MÁSTER UNIVERSITARIO EN BIG DATA

TRABAJO FIN DE MÁSTER

SISTEMA MULTI AGENTE PARA EL ENTRENAMIENTO Y OPTIMIZACIÓN DE MODELOS DE MACHINE LEARNING BASADO EN LOS PROTOCOLOS MCP Y A2A

Autor: Daniel Valverde Gómez

Director: David Contreras Bárcena

Madrid

Mayo de 2026

Declaro, bajo mi responsabilidad, que el Proyecto presentado con el título

Sistema multi-agente para el entrenamiento y optimización de modelos de machine learning basado en los protocolos MCP y A2A

en la ETS de Ingeniería - ICAI de la Universidad Pontificia Comillas en el

curso académico 2025/26 es de mi autoría, original e inédito y

no ha sido presentado con anterioridad a otros efectos.

El Proyecto no es plagio de otro, ni total ni parcialmente y la información que ha sido

tomada de otros documentos está debidamente referenciada.

Fdo.: Daniel Valverde Gómez Fecha: 17/ 05/ 2026

Autorizada la entrega del proyecto

EL DIRECTOR DEL PROYECTO

Fdo.: David Contreras Bárcena Fecha: 08/ 06/ 2026

Uso de Inteligencia Artificial[1]

Declaro bajo mi responsabilidad que (indicar la opción correcta):

☐ No he utilizado Inteligencia Artificial en la elaboración del presente documento.

X He utilizado Inteligencia Artificial en la elaboración del presente documento y/o del Anexo B siempre en las condiciones permitidas por la Universidad Pontificia Comillas, es decir, aplicando el Nivel 2 de la Escala de Evaluación de Perkins et al. (2024): "*La IA puede utilizarse para actividades previas a la tarea, como la lluvia de ideas, la descripción y la investigación inicial. Este nivel se centra en el uso de la IA para la planificación, las síntesis y la generación de ideas, pero las evaluaciones deben hacer hincapié en la capacidad de desarrollar y refinar estas ideas de forma independiente*". En concreto, las Inteligencia Artificial ha sido empleada para:

El apoyo en tareas de investigación y análisis. En concreto, la temática del proyecto, los objetivos principales y las herramientas a utilizar (MCP y A2A) fueron definidas por mí, mientras que la IA se empleó para realizar una búsqueda más eficiente del estado del arte. Asimismo, se utilizó como apoyo para obtener sugerencias y alternativas relacionadas con la propuesta de arquitectura planteada inicialmente, con el objetivo de refinar la solución diseñada. En todo momento la toma de decisiones finales estuvo bajo mi responsabilidad.

[1] Esta declaración se refiere al uso de la Inteligencia Artificial generativa para realizar los documentos del Proyecto (Anexo B y Memoria). No aplica a Proyectos donde, por su naturaleza, deban emplear inteligencia artificial como parte de los mismos (aplicación de técnicas de aprendizaje automático, redes neuronales, análisis de datos…)


Firmado (alumno): Daniel Valverde Gómez
Fecha: 30/05/2026

Autorización para la entrega del Proyecto

El Director del Proyecto	El co-Director del Proyecto (si aplica)
	
Fdo: David Contreras Bárcena	Fdo:
Fecha: 08/06/2026	Fecha:

MÁSTER UNIVERSITARIO EN BIG DATA

TRABAJO FIN DE MÁSTER

SISTEMA MULTI AGENTE PARA EL ENTRENAMIENTO Y OPTIMIZACIÓN DE MODELOS DE MACHINE LEARNING BASADO EN LOS PROTOCOLOS MCP Y A2A

Autor: Daniel Valverde Gómez

Director: David Contreras Bárcena

Madrid

Agradecimientos

Me gustaría agradecer a mi directo David Contreras Bárcena por sus consejos y apoyo durante la realización del trabajo, tanto para guiar la toma de decisiones como para indicar puntos de mejora.

Adicionalmente, quiero expresar mi gratitud a todas las personas que contribuyeron a este proyecto y cuyo nombre no se menciona aquí, pero su ayuda fue igualmente valiosa.

SISTEMA MULTI AGENTE PARA EL ENTRENAMIENTO Y OPTIMIZACIÓN DE MODELOS DE MACHINE LEARNING BASADO EN LOS PROTOCOLOS MCP Y A2A

Autor: Valverde Gómez, Daniel
Director: Contreras Bárcena, David
Entidad Colaboradora: ICAI – Universidad Pontificia Comillas

RESUMEN DEL PROYECTO

Palabras clave: AutoML, Model Context Protocol, scikit-learn, XGBoost, pipeline ML, LLM, AI Agents, evaluación de modelos.

La automatización del ciclo de vida del aprendizaje automático es un problema de gran relevancia práctica. Los enfoques tradicionales de *AutoML* abordan la selección de modelos y la optimización de hiperparámetros de forma algorítmica, pero no incorporan capacidad de razonamiento explícito sobre el proceso ni flexibilidad para adaptarse a requerimientos expresados en lenguaje natural. En este proyecto se propone una alternativa: agentes basados en LLM que razonan sobre qué operaciones ML ejecutar y delegan la ejecución en servidores MCP especializados.

La integración de LLMs en pipelines de datos ha sido explorada en trabajos recientes como DS-Agent [1], que utiliza un agente para generar y ejecutar código de análisis, o AutoML-Agent [2] que propone una arquitectura multi-agente con recuperación de plantillas (RAP) para consultar flujos exitosos pasados. El presente trabajo se diferencia de estos enfoques al adoptar una arquitectura multi-agente con separación explícita de responsabilidades mediante los protocolos *Agent-to-Agent* (A2A) [4], y *Model Context Protocol* (MCP) [5], lo que aporta trazabilidad y modularidad al proceso, además de estandarización.

Este trabajo constituye la segunda parte de una serie de dos trabajos de fin de máster. La primera parte estableció la infraestructura de comunicación de un sistema multi-agente basado en los protocolos A2A y MCP. El presente trabajo profundiza en la capa de ejecución del sistema: los tres servidores MCP que implementan el pipeline de aprendizaje automático (ML) y el contenido ML de las herramientas que exponen. Se pondrá foco en la implementación de dichos servidores MCP y en la evaluación de calidad del pipeline ML resultante.

El objetivo es demostrar que un sistema multi-agente basado en LLM puede automatizar de forma efectiva un pipeline completo de ML clásico, desde la ingesta y preprocesamiento de datos hasta la evaluación e interpretación del modelo, produciendo resultados comparables a los obtenidos mediante un proceso automatizado con código. A continuación, se muestra un diagrama completo de la arquitectura propuesta señalando el enfoque de este proyecto:

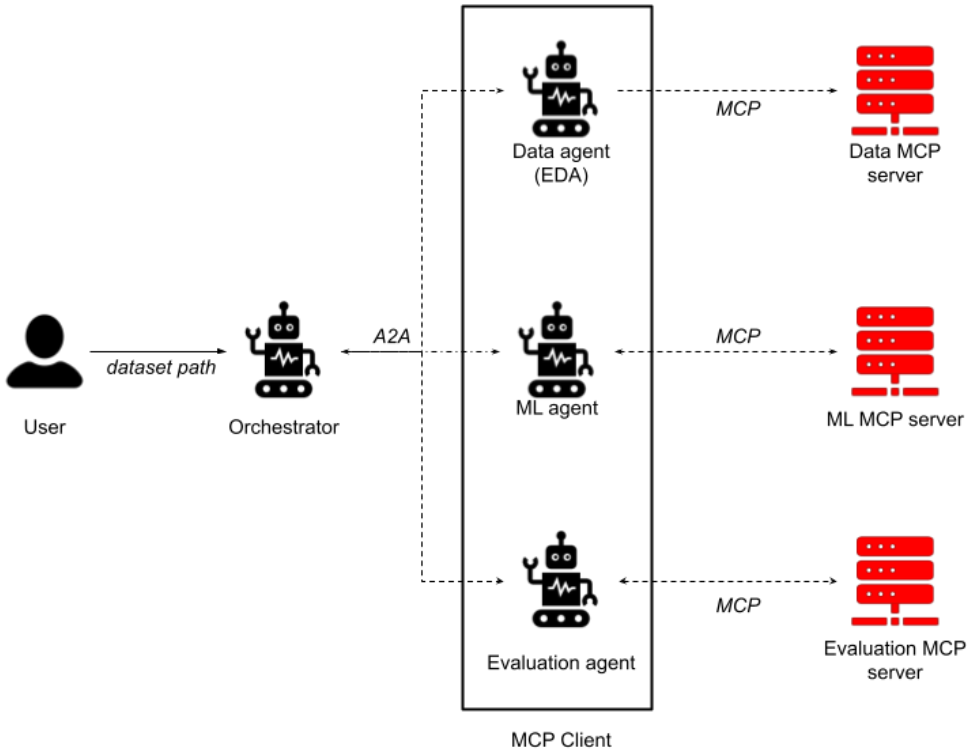

Figura 1 - Diagrama de la arquitectura propuesta. Se ha resaltado la parte en la que se hará foco en este proyecto (los servidores MCP). Fuente: elaboración propia

El *Data MCP Server* es el componente encargado de la carga, exploración, diagnóstico y preprocesamiento de *datasets* dentro del pipeline de aprendizaje automático. El servidor expone un conjunto de herramientas para realizar análisis exploratorio, detección de problemas en los datos, transformación de variables y partición en conjuntos de entrenamiento y prueba. Las operaciones de manipulación y preprocesado se implementan mediante *pandas* y *scikit-learn*, incluyendo imputación de valores faltantes, codificación categórica, normalización y selección de características.

Cada herramienta recibe parámetros serializados en JSON y devuelve resultados estructurados con metadatos relevantes para las etapas posteriores del pipeline. Sobre estos resultados, el *Data Agent* decide dinámicamente qué operaciones ejecutar y en qué secuencia, razonando explícitamente sobre las características del *dataset* antes de coordinarse con el orquestador del sistema.

