O.

La lógica difusa (variables, funciones de pertenencia, reglas, motor de inferencia y defuzzificación) es completamente reutilizable: el desarrollador solo aporta el **conocimiento clínico** en forma de JSONs y el **mapeo de entradas** en la clase de predicción. El resto del pipeline —incluyendo la confianza compuesta y la explicabilidad de reglas— funciona de forma automática para cualquier nueva patología.

Glosario

ACTH

Hormona que da la orden a las glándulas suprarrenales para generar cortisol.

Adenoma hipofisario funcional

Producción exagerada de una hormona.

Adrenocortical

Referido a la capa externa de la glándula suprarrenal, que produce hormonas esenciales como cortisol, aldosterona y andrógenos.

Alopecia (endocrina)

Pérdida de pelo de distribución simétrica y origen hormonal, sin lesión primaria en la piel. Es un signo endocrino típico del HAC, presente en el 29 % de los casos, y uno de los predictores positivos más consistentes del sistema.

ALT/AST

Alanina aminotransferasa (ALT) y aspartato aminotransferasa (AST). Enzimas hepáticas cuya elevación es frecuente en el HAC, aunque inespecífica, ya que también se observa en otras enfermedades sistémicas. En el sistema actúan como variables contextuales complementarias, sin papel decisivo en el diagnóstico.

Cortisol

Hormona producida por la capa externa de las glándulas suprarrenales bajo la orden de la ACTH. Regula el metabolismo de carbohidratos, proteínas y grasas, la respuesta inmunitaria y la reacción al estrés. Su exceso prolongado es la causa directa de los signos clínicos y las alteraciones analíticas del hiperadrenocorticismos, entre ellos la polidipsia, la polifagia, la alopecia endocrina y la elevación de la ALP.

Densidad urinaria

Véase USG.

Diabetes mellitus canina

Enfermedad metabólica crónica producida por un déficit de insulina o por resistencia a ella, que provoca hiperglucemia. Es la enfermedad con mayor similitud sintomática con el HAC, lo que dificulta el diagnóstico diferencial. Además, un porcentaje significativo de perros con HAC desarrolla diabetes mellitus de forma secundaria debido al exceso prolongado de cortisol.

Ecografía abdominal

Técnica de imagen que utiliza ultrasonidos para visualizar órganos internos. En el diagnóstico del HAC se emplea para evaluar el tamaño y la morfología de las glándulas suprarrenales, y facilita la diferenciación entre el hiperadrenocorticismismo dependiente de la hipófisis y el de origen adrenal.

Endocrinopatía

Enfermedad causada por un mal funcionamiento del sistema endocrino, es decir, del conjunto de glándulas que producen y secretan hormonas. El hiperadrenocorticismismo, el hipotiroidismo y la diabetes mellitus son ejemplos de endocrinopatías frecuentes en perros.

Fosfatasa Alcalina (ALP)

Enzima presente en el hígado, los huesos y otros tejidos cuya elevación en sangre es uno de los hallazgos analíticos más frecuentes en el HAC. En el sistema C.O.C.O. constituye la variable central del módulo de laboratorio, dado que una elevación marcada es altamente sugestiva de hiperadrenocorticismismo..

Glándulas suprarrenales**Hepatomegalia**

Aumento del tamaño del hígado. En el contexto del HAC suele asociarse a distensión abdominal visible y constituye uno de los signos con mayor peso predictivo individual en el modelo, presente en casi el 50 % de los casos.

Hiperadrenocorticismismo (HAC)

Término general, sinónimo de síndrome de Cushing, que engloba los cuadros clínicos derivados de una producción excesiva de hormonas

adrenocorticales, principalmente cortisol. En perros, la forma más frecuente es la dependiente de hipófisis, originada por una secreción excesiva de ACTH secundaria a, normalmente, una hiperplasia. Sus manifestaciones clínicas son poco específicas y se superponen con las de otras enfermedades, lo que dificulta el diagnóstico y convierte esta patología en el objeto central de este trabajo.

Hiperplasia

Multiplicación excesiva de células en un órgano o en un tejido.

Hipófisis

Órgano interno situado en la base del cráneo. Las hormonas que produce influyen en el crecimiento, en el desarrollo sexual, etc.

LDDST

Low Dose Dexamethasone Suppression Test, o test de supresión con dexametasona a baja dosis. Prueba hormonal dinámica considerada la de cribado de elección para el diagnóstico del HAC debido a su elevada sensibilidad. Consiste en administrar una dosis baja de dexametasona y medir la respuesta del cortisol; en perros sanos, el cortisol se suprime, mientras que en los afectados por HAC dicha supresión no se produce.

Odds Ratio (OR)

Medida estadística que expresa la razón entre la probabilidad de que un evento ocurra y la probabilidad de que no ocurra, comparando dos grupos. En el contexto de este trabajo, las OR ajustadas cuantifican la asociación de cada variable demográfica o clínica con el diagnóstico de HAC. Un valor $OR > 1$ indica mayor riesgo respecto al grupo de referencia; $OR < 1$, efecto protector. Se obtiene como $OR = e^{\beta}$, donde β es el coeficiente del modelo de regresión logística.

Piometra

Infección interna del útero.

Polidipsia

Ingesta de agua en cantidad anormalmente elevada. Es el signo clínico más frecuente en el HAC, presente en el 70 % de los casos, y constituye el predictor positivo individual más fuerte del sistema. Suele presentarse junto con poliuria.

Polifagia

Apetito excesivo y persistente. Presente en el 24 % de los casos de HAC, se asocia al exceso de cortisol y puede ir acompañada de

obesidad. Constituye una variable complementaria en el modelo, con mayor relevancia cuando coincide con alteraciones metabólicas como la hiperglucemia o la hipercolesterolemia.

Poliuria

Producción de orina en volumen anormalmente elevado. Aparece en el 58 % de los casos de HAC y está estrechamente asociada a la polidipsia, con la que presenta colinealidad estadística. La densidad urinaria baja ($USG \leq 1.020$) actúa como evidencia analítica de este signo.

Prueba de cribado

Prueba diagnóstica aplicada a una población con sospecha clínica para identificar los casos que requieren confirmación posterior. Prioriza la sensibilidad sobre la especificidad, es decir, busca no perder casos positivos aunque se generen algunos falsos positivos. En el HAC, el LDDST es la prueba de cribado de elección.

Transaminasas séricas

Enzimas que se encuentran en el hígado, el corazón y otros tejidos. Reaccionan ante daño en los tejidos.

USG

Urine Specific Gravity, o gravedad específica de la orina. Mide la concentración de solutos en orina y refleja la capacidad del riñón para diluir o concentrar. Valores bajos (orina diluida, ≤ 1.020) son coherentes con la poliuria/polidipsia asociada al HAC. Actúa como variable de coherencia entre la clínica del paciente y la evidencia analítica.