El *ML MCP Server* es el componente central del pipeline desde la perspectiva del aprendizaje automático. Expone herramientas para la selección de algoritmos, búsqueda de hiperparámetros y registro de experimentos, soportando modelos de clasificación y regresión basados en *scikit-learn* como *Logistic Regression, Random Forest* o *Gradient Boosting*. A partir de los datos generados por el *Data MCP Server*, el *ML Agent* selecciona los algoritmos adecuados utilizando su propio razonamiento en lugar de reglas heurísticas fijas, diferenciándose así de los enfoques AutoML tradicionales. La optimización de hiperparámetros se implementa mediante *RandomizedSearchCV* con espacios de búsqueda configurables según el algoritmo

seleccionado. La herramienta de entrenamiento recibe el algoritmo seleccionado, los hiperparámetros óptimos y el *dataset* preprocesado, y devuelve el modelo serializado junto con las métricas de validación cruzada obtenidas durante el entrenamiento.

El *Eval MCP Server* es el componente responsable del cierre del pipeline: evalúa el modelo entrenado sobre el conjunto de *test*, lo compara con *runs* anteriores y genera el artefacto final que llega al usuario. Sus herramientas calculan métricas de rendimiento estándar adaptadas al tipo de tarea (accuracy, F1-score macro y weighted, y AUC-ROC para clasificación; RMSE, MAE, R^2 y MAPE para regresión). Se mantiene un registro histórico de experimentos que permite a la herramienta *compare_models* seleccionar el mejor modelo disponible entre todas las ejecuciones pasadas sobre el mismo *dataset*, no solo entre los modelos del run actual.

El *Evaluation Agent* utiliza el razonamiento del LLM para interpretar los resultados numéricos y traducirlos en diagnóstico accionable. A partir de las métricas devueltas por el servidor, el agente analiza si el rendimiento es adecuado para el tipo de problema, identifica señales de sobreajuste comparando las métricas de *train* y *test*, evalúa el comportamiento por clase en problemas de clasificación desbalanceada y genera un reporte final en Markdown con el razonamiento completo del pipeline. Este razonamiento, documentado en el campo agent_notes del reporte, constituye el principal elemento diferencial del sistema respecto a los *AutoML* clásicos.

La validación del sistema se realizó utilizando cuatro conjuntos de datos de referencia ampliamente empleados en aprendizaje automático: *Iris* [6] para clasificación multiclase básica, *Titanic* [7] para una tarea de clasificación binaria más realista, *WineQT (Wine Quality)* [8] para un problema de clasificación más desafiante debido al desbalance entre clases, y *Housing Prices* [9] para regresión tabular.

Estos *datasets* fueron seleccionados por su diversidad en términos de tipo de problema, tamaño y naturaleza de las variables, lo que permite evaluar la capacidad de generalización del sistema.

En cada ejecución, el usuario proporciona únicamente la ruta del *dataset* y el agente orquestador descompone la entrada en subtareas y orquesta la ejecución secuencial de los tres agentes especializados. La trazabilidad completa del proceso, incluyendo las herramientas MCP invocadas, los parámetros utilizados y el razonamiento del LLM en cada decisión, queda registrada en los logs del sistema y en la respuesta final. Esto permite una auditoria detallada del pipeline.

MULTI-AGENT SYSTEM FOR THE TRAINING AND OPTIMIZATION OF MACHINE LEARNING MODELS BASED ON MCP AND A2A PROTOCOLS.

Author: Valverde Gómez, Daniel
Supervisor: Contreras Bárcena, David
Collaborating Entity: ICAI – Universidad Pontificia Comillas

ABSTRACT

Keywords: AutoML, Model Context Protocol, scikit-learn, XGBoost, ML pipeline, LLM, AI Agents, model evaluation.

The automation of the machine learning lifecycle remains a problem of considerable practical relevance. Traditional AutoML approaches address model selection and hyperparameter optimization algorithmically; however, they do not incorporate explicit reasoning capabilities regarding the process itself, nor the flexibility to adapt to requirements expressed in natural language. This project proposes an alternative approach: LLM-based agents that reason about which ML operations should be executed and delegate their execution to specialized MCP servers.

The integration of LLMs into data pipelines has been explored in recent works such as *DS-Agent* [1], which employs an agent to generate and execute analysis code, and *AutoML-Agent* [2], which proposes a multi-agent architecture with Retrieval-Augmented Planning (RAP) to query previously successful workflows. The present work differs from these approaches by adopting a multi-agent architecture with an explicit separation of responsibilities through the A2A [4] protocol and the MCP protocol [5]. This design provides improved traceability, modularity, and standardization throughout the process.

This work constitutes the second part of a two-part master's thesis series. The first part established the communication infrastructure of a multi-agent system based on the A2A and MCP protocols. The present work focuses on the execution layer of the system: the three MCP servers implementing the machine learning (ML) pipeline and the ML functionality exposed through their tools. Emphasis is placed on the implementation of these MCP servers and on the quality evaluation of the resulting ML pipeline.

The objective is to demonstrate that an LLM-based multi-agent system can effectively automate a complete classical ML pipeline, from data ingestion and preprocessing to model evaluation and interpretation, producing results comparable to those obtained through a conventional code-based automated process. The following diagram presents the complete architecture of the proposed system, highlighting the scope of this project:

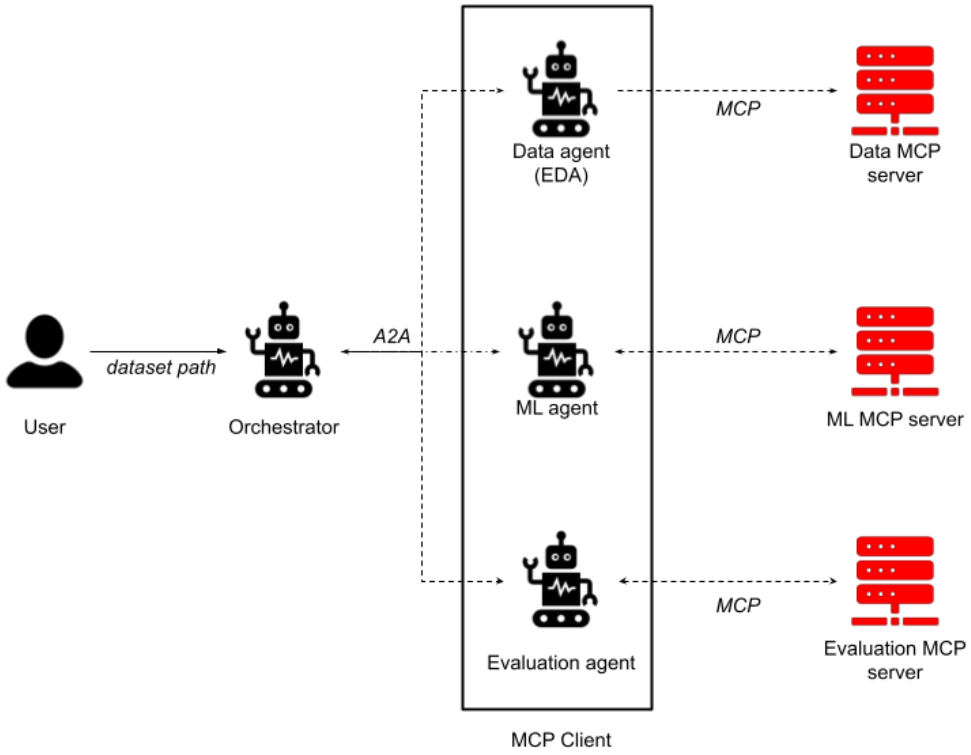

Figura 2 - Diagram of the proposed architecture. The section on wich this project will focus (the MCP servers) has been highlighted. Source: own elaboration.

The MCP Data Server is the component responsible for loading, exploring, diagnosing, and preprocessing datasets within the machine learning pipeline. The server exposes a set of tools for exploratory data analysis, data quality assessment, feature transformation, and splitting into training and test sets. Data manipulation and preprocessing operations are implemented using pandas and scikit-learn, including missing value imputation, categorical encoding, normalization, and feature selection.

Each tool receives parameters serialized in JSON format and returns structured outputs enriched with metadata relevant to subsequent stages of the pipeline. Based on these outputs, the Data Agent dynamically determines which operations to execute and in what sequence, explicitly reasoning about dataset characteristics before coordinating with the system orchestrator.

The MCP ML Server is the central component of the pipeline from a machine learning perspective. It exposes tools for algorithm selection, hyperparameter search, and experiment tracking, supporting classification and regression models based on scikit-learn, such as Logistic Regression, Random Forest, and Gradient Boosting. Using the data provided by the Data Server, the ML Agent selects appropriate algorithms based on its own reasoning rather than fixed heuristic rules, thereby differentiating itself from traditional AutoML approaches. Hyperparameter optimization is implemented via RandomizedSearchCV, with search spaces configured according to the selected algorithm. The training tool receives the chosen

algorithm, optimal hyperparameters, and the preprocessed dataset, and returns the serialized model along with cross-validation metrics obtained during training.

The Eval MCP Server is the component responsible for the pipeline's final stage: it evaluates the trained model on the test set, compares it with previous runs, and generates the final artifact delivered to the user. Its tools compute standard performance metrics adapted to the task type (accuracy, macro and weighted F1-score, and AUC-ROC for classification; RMSE, MAE, R^2, and MAPE for regression). The system keeps a historical experiment log that allows compare_models tool to select the best available model across all past runs on the same dataset, not only among models from the current run.

The Evaluation Agent uses LLM reasoning to interpret numerical results and translate them into actionable diagnostics. Based on the metrics returned by the server, the agent assesses whether performance is adequate for the problem type, identifies signs of overfitting by comparing training and test metrics, evaluates per-class behavior in imbalanced classification problems, and generates a final Markdown report with the full pipeline reasoning. This reasoning, documented in the agent_notes field of the report, is the main distinguishing feature of the system compared to classical AutoML approaches.

The system was validated using four widely used benchmark datasets: *Iris* [6] for basic multiclass classification, *Titanic* [7] for a more realistic binary classification task, *WineQT (Wine Quality)* [8] for a more challenging classification problem due to class imbalance, and *Housing Prices* [9] for tabular regression. These datasets were selected due to their diversity in terms of problem type, dataset size, and feature characteristics, enabling a comprehensive evaluation of the system's generalization capabilities.

In each execution, the user provides only the dataset path, and the orchestrator agent decomposes the input into subtasks and coordinates the sequential execution of the three specialized agents. Full process traceability, including invoked MCP tools, parameter configurations, and LLM reasoning at each decision step recorded in the system logs and in the final response. This enables detailed auditability of the entire pipeline.

Índice de la memoria

Índice de figuras

Índice de tablas

Capítulo 1. INTRODUCCIÓN

El aprendizaje automático ha pasado de ser una disciplina académica a convertirse en uno de los motores fundamentales de la transformación digital. Sin embargo, la brecha entre el potencial de estas técnicas y su aplicación real en las organizaciones sigue siendo significativa. Construir un pipeline de ML de calidad requiere un ciclo largo y costoso de decisiones expertas (selección de algoritmos, preprocesamiento de datos, optimización de hiperparámetros, evaluación de modelos) que pocas organizaciones pueden sostener de forma sistemática. Automatizar este ciclo con rigor y transparencia es uno de los retos más relevantes del estado actual de la disciplina.

Este trabajo constituye la segunda parte de una serie de dos trabajos en los que se aborda la arquitectura completa. La primera parte de esta serie abordó una dimensión fundamental del proyecto: la infraestructura de comunicación que permite a múltiples agentes de inteligencia artificial colaborar de forma autónoma para ejecutar un pipeline de ML completo. En él se diseñó e implementó un sistema multi-agente compuesto por cuatro agentes especializados (*Orchestrator Agent, Data Agent, ML Agent y Evaluation Agent*) cuya coordinación se articula mediante el protocolo *Agent-to-Agent* (A2A) [4] y cuyo acceso a herramientas se gestiona mediante el *Model Context Protocol* (MCP) [5]. Ese trabajo demostró que los protocolos MCP y A2A pueden combinarse de forma nativa para construir un sistema de orquestación modular, auditable y extensible, y estableció la infraestructura sobre la que se construye el presente trabajo.

Este proyecto abre la capa de ejecución del sistema, concretamente los tres servidores MCP que implementan el *pipeline* de ML y las herramientas que exponen. Si la primera parte respondía a la pregunta de cómo los agentes se comunican, este responde a la pregunta de qué ejecutan cuando lo hacen. El *Data MCP Server* implementa el ciclo completo de preparación de datos: análisis exploratorio, detección de problemas, preprocesamiento y partición del dataset. El *ML MCP Server* implementa la selección de algoritmos, la optimización de hiperparámetros y el entrenamiento de modelos clásicos sobre los datos preparados. El *Evaluation MCP Server* implementa el cálculo de métricas, la comparación entre modelos y la generación del informe final.

Los sistemas *AutoML* tradicionales (*Auto-sklearn* [10], *TPOT* [11], *H2O AutoML [12]*, *AutoGluon* [13]) han demostrado que es posible automatizar la selección de modelos y la optimización de hiperparámetros con resultados competitivos. Sin embargo, operan como cajas negras: no justifican sus decisiones en lenguaje comprensible, no pueden adaptar su estrategia a requisitos expresados en lenguaje natural y no generan recomendaciones de mejora interpretables por un experto humano. El sistema de este trabajo propone un enfoque alternativo: agentes LLM que razonan explícitamente sobre los datos, seleccionan algoritmos con justificación argumentada y producen reportes que documentan cada decisión del pipeline en lenguaje natural. La comparación entre ambos enfoques (*AutoML* clásico y *AutoML* basado en agentes LLM) en términos de calidad de modelos, cobertura de tipos de problema y capacidad de adaptación contextual es el núcleo experimental de este trabajo.

La relevancia de este trabajo se extiende más allá del resultado técnico inmediato. Desde una perspectiva de democratización del ML, un sistema capaz de automatizar el pipeline completo con razonamiento explícito y resultados comprensibles reduce la dependencia de perfiles altamente especializados para tareas de análisis estándar, acercando las capacidades del ML a equipos sin formación estadística avanzada. Desde una perspectiva científica, el trabajo contribuye al área emergente de *AutoML* basado en agentes LLM, que ha cobrado impulso significativo en los últimos dos años con trabajos como *DS-Agent* [1], *AutoML-Agent* [2] *y KompeteAI* [3], pero que carece aún de evaluaciones sistemáticas sobre *datasets* estándar con comparativas directas frente a sistemas *AutoML* clásicos.

Este trabajo busca, en primer lugar, la implementación completa de los tres servidores MCP del sistema: *Data MCP Server, ML MCP Server y Evaluation MCP Server*, con un catálogo de herramientas que cubre preprocesamiento, entrenamiento con cinco algoritmos (*logistic regression, linear regression, random forest* y *XGBoost*) y evaluación con métricas estándar para clasificación y regresión. En segundo lugar**,** el diseño y evaluación de los *system prompts* de los agentes especializados, analizando cómo el razonamiento del LLM se traduce en decisiones de preprocesamiento y selección de modelos coherentes con el contexto de cada *dataset*. En tercer lugar**,** una evaluación comparativa del pipeline automatizado frente a *baselines* de *AutoML* clásico sobre *datasets* de *benchmark* estándar, analizando la calidad de los modelos producidos, la capacidad de adaptación a distintos tipos de problema y las limitaciones del enfoque basado en agentes. En la Figura 3 se muestra un diagrama simplificado de la arquitectura del sistema

desarrollado. El foco de este trabajo se pone en los servidores MCP cuyo funcionamiento es el mismo que una arquitectura cliente-servidor, dónde los clientes serán los distintos agentes del sistema y los servidores los elementos donde se encuentran cada una de las herramientas que permiten a los agentes llevar a cabo las distintas fases del *pipeline AutoML*.

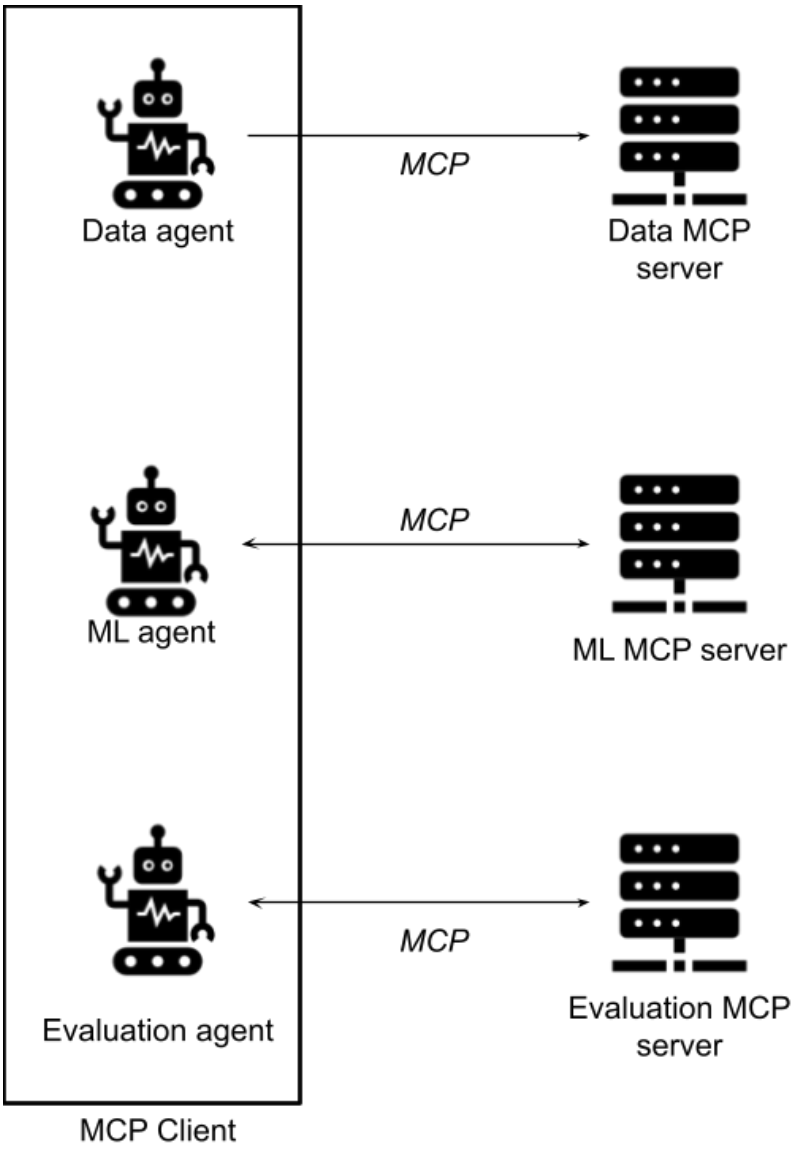

Figura 3 - Diagrama con los actores que interactúan en el flujo de conexión MCP. Los agentes hacen de clientes y los servidores MCP de repositorio con las herramientas para implementar el pipeline AutoML. Fuente: elaboración propia.

Capítulo 2. DESCRIPCIÓN DE LAS TECNOLOGÍAS

Este capítulo describe las tecnologías sobre las que se construye la capa de ejecución del sistema: los servidores MCP que implementan el pipeline de ML. Aquí el énfasis recae sobre las librerías de ML (scikit-learn, XGBoost y statsmodels) y sobre el concepto de *AutoML* como marco de referencia para entender el problema que el sistema trata de resolver. Las tecnologías de infraestructura compartidas con la primera parte de esta serie (MCP, A2A, *FastAPI*) se describen de forma concisa, enfatizando únicamente su rol en el contexto de los servidores MCP.

2.1 AUTOMATED MACHINE LEARNING *(AUTOML)*

El *Machine Learning* automatizado (*AutoML*) es el campo de investigación y desarrollo que busca automatizar las fases más costosas del ciclo de vida del aprendizaje automático: la selección del algoritmo más adecuado para un problema dado, la optimización de sus hiperparámetros y la construcción del pipeline de preprocesamiento [14]. El objetivo último del *AutoML* es reducir la intervención experta necesaria para obtener modelos de ML de calidad, haciendo estas capacidades accesibles a usuarios sin formación avanzada en estadística o ciencia de datos.

2.1.1 COMPONENETES DEL PIPELINE AUTOML

Un *pipeline* de ML automatizado abarca típicamente cuatro fases secuenciales que el sistema de este trabajo implementa de forma distribuida entre sus servidores MCP:

- **Preparación de datos:** Análisis exploratorio del *dataset*, detección y tratamiento de valores faltantes, codificación de variables categóricas, detección de *outliers* y partición en conjuntos de entrenamiento y prueba. Implementado en el MCP Data Server.

- **Selección de algoritmo:** Decisión sobre qué familia de modelos es más adecuada para el tipo de problema (clasificación o regresión), el tamaño del dataset y sus características estadísticas. En el sistema de este trabajo, esta decisión la toma el ML Agent mediante razonamiento LLM.

- **Optimización de hiperparámetros (HPO):** Búsqueda de la configuración de parámetros del modelo que maximiza una métrica de validación. El *ML MCP Server* implementa *RandomizedSearchCV* [15] con espacios de búsqueda predefinidos por algoritmo.

- **Evaluación y selección del mejor modelo:** Cálculo de métricas sobre el conjunto de *test*, comparación entre *runs* de entrenamiento y selección del modelo óptimo para producción. Implementado en el *Evaluation MCP Server*.

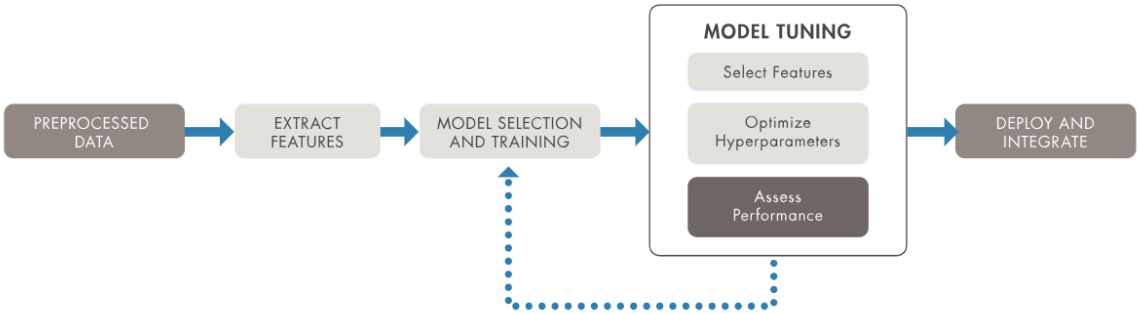

Figura 4 - Diagrama de bloques que detalla el ciclo de vida del aprendizaje automático. Fuente: [16]

2.1.2 AUTOML CLÁSICO VS AUTOML BASADO EN AGENTES LLM

Los sistemas *AutoML* clásicos más representativos (*Auto-sklearn* [10], *TPOT* [11], *H2O AutoML* [12] y *AutoGluon* [13]) automatizan la selección de algoritmos y la HPO mediante búsqueda algorítmica: optimización bayesiana, programación genética o meta-aprendizaje. Estos enfoques han demostrado resultados sólidos en *benchmarks* estándar, pero presentan tres limitaciones estructurales relevantes para este trabajo.

En primer lugar, operan sin razonamiento explícito: seleccionan el mejor modelo según una métrica, pero no justifican esa elección en lenguaje comprensible. En segundo lugar, no pueden adaptarse a requisitos contextuales expresados en lenguaje natural: si el usuario necesita un modelo interpretable o si el *dataset* tiene características particulares que aconsejan una estrategia específica, el sistema no puede incorporar esa información. En tercer lugar, no generan recomendaciones de mejora orientadas al usuario: cuando el rendimiento es bajo, el sistema *AutoML* clásico prueba más configuraciones, pero no explica por qué el rendimiento es bajo ni qué podría hacerse para mejorarlo.

2.2 SCIKIT-LEARN

scikit-learn es la librería de referencia para el aprendizaje automático clásico en Python, publicada orginalmente en 2007 y mantenida activamente por una comunidad open-source [17]. Su adopción es universal en el ecosistema Python de ciencia de datos, lo que la convierte en la elección natural para el *ML MCP Server*: cualquier investigador o ingeniero que trabaje con el sistema encontrará los algoritmos expuestos bajo nombres y APIs que ya conoce.

2.2.1 API UNIFICADA

El diseño más relevante de *scikit-learn* para el presente trabajo es su API unificada basada en el patrón *estimator* [18]: todos los algoritmos implementan la misma interfaz de tres métodos (*fit, predict y score*) independientemente de si son clasificadores, regresores o transformadores. Esta uniformidad es lo que permite al *ML MCP Server* implementar una única herramienta *train_model* que acepta cualquier alias de modelo (*logistic_regression, random_forest, xgboost*) y ejecuta el mismo flujo de entrenamiento, serialización y evaluación sin lógica condicional por algoritmo.

2.2.2 ALGORITMOS UTILIZADOS

El *ML MCP Server* expone los siguientes algoritmos de *scikit-learn*:

- **Logistic Regression:** Modelo lineal para clasificación binaria y multiclase. Estima la probabilidad de pertenencia a una clase mediante una función logística (sigmoide) en problemas binarios o mediante extensiones como *softmax* en calsificaión multiclase. Es especialmente adecuado para datasets donde existe una relación aproximadamente lineal entre las variables de entrada (*features*) y la variable objetivo (*target*).

- **Linear Regression:** Modelo lineal para regresión. Implementado mediante LinearRegression de scikit-learn con el estimador de mínimos cuadrados ordinarios. En el sistema, actúa como baseline de referencia para tareas de regresión y como alternativa estable cuando modelos más complejos muestran overfitting severo.

- **Random Forest:** Ensemble de árboles de decisión con bagging y selección aleatoria de features. Es el algoritmo más frecuentemente seleccionado por su robustez ante outliers, su bajo riesgo de overfitting con regularización

adecuada y su aplicabilidad tanto a clasificación como a regresión sin necesidad de escalar las features.

2.2.3 BÚSQUEDA DE HIPERPARÁMETROS

RandomizedSearchCV es el componente de *scikit-learn* utilizado para la optimización de hiperparámetros en el *ML MCP Server* [19]. A diferencia de *GridSearchCV* [20], que evalúa todas las combinaciones posibles del espacio de búsqueda, *RandomizedSearchCV* muestrea aleatoriamente un número configurable de combinaciones (*n_iter*), lo que permite explorar espacios de búsqueda grandes con un presupuesto computacional acotado. En el sistema, se utiliza con *cv_folds=3* y *n_iter=10* o 20 según el tamaño del *dataset*, siguiendo las instrucciones del *system prompt* del *ML Agent.*

Para la búsqueda de hiperparámetros se hace uso de validación cruzada (*Cross-Validation,* CV), una técnica de evaluación que divide el conjunto de datos en varios subconjuntos (*folds*). En cada iteración, el modelo se entrena con una parte de los datos y se valida con el subconjunto restante, repitiendo el proceso hasta utilizar todos los *folds* como conjunto de validación. Esto proporciona una estimación más robusta del rendimiento del modelo y reduce el riesgo de sobreajuste a una única partición de los datos.

2.3 XGBOOST

XGBoost (*eXtreme Gradient Boosting*) es una implementación optimizada del algoritmo de *gradient boosting* publicada por *Chen* y *Guestrin* en 2016 [21]. Se convirtió en el algoritmo dominante en competiciones de ML sobre datos tabulares durante varios años, gracias a su eficiencia computacional, su robustez ante valores faltantes y su capacidad para capturar relaciones no lineales complejas. En el sistema de este trabajo, *XGBoost* complementa el catálogo de *scikit-learn* para tareas donde el *ML Agent* estima que la complejidad del *dataset* justifica un modelo de *boosting* sobre un ensamble de *bagging*.

2.3.1 INTEGRACIÓN CON *SCIKIT-LEARN*

XGBoost expone una API compatible con el patrón *estimator* de *scikit-learn* mediante las clases *XGBClassifier* y *XGBRegressor*, lo que permite su integración directa en el *ML MCP Server* sin código específico por algoritmo: la misma herramienta train_model que entrena un *RandomForest* entrena un *XGBoost*, simplemente cambiando el alias del modelo. Esta compatibilidad es una propiedad

del diseño del catálogo de algoritmos del servidor: todos los modelos expuestos son compatibles con la API de *scikit-learn*, garantizando la uniformidad del flujo de entrenamiento independientemente del algoritmo seleccionado.

2.3.2 HIPERPARÁMETROS RELEVANTES

El espacio de búsqueda de hiperparámetros de *XGBoost* en el *ML MCP Server* incluye *n_estimators* (número de árboles), *learning_rate* (tasa de aprendizaje), *max_depth* (profundidad máxima de cada árbol), *subsample* (fracción de muestras por árbol) y *colsample_bytree* (fracción de features por árbol). Los dos últimos actúan como mecanismos de regularización que reducen el *overfitting* al introducir aleatoriedad en la construcción de cada árbol, análogamente al *bagging* de Random Forest.

2.4 FASTAPI Y UVICORN COMO BASE DE LOS SERVIDORES MCP

Cada servidor MCP del sistema es una aplicación *FastAPI* [22] servida mediante *Uvicorn*, [23] el servidor ASGI (*Asynchronous Server Gateway Interface*) de alto rendimiento recomendado para entornos de producción con *FastAPI*. La primera parte de esta serie describió en detalle la arquitectura técnica de *FastAPI* y su rol en los *endpoints* A2A de los agentes; aquí se describe únicamente el rol específico de *FastAPI* en el contexto de los servidores MCP.

En los servidores MCP, *FastAPI* implementa dos tipos de *endpoints*. El *endpoint* de descubrimiento de herramientas (*/tools/list*) es una petición HTTP GET que devuelve el catálogo completo de herramientas disponibles en el servidor con sus esquemas JSON. El *endpoint* de invocación *(/tools/call*) es una petición HTTP POST que recibe el nombre de la herramienta y sus parámetros, ejecuta la función Python correspondiente y devuelve el resultado serializado en JSON. Para operaciones largas como el entrenamiento de modelos, el *endpoint* utiliza *StreamingResponse* con *Server-Sent Events* (SSE), permitiendo al agente recibir actualizaciones de progreso antes de que la operación complete.

Un aspecto relevante del diseño es la validación automática de parámetros mediante *Pydantic* [24]: cada herramienta MCP define un *inputSchema* JSON que *FastAPI* valida automáticamente antes de ejecutar la función. Esto garantiza que los parámetros malformados (por ejemplo, un *model_alias* con un valor que el servidor no reconoce) son rechazados con mensajes de error descriptivos antes de llegar al código de entrenamiento, mejorando la robustez del sistema ante decisiones inesperadas del LLM.

2.5 PROTOCOLOS MCP Y A2A

Los protocolos MCP y A2A se describen en profundidad en la primera parte de esta serie, donde se analiza su especificación técnica, su pila de protocolos y su implementación en el sistema. Desde la perspectiva de este trabajo, su rol es el siguiente:

MCP [5] es la interfaz a través de la cual los agentes del sistema invocan las herramientas de ML implementadas en los servidores. Desde la perspectiva de los servidores MCP, el protocolo define el contrato que cada herramienta debe cumplir: un nombre único, una descripción en lenguaje natural que el LLM utiliza para decidir cuándo invocarla, y un *inputSchema* que especifica los parámetros esperados. Diseñar buenos *inputSchemas* y descripciones de herramientas es una tarea de ingeniería no trivial: descripciones ambiguas producen invocaciones incorrectas del LLM, y *schemas* demasiado restrictivos impiden que el agente pase parámetros válidos en situaciones no anticipadas.

A2A [4] es el protocolo mediante el cual el *Orchestrator Agent* delega las fases del pipeline a los agentes especializados. Desde la perspectiva de este trabajo, A2A es el mecanismo que garantiza que el reporte completo del *Data Agent* llegue íntegro al *ML Agent*, y que los reportes de ambos lleguen al *Evaluation Agent*. La calidad del contexto que fluye por A2A entre fases determina directamente la calidad de las decisiones de los agentes receptores: un reporte del *Data Agent* incompleto o mal estructurado produciría selecciones de modelo sub-óptimas en el *ML Agent*, independientemente de la calidad de las herramientas MCP disponibles.

La separación entre MCP (qué ejecutan los agentes) y A2A (cómo se comunican) que define la arquitectura del sistema tiene una consecuencia práctica directa en este trabajo: mejorar las herramientas ML de los servidores MCP no requiere modificar nada de la capa A2A ni de los agentes, y viceversa. Los dos de esta serie pueden desarrollarse de forma independiente precisamente porque trabajan sobre capas distintas del mismo sistema.

Capítulo 3. ESTADO DE LA CUESTIÓN

El presente capítulo revisa el estado del arte en dos ámbitos directamente relevantes para este trabajo: el *AutoML* clásico basado en búsqueda algorítmica y las aproximaciones más recientes basadas en agentes LLM para la automatización del ML. El objetivo de esta revisión es doble: por un lado, establecer el marco de referencia respecto al cual se compararán los resultados del sistema en el capítulo de resultados; por otro, identificar las brechas que justifican el enfoque adoptado. El estado del arte en protocolos de comunicación (MCP, A2A) y sistemas multi-agente fue revisado exhaustivamente en la primera parte del trabajo y no se duplica aquí.

3.1 AUTOML CLÁSICO

Los primeros sistemas *AutoML* automatizaron la selección de modelos y la optimización de hiperparámetros mediante técnicas de búsqueda algorítmica. Esta sección revisa los sistemas más representativos y relevantes para la comparativa experimental de este trabajo.

3.1.1 AUTO-SKLEARN

Auto-sklearn [25], desarrollado por Feurer et al. en la Universidad de Friburgo, es uno de los sistemas *AutoML* más influyentes de la literatura. Su diseño combina tres componentes fundamentales: metaaprendizaje para inicializar la búsqueda en configuraciones prometedoras basándose en el rendimiento histórico obtenido en conjuntos de datos similares; optimización bayesiana mediante SMAC para explorar de forma eficiente el espacio de configuraciones; y construcción de ensamblados (*ensembles*) a partir de las mejores configuraciones encontradas. No obstante, la primera versión de esta librería ya no está mantenida por lo que se recomienda usar la versión 2.0.

Auto-sklearn 2.0 [26] introdujo mejoras significativas en la estrategia de búsqueda, así como un modo basado en *Successive Halving* que reduce el tiempo necesario para evaluar configuraciones prometedoras. No obstante, su principal limitación continúa siendo el tiempo de búsqueda: en conjuntos de datos complejos, la obtención de resultados competitivos puede requerir varias horas de optimización secuencial.

3.1.2 TPOT

TPOT [11] (*Tree-based Pipeline Optimization Tool*) adopta un enfoque radicalmente diferente al de *Auto-sklearn*, basado en el uso de programación genética para evolucionar pipelines completos de aprendizaje automático a lo largo de múltiples generaciones. Estos pipelines pueden incluir etapas de preprocesamiento, selección de características y elección del algoritmo de aprendizaje, optimizadas de manera conjunta.

Su principal ventaja frente a los sistemas basados en optimización bayesiana radica en su capacidad para explorar espacios de búsqueda discontinuos y generar combinaciones novedosas de transformaciones que difícilmente serían consideradas mediante estrategias de búsqueda más dirigidas. Además, una de sus características más distintivas es la posibilidad de exportar el pipeline óptimo como código Python ejecutable, lo que facilita la interpretación, validación y reutilización de los resultados por parte de los desarrolladores.

No obstante, esta flexibilidad tiene un coste significativo en términos computacionales, ya que los algoritmos evolutivos suelen requerir un elevado número de evaluaciones para converger hacia soluciones de alta calidad.

3.1.3 FLAML

El marco de trabajo *Fast and Lightweight AutoML* (FLAML), introducido por Wang et al. (2021) [27], surge como una solución eficiente para abordar el elevado coste computacional y de tiempo asociado a los sistemas de *AutoML* tradicionales. A diferencia de las aproximaciones convencionales que dependen de búsquedas exhaustivas o costosos procesos de optimización bayesiana en espacios de búsqueda estáticos, FLAML fundamenta su ventaja competitiva en un diseño centrado en la economía de recursos económicos y temporales.

El núcleo de su arquitectura integra un optimizador novedoso denominado *BlendSearch* [28], el cual combina de manera estratégica la exploración global de baja intensidad con la explotación local intensiva. Este mecanismo permite adaptar dinámicamente el espacio de búsqueda de hiperparámetros basándose en la retroalimentación en tiempo real del rendimiento del modelo y el coste de evaluación, priorizando aquellos modelos que demuestren una convergencia rápida y un consumo mínimo de memoria, tales como *LightGBM* o *XGBoost*.

3.1.4 H2O AUTOML

H2O AutoML [12] es un sistema de alto nivel que entrena y compara automáticamente múltiples algoritmos (como *Random Forest, Gradient Boosting Machines (GBM), XGBoost,* redes de *Deep Learning y ensembles* de tipo *stacked*) dentro de un presupuesto de tiempo configurable. Su arquitectura distribuida le permite escalar a conjuntos de datos de gran tamaño, mientras que su API simplificada lo convierte en una herramienta ampliamente utilizada en entornos industriales y en *benchmarks* de referencia.

3.1.5 AUTOGLUON

AutoGluon [13], desarrollado por *Amazon*, está especialmente orientado a datos tabulares mediante un enfoque de *stacking* multinivel. El sistema entrena múltiples modelos base y, posteriormente, modelos de nivel superior que aprenden a combinar sus predicciones de forma óptima. Este esquema de ensamble intensivo suele producir resultados altamente competitivos en benchmarks estándar, aunque con el coste de una mayor complejidad computacional y una menor interpretabilidad del modelo final.

3.2 AUTOML CON AGENTES LLM

La integración de LLMs en pipelines de ML es un área de investigación muy reciente, con la mayoría de los trabajos relevantes publicados entre 2024 y 2025. Esta sección revisa los sistemas más directamente comparables al presente trabajo.

3.2.1 DS-AGENT

DS-Agent [1], publicado en ICML 2024, fue uno de los primeros trabajos en demostrar que los agentes basados en LLM pueden superar a los enfoques clásicos de *AutoML* en tareas de aprendizaje automático tabular. Su contribución principal es la integración del razonamiento basado en casos (*Case-Based Reasoning,* CBR): el agente no parte de cero, sino que recupera soluciones humanas exitosas de competiciones de *Kaggle* similares al problema actual, las adapta al contexto específico y las refina mediante retroalimentación derivada de los resultados de ejecución.

DS-Agent opera en dos fases. En la fase de desarrollo, el agente itera sobre el pipeline con acceso al resultado de cada ejecución. En la fase de despliegue, de

bajo coste, las soluciones validadas en la fase anterior se reutilizan y adaptan a nuevos conjuntos de datos con mínima intervención del LLM.

La principal limitación de *DS-Agent* es su dependencia de una base de casos construida manualmente a partir de soluciones de *Kaggle*. Sin esta base de conocimiento previo, el rendimiento del sistema se degrada de forma significativa. En contraste, el sistema propuesto en este trabajo no requiere una base de casos: las decisiones del *ML Agent* se toman exclusivamente a partir del informe del *Data Agent*, sin conocimiento previo del conjunto de datos específico.

3.2.2 AUTOML-AGENT

AutoML-Agent [2], presentado en ICML 2025, es el trabajo más similar en cuanto a arquitectura en este estudio ya que propone un *framework multiagente* para *AutoML* de extremo a extremo, con agentes especializados en recuperación de datos, preprocesamiento, selección de modelos, entrenamiento y predicción.

Su contribución diferencial es una estrategia de planificación aumentada por recuperación (*retrieval-augmented planning*), que explora múltiples planes en paralelo en lugar de seguir una única secuencia de acciones, junto con un mecanismo de verificación en múltiples etapas que valida las soluciones generadas antes de su ejecución. Los autores evalúan el sistema en catorce conjuntos de datos de siete dominios y reportan mejoras consistentes frente a sistemas *AutoML* basados en un único agente.

La principal diferencia respecto al sistema de este trabajo reside en la capa de comunicación entre agentes. *AutoML-Agent* implementa un protocolo interno propietario basado en consultas y respuestas, sin adoptar estándares de interoperabilidad como MCP o A2A. Esta elección reduce la complejidad del sistema, pero limita su extensibilidad y su capacidad de interoperar con otras implementaciones.

3.2.3 KOMPETEAI

KompeteAI [3], publicado en agosto de 2025 y evaluado en *MLE-Bench* de *OpenAI* [29], aborda dos limitaciones específicas de los sistemas basados en LLM para *AutoML*: la exploración restringida (los métodos *one-shot* carecen de diversidad y los enfoques basados en MCTS no recombinan adecuadamente soluciones parciales de alta calidad) y el cuello de botella de ejecución asociado a los largos ciclos de validación de código.

Su principal innovación técnica es un mecanismo de fusión de candidatos (*merging stage*), que combina las mejores soluciones parciales exploradas, junto con un modelo de *scoring* predictivo que estima el potencial de una solución a partir de métricas en fases tempranas, evitando la ejecución completa de candidatos poco prometedores antes de confirmar su utilidad.

Capítulo 4. DEFINICIÓN DEL TRABAJO

4.1 JUSTIFICACIÓN

La revisión del estado del arte permite identificar una brecha técnica concreta que ninguno de los sistemas revisados cubre de forma completa. Los sistemas *AutoML* clásicos (*Auto-sklearn, TPOT, H2O, AutoGluon*) producen modelos competitivos, pero sin razonamiento explícito: no justifican sus decisiones, no adaptan su estrategia a contexto semántico y no generan recomendaciones comprensibles para el usuario. Los sistemas basados en agentes LLM (*DS-Agent, AutoML-Agent, KompeteAI*) incorporan razonamiento explícito, pero implementan la comunicación de forma *ad-hoc* y propietaria, sin adoptar los protocolos estándar emergentes (MCP y A2A) que garantizan la interoperabilidad a largo plazo.

La brecha se concreta en tres dimensiones técnicas. En primer lugar, la justificación de las decisiones de selección de modelo: mientras que *Auto-sklearn* o *AutoGluon* seleccionan el mejor modelo según una métrica sin explicación, el *ML Agent* del sistema de este trabajo genera una nota en lenguaje natural que explica por qué eligió ese algoritmo y no otro, qué características del *dataset* influyeron en la decisión y qué limitaciones tiene el modelo seleccionado. Esta justificación es el artefacto que convierte el sistema en una herramienta de aprendizaje y auditoría, no solo de automatización.

En segundo lugar, la adaptación contextual: el *Data Agent* identifica características del *dataset* (distribución sesgada del target, desbalance de clases, multicolinealidad entre características) y se las comunica al *ML Agent* en lenguaje natural como recomendaciones. El *ML Agent* incorpora esta información en su razonamiento. No se ha encontrado ningún sistema *AutoML* clásico que peuda realizar este tipo de adaptación contextual basada en lenguaje natural.

En tercer lugar, la extensibilidad mediante estándares, al estar construido sobre MCP y A2A, cualquier herramienta o agente compatible con estos protocolos puede integrarse en el sistema sin modificar los componentes existentes. Un nuevo algoritmo de ML puede añadirse como herramienta MCP; un nuevo tipo de agente especializado puede integrarse como participante A2A. Esta extensibilidad estructural es inexistente en los sistemas *AutoML* revisados, que son monolíticos por diseño.

4.1.1 JUSTIFICACIÓN DE MERCADO

Este apartado busca justificar la adopción de este proyecto en base a tres perfiles relevantes en el ámbito de estudio.

Para el investigador académico, el valor reside en la trazabilidad y la reproducibilidad. Un pipeline de ML cuyas decisiones están justificadas en lenguaje natural y cuyos experimentos quedan registrados con todos sus parámetros es un instrumento de investigación reproducible. Un sistema *AutoML* de caja negra produce un modelo, pero no un registro auditable del proceso que llevó a ese modelo. En entornos de investigación donde la reproducibilidad es un requisito metodológico, esta diferencia es determinante.

Para la organización sin equipo de ciencia de datos, el valor reside en la accesibilidad y la comprensibilidad. Proporcionar la ruta a un CSV y recibir un modelo entrenado con una explicación en lenguaje natural de qué se ha hecho y por qué es lo más próximo a un científico de datos autónomo que existe actualmente. La diferencia respecto a un *AutoML* clásico no es el modelo resultante, sino que el informe final puede ser leído y comprendido por alguien sin formación estadística avanzada.

Para el ingeniero de ML que necesita un *baseline* rápido, el valor reside en la velocidad y la adaptabilidad. En 60-90 segundos, el sistema produce un modelo razonable con justificación de la selección y recomendaciones de mejora, sin configuración manual. Los sistemas *AutoML* clásicos requieren típicamente minutos u horas para producir resultados equivalentes. Y si el resultado no es satisfactorio, el sistema explica por qué (*overfitting*, desbalance de clases, *skewness* del *target*) en lugar de limitarse a devolver una métrica.

4.1.2 POSICIONAMIENTO RESPECTO AL ESTADO DEL ARTE

Es necesario distinguir este trabajo de los sistemas más directamente comparables (*DS-Agent y AutoML-Agent*) en términos de ambición y alcance. Ambos sistemas compiten en *benchmarks* de *Kaggle* [30] con el objetivo de maximizar el rendimiento del modelo final, utilizando técnicas avanzadas como CBR con casos de *Kaggle* o planificación paralela con verificación multi-etapa. El presente trabajo no persigue maximizar el rendimiento sobre *benchmarks* competitivos: persigue demostrar la viabilidad de un pipeline ML automatizado con razonamiento explícito, justificación comprensible y arquitectura basada en protocolos estándar, produciendo resultados razonables en tiempos razonables sobre *datasets* estándar de acceso público.

4.2 OBJETIVOS

4.2.1 OBJETIVO GENERAL

El objetivo general de este proyecto consiste en implementar y evaluar la capa de ejecución del *pipeline* ML automatizado, compuesta por tres servidores MCP (*Data MCP Server, ML MCP Server, Evaluation MCP Server*). Cada servidor tendrá un catálogo de herramientas que cubra preprocesamiento, entrenamiento y evaluación para tareas de clasificación y regresión. Además, se compararán los resultados obtenidos frente a *baselines* de *AutoML* clásico sobre *datasets* de *benchmark* estándar.

4.2.2 OBJETIVOS ESPECÍFICOS

1. **Implementar el *Data MCP Server:*** Diseñar e implementar el servidor MCP de preprocesamiento con herramientas para análisis exploratorio (*preview_dataset, describe_dataset*), detección de problemas (*detect_problems*), transformación de datos (*preprocess_dataset*) y partición (*split_dataset*). El servidor debe manejar correctamente variables numéricas y categóricas, y producir un reporte estructurado legible por el ML Agent.

2. **Implementar el *ML MCP Server:*** Diseñar e implementar el servidor MCP de entrenamiento con herramientas para optimización de hiperparámetros (*tune_hyperparams*) y entrenamiento (*train_model*) de cuatro algoritmos: *logistic regression, linear regression, random forest, XGBoost*. Incluir registro de experimentos (*log_run*) con métricas, hiperparámetros y notas de razonamiento del agente.

3. **Implementar el *Evaluation MCP Server:*** Diseñar e implementar el servidor MCP de evaluación con herramientas para cálculo de métricas (*compute_metrics*) en clasificación y regresión, comparación entre *runs* (*compare_models*), promoción del mejor modelo (*save_best_model*) y generación del informe final en *Markdown* (*generate_report*).

4. **Evaluar el *pipeline* frente a *datasets* de *benchmark:*** Ejecutar el *pipeline* completo sobre al menos cuatro *datasets* de *benchmark* estándar que cubran clasificación binaria, clasificación multiclase, regresión y al menos un *dataset* con complejidad de preprocesamiento real (*missing values, variables mixtas, desbalance de clases*). Documentar las decisiones de los agentes y las métricas obtenidas para cada ejecución.

5. **Comparar frente a *baselines AutoML* clásicos:** Ejecutar *Auto-sklearn* o *AutoGluon* sobre los mismos *datasets* y particiones que el sistema de este trabajo, y comparar las métricas obtenidas. El objetivo no es superar a los sistemas *AutoML* clásicos en rendimiento, sino cuantificar la diferencia y analizar en qué tipos de *dataset* y problema el razonamiento contextual del agente produce decisiones equivalentes o superiores a la búsqueda exhaustiva.

6. **Analizar la calidad del razonamiento del *ML Agent:*** Evaluar cualitativamente si las decisiones de selección de modelo del *ML Agent* son coherentes con las características del *dataset* identificadas por el *Data Agent*: ¿elige algoritmos adecuados para el tipo de problema? ¿adapta su estrategia ante *datasets* con desbalance, *skewness* o multicolinealidad? ¿sus justificaciones son correctas desde la perspectiva del dominio ML?

4.3 METODOLOGÍA

El desarrollo de los servidores MCP sigue el mismo enfoque iterativo e incremental adoptado en la primera parte de la serie, con la particularidad de que la infraestructura de comunicación ya está establecida y no requiere iteración: cada iteración de este trabajo produce un servidor MCP funcional que se integra directamente con los agentes existentes sin necesidad de modificarlos.

El ciclo de cada iteración comprende cuatro fases: diseño del catálogo de herramientas del servidor (qué herramientas exponer, con qué parámetros y qué respuestas), implementación de las funciones Python correspondientes, pruebas de integración invocando las herramientas directamente desde un cliente MCP de prueba, y validación *end-to-end* ejecutando el pipeline completo con los agentes sobre un *dataset* de *benchmark*.

4.3.1 RELACIÓN CON LA PRIMERA PARTE

Este trabajo comparte la infraestructura de comunicación de la primera parte sin modificarla. Los cuatro agentes (*Orchestrator Agent*, *Data Agent, ML Agent y Evaluation Agent*) se despliegan tal como fueron implementados en el primer trabajo. Los *system prompts* de los agentes especializados se refinan en función del comportamiento observado durante las pruebas de integración, pero la arquitectura A2A, los *endpoints FastAPI* y la lógica de coordinación del orquestador

permanecen inalterados. Esta independencia entre los dos trabajos es la manifestación práctica de la modularidad arquitectónica descrita en la primera parte: mejorar la capa de ejecución no requiere tocar la capa de comunicación.

Capítulo 5. SERVIDORES MCP

Este capítulo describe la implementación de los tres servidores MCP que constituyen la capa de ejecución del sistema. Para cada servidor se describe su diseño, el catálogo de herramientas expuestas, las decisiones de implementación más relevantes y la relación entre la interfaz MCP (lo que el agente ve) y la lógica Python interna (lo que realmente se ejecuta). El código presentado es el código real del sistema.

Los tres servidores comparten un principio de diseño fundamental: son completamente *stateless*. Cada herramienta recibe todos los datos necesarios como parámetros y devuelve un resultado serializable en JSON. El estado del pipeline (qué se ha hecho, qué paths se han generado, qué métricas se han obtenido) lo acumula y gestiona el agente LLM, no el servidor.

5.1 DATA MCP SERVER

El *Data MCP Server* es el servidor de preparación de datos del pipeline. Expone cinco herramientas que cubren el ciclo completo de comprensión y preparación de un *dataset*: desde la verificación de que el fichero existe y es legible hasta la generación de los conjuntos de entrenamiento y prueba. El servidor está implementado con *FastMCP* [31] (una abstracción de alto nivel sobre el protocolo MCP que genera automáticamente los *inputSchemas* de cada herramienta a partir de las anotaciones de tipo de Python) y se sirve mediante *Uvicorn* en el puerto 9001.

5.1.1 DISEÑO DE ENTRADA

El servidor soporta seis formatos de entrada: *CSV*, *Parquet*, *JSON*, *JSONL*, *Excel* (.xlsx y .xls). La carga se gestiona mediante un *util* interno *_load_df* que selecciona el *loader* de pandas correspondiente según la extensión del fichero. Además, se añade una función de utilidad *_safe_float*, que convierte valores numéricos a *float* Python nativo transformando NaN e Infinito en *None*: esto garantiza que los resultados sean siempre serializables en JSON, evitando errores de serialización cuando las estadísticas descriptivas contienen valores especiales.

5.1.2 CATÁLOGO DE HERRAMIENTAS

5.1.2.1 Inspección preliminar (*preview_dataset*)

Esta herramienta constituye el punto de entrada al flujo de ejecución. Su propósito es validar que el *dataset* exista y sea legible, cargarlo y devolver el *schema* básico de cada columna. Además, se incluyen los tipos de datos por columna, el porcentaje de valores ausente (*missing values*) y el número de valores únicos. Esta diseñada para ser rápida y dotar al agente de los metadatos necesarios para que identifique la columna *target* sin necesidad de ejecutar el Análisis Exploratorio de Datos (EDA) completo.

5.1.2.2 Calidad de los datos (*detect_problems*)

Diseñada como mecanismo de control de calidad, esta herramienta audita el *dataset* en busca de problemas que puedan comprometer el entrenamiento. La herramienta implementa reglas heurísticas cuantitativas y umbrales de severidad estadísticos. El estado del *pipeline* se gestiona mediante la variable booleana *"can_train"*, un valor *False* indica al agente que debe detener el *pipeline.*

Los umbrales que informan sobre distintos grados de severidad (WARNIGN, CRITICAL) son configurables, por defecto se establecen valores faltantes (WARNING si >5%, CRITICAL si >40% en el target), *outliers* extremos (Z-score >5), columnas de alta cardinalidad (>50 valores únicos), columnas con correlación muy alta (>0.95) y columnas de varianza cero.

5.1.2.3 Análisis Exploratorio de Datos (*describe_dataset*)

A diferencia de los métodos de exploración de datos tradicionales, este módulo ejecuta un EDA optimizado para la comprensión semántica por parte de los LLM. Para cada columna numérica calcula media, desviación típica, percentiles, *skewness* y *kurtosis*, así como la detección de *outliers* mediante el Rango Intercuartílico (IQR).

Asimismo, se evalúa la colinealidad mediante matrices de correlación (*Pearson* o *Spearman*), aislando los pares redundantes ($\rho > 0.95$). Para la variable objetivo (*target*), identifica el tipo de tarea (clasificación o regresión) y calcula el *ratio* de desbalance de clases.

5.1.2.4 Preprocesado de datos (*preprocess_dataset*)

Esta herramienta permite aplicar un preprocesado configurable al *dataset.*El agente decide los parámetros en base al EDA obtenido de la herramienta anterior. Las operaciones disponibles incluyen eliminación de columnas (IDs, *leakage*), estrategia de imputación numérica (*mean/median/zero*) y categórica (*mode/unknown*), si aplicar *one-hot encoding* y si eliminar duplicados y constantes.

La variable objetivo (*target*) se separa antes del preprocesado y se reintegra al final para evitar transformaciones no deseadas.

5.1.2.5 Partición del conjunto de datos (*split_dataset*)

La fase final del flujo del *Data Agent* consiste en dividir el *dataset* en conjuntos de entrenamiento y *test.* La herramienta acepta tamaños de *test* de entre 0.05 hasta 0.5, y por defecto 0.2. Además, implementa una semilla de aleatoriedad (*random_state*) para asegurar la reproducibilidad de los resultados. Los conjuntos resultantes se guardan de forma automática en el directorio de salida indicado y devuelve la distribución del *target* para que el agente verifique que la estratificación es correcta.

Herramienta	**Parámetros de entrada (*inputs*)**	**Parámetros de salida (*outputs*)**
preview_dataset	*dataset_path, separator*	*status, shape, memory_usage_mb, schema_by_column, has_missing, total_missing*
detect_problems	*dataset_path, target_column*	*total_problems, metrics_counters, can_train, problems_list(severity, details)*
describe_dataset	*dataset_path, target_columns, method*	*shape, column_types, columns_analysis, correlation_matrix, target_analysis*
preprocess_dataset	*dataset_path, target_column, output_path, drop_columns,*	*original_shape, final_shape, output_path,*

	numeric_imputation, categorical_imputation, encode_categoricals, drop_duplicates, drop_constants	*columns_after, processing_report*
split_dataset	*dataset_path, output_dir, target_column, test_size, random_state*	*original_rows, split_dimensions, file_paths, target_distribution_pct*

Tabla 1 - Parámetros de entrada y retornos para cada herramienta del Data MCP Server

5.1.3 DECISIONES DE DISEÑO

Una decisión de diseño que merece mención explícita es la separación entre las herramientas *detect_problems* y *describe_dataset*. Podría haberse implementado todo el análisis en una única herramienta, pero la separación tiene una justificación funcional: *detect_problems* está diseñada para fallar rápido. Si el *dataset* tiene problemas críticos (por ejemplo; *target* con 80% de missing, fichero corrupto, cero filas) no tiene sentido ejecutar el EDA completo, que puede ser costoso en *datasets* grandes. El agente puede invocar *detect_problems* primero y abortar el pipeline antes de consumir recursos innecesarios.

Otra decisión relevante es la ausencia de *scaling* en el preprocesado del *Data MCP Server*. La normalización de *features* (*StandardScaler*) se aplica dentro del pipeline de *scikit-learn* del *ML MCP Server*, después del *split*, para evitar data *leakage*: si se normalizase antes del *split*, la normalización utilizaría estadísticos calculados sobre el *test* set, lo que sesgaría el modelo hacia datos que en producción no estarían disponibles durante el entrenamiento.

5.2 ML MCP SERVER

El *ML MCP Server* es el servidor de entrenamiento y registro de experimentos. Expone tres herramientas orientadas a las tres responsabilidades distintas que el *ML Agent* necesita cubrir: optimización de hiperparámetros, entrenamiento del modelo y registro del experimento. A diferencia del *Data MCP Server*, donde el agente sigue un flujo secuencial casi invariable, en el *ML MCP Server* el agente decide si invocar o no *tune_hyperparams* según el contexto del *dataset*, haciendo de este servidor el que más razonamiento contextual requiere del LLM.

5.2.1 ARQUITECTURA INTERNA

El componente técnico más importante del *ML MCP Server* es la función interna *_build_sklearn_pipeline*, que construye un Pipeline de *scikit-learn* completo de forma transparente para el agente. El pipeline incluye una etapa de preprocesado mediante *ColumnTransformer* que aplica transformaciones diferentes según el tipo de columna: las columnas numéricas reciben imputación por mediana seguida de *StandardScaler*; las columnas categóricas y booleanas reciben imputación por moda seguida de *OneHotEncoder* con *handle_unknown='ignore'*.

```
def _build_sklearn_pipeline(model_alias, task, hyperparams, X):
    numeric_transformer = Pipeline([
        ("imputer", SimpleImputer(strategy="median")),
        ("scaler",  StandardScaler()),
    ])
    categorical_transformer = Pipeline([
        ("imputer", SimpleImputer(strategy="most_frequent")),
        ("encoder", OneHotEncoder(handle_unknown="ignore")),
    ])
    preprocessor = ColumnTransformer(transformers=[
        ("num", numeric_transformer, numeric_cols),
        ("cat", categorical_transformer, categorical_cols),
    ])
    # Combinar defaults del MODEL_REGISTRY con hiperparámetros del agente
    defaults = MODEL_REGISTRY[model_alias]['default_params'].copy()
    defaults.update(hyperparams)
    return Pipeline([('preprocessor', preprocessor), ('model', estimator)])
```

La inclusión del preprocesado dentro del pipeline *scikit-learn* tiene una consecuencia importante: el objeto serializado por *joblib* incluye tanto el preprocesador como el modelo, lo que garantiza que el modelo en producción aplica exactamente las mismas transformaciones que durante el entrenamiento. Esto elimina el riesgo de inconsistencia entre el preprocesado de entrenamiento y el de inferencia, uno de los errores más frecuentes en pipelines ML manuales.

5.2.2 CATÁLOGO DE HERRAMIENTAS

5.2.2.1 Entrenamiento de modelos (*train_model*)

Esta herramienta constituye el núcleo ejecutor del servidor. El proceso técnico se estructura en las siguientes fases secuenciales:

1. Construcción del *pipeline:* Carga los *splits train/test* y construye el *pipeline scikit-learn* que integra los pasos de transformación y el modelo seleccionado.
2. Ajuste y evaluación: Se entrena el modelo sobre el conjunto de datos de entrenamiento y se calculan métricas sobre *train* y *test* para evaluar la capacidad de generalización.
3. Serialización: Se serializa el modelo en disco mediante la librería *joblib.*
4. Identificación: Cada ejecución se indexa mediante un identificador único (*run_id*) que consiste en 8 caracteres hexadecimales derivadors de un algoritmo UUID4.

También incluye un veredicto que clasifica el rendimiento del modelo en tres categorías: *good* (f1_weighted > 0.75 o r2 > 0.6), *acceptable* (f1_weighted > 0.5 o r2 > 0.3) y *poor* (por debajo de los umbrales anteriores). Este veredicto está diseñado para orientar al orquestador en la decisión de si activar el mecanismo de *retry* del *ML Agent*.

5.2.2.2 Optimización de hiperparámetros (*tune_hyperparms*)

Con el objetivo de encontrar la configuración óptima del modelo indicado, se ejecuta una búsqueda estocástica mediante el algoritmo de validación cruzada aleatoria (*RandomizedSearchCV*). El espacio de búsqueda se determina dinámicamente: bien mediante diccionarios preconfigurados (HPO_SEARCH_SPACES), o bien mediante valores personalizados provistos semánticamente por el agente.

La herramienta no entrena el modelo final, sino que implementa la directiva *refit=False*, es decir, solo devuelve los mejores hiperparámetros para que el agente los use en la herramienta *train_model*. Asimismo, para simplificar la respuesta de la herramienta hacia el agente se implementa un filtro de limpieza que elimina los prefijos "*model__"* de los nombres de los parámetros.

5.2.2.3 Trazabilidad de los expermientos (*log_run*)

La reproducibilidad de los experimentos se gestiando mediante esta herramienta. Cada experimento se guarda en disco en un registro de tipo JSON en la ruta *experiments/<run_id>.json*.

Más allá de las variables cuantitativas de rendimiento, el diseño de la herramienta prioriza la inclusión del metadato "*notes"*. Este campo de texto permite al agente documentar su razonamiento (por qué eligió ese modelo, qué observó sobre el

dataset, qué limitaciones detecta). Estos registros son cosumidos por el *Eval MCP Server* para llevar a cabo la comparación histórica.

Herramienta	**Parámetros de entrada (*inputs*)**	**Parámetros de salida (*outputs*)**
train_model	*train_path, test_path, target_column, model_alias, task, hyperparams, experiment_name*	*run_id, model_alias, task, hyperparams_used, train_metrics, test_metrics, model_path, verdict, overfit_check, recommendation*
tune_hyperparms	*train_path, target_column, model_alias, task, n_iter, cv_folds, scoring*	*best_params, best_score, scoring_metric, total_models_trained, search_space_used*
log_run	*run_id, model_alias, task, hyperparams, train_metrics, test_metrics, model_path, experiment_name, notes, dataset_path*	*status, run_id, experiment_path*

Tabla 2 - Parámetros de entrada y retornos para cada herramienta del ML MCP Server

5.3 EVAL MCP SERVER

El *Eval MCP Server* es el servidor de evaluación y cierre del pipeline. Su responsabilidad es calcular métricas definitivas sobre el conjunto de *test*, comparar el modelo actual con *runs* anteriores, seleccionar el mejor disponible y generar el informe final que llega al usuario. A diferencia de los servidores anteriores, cuyas herramientas trabajan sobre datos en memoria o en ficheros procesados, el *Eval MCP Server* trabaja sobre el fichero de experimentos persistidos en disco por *el ML Server*, lo que le permite comparar modelos de diferentes ejecuciones del *pieline.*

5.3.1 CATÁLOGO DE HERRAMIENTAS

5.3.1.1 Evaluación Post-Entrenamiento (*compute_metrics*)

Con el fin de evaluar de forma exhaustiva el modelo generado, esta herramienta carga el modelo serializado con *joblib* y lo aplica sobre el conjunto de *test* para

calcular métricas detalladas. Para tareas de clasificación, *accuracy*, F1 macro y *weighted*, y ROC-AUC (binario o multiclase mediante OvR). Además, se extrae una matriz de confusión estructurada en un formato serializable. Para regresión ee evalúa el error cuadrático medio (RMSE) el error absoluto medio (MAE), el coeficiente de determinación (R^2) y el error porcentual absoluto medio (MAPE).

5.3.1.2 Ranking histórico (*compare_models*)

Esta herramienta lee todos los experimentos registrados en la ruta *experiments/*.json* y construye un *ranking* ordenado por la métrica primara (f1_weighted o r2). Además, filtra por tipo de *dataset* para comparar solo modelos entrenados sobre el mismo *dataset.*

La detección de *overfitting* compara la métrica primaria (*f1_weighted* para clasificación, r2 para regresión) entre *train* y *test*: una diferencia superior a 0.10 activa el *flag overfit_detected*, que el *Evaluation MCP Server* propagará como advertencia en el reporte final.

5.3.1.3 Promoción del modelo (*sabe_best_model*)

La herramienta copia el modelo ganador del *ranking* anterior al directorio de producción. Además, se construye un fichero adicional con metadatos del modelo que consolida la trazabilidad del artefacto en producción de manera que otros sistemas referencien el modelo en producción sin conocer el *run_id.*

```
{
  "run_id": "e07e038d",
  "model_alias": "random_forest_balanced_v2",
  "source_path": "models\\random_forest_e07e038d.pkl",
  "promoted_at": "2026-06-03T11:42:31.220187",
  "production_path": "models\\production\\best_model.pkl"
}
```

5.3.1.4 Reporte final (*generate_report*)

El flujo de trabajo concluye con la síntesis de un reporte final en formato *Markdown*. El documento resultante incluye una cabecera con metadatos del experimento, tabla de hiperparámetros, tabla de métricas principales, tabla de métricas por clase y matriz de confusión (clasificación), advertencias de *overfitting*, notas del agente (*agent_notes*) y tabla comparativa de *runs* si hay más de uno disponible. El fichero se nombra bajo la siguiente rúbrica: *report_<run_id>_<timestamp>.md.*

Herramienta	**Parámetros de entrada (*inputs*)**	**Parámetros de salida (*outputs*)**
compute_metrics	*model_path, test_path, target_column, task, run_id*	*task, metrics_dict, per_class_metrics, confusion_matrix_serializable, plot_path*
compare_models	*task, top_n, run_id (opcional)*	*best_run(run_id, model_path), full_ranking_dict, overfit_warning, recommendation*
save_best_model	*run_id, model_alias, model_path, production_dir*	*status, production_path, metadata_path, run_id, model_alias*
generate_report	*task, best_run_id, metrics, model_alias, hyperparams, per_class_metrics, confusion_matrix_data, overfit_warning, ranking, agent_notes, output_dir*	*status, report_path, run_id, model_alias, summary_dict*

Tabla 3 - Parámetros de entrada y retornos para cada herramienta del Eval MCP Server

El parámetro *agent_notes* de la herramienta *generate_report* merece una mención especial desde la perspectiva del sistema. Es el único campo del informe final que contiene razonamiento LLM explícito: el *Evaluation Agent* escribe en él un análisis en lenguaje natural del pipeline completo (qué modelo se entrenó y por qué, qué indican las métricas, si hay *overfitting* y cómo reducirlo, comparativa con *runs* anteriores y recomendación final). Desde la perspectiva del usuario, este campo es el valor diferencial del sistema respecto a un *AutoML* clásico: transforma un conjunto de números en un diagnóstico comprensible.

5.3.2 INTEGRACIÓN

Los tres servidores MCP están diseñados para encadenarse en un flujo determinado, pero son completamente independientes entre sí: ningún servidor llama directamente a otro. La integración se produce a través del agente LLM, que actúa como intermediario: recibe el output de una herramienta y lo pasa como parámetro a la siguiente. Esto significa que los contratos de datos entre servidores son implícitos (definidos por los campos que cada herramienta devuelve y los que

la siguiente espera) y deben estar documentados en los *inputSchemas* y descripciones de las herramientas.

El contrato más crítico es el que conecta el *Data MCP Server* con el *ML MCP Server*: las rutas *train_path* y *test_path* generadas por *split_dataset* deben pasarse exactamente como fueron devueltas (con el separador de directorios del sistema operativo donde se ejecuta el servidor) a *train_model*. Cualquier transformación de la ruta por parte del agente (normalización, recorte, adición de prefijos) puede romper el *pipeline*.

Capítulo 6. ANÁLISIS DE RESULTADOS

Este capítulo presenta los resultados de la evaluación experimental del sistema sobre cuatro *datasets* de *benchmark* y los contrasta con tres sistemas *AutoML* clásicos: *AutoGluon* [13], *Auto-Sklearn 2.0* [26] y *FLAML* [27]. El objetivo de la comparativa no es demostrar superioridad del sistema propuesto sino caracterizar en qué condiciones el razonamiento contextual del *ML Agent* produce decisiones equivalentes a la búsqueda exhaustiva, y en qué condiciones la diferencia de rendimiento es atribuible a limitaciones del catálogo más que del razonamiento.

Esta evaluación presenta un reto metodológico clave: las métricas de rendimiento predictivo tradicionales (como F1, R^2 o *accuracy*) no son la única dimensión de valor relevante. Mientras que un sistema *AutoML* clásico optimiza exclusivamente dichos indicadores, un sistema basado en agentes LLM aporta adicionalmente justificaciones de sus decisiones, recomendaciones de mejora y un registro auditable del proceso. Por ello, este capítulo presenta los resultados cuantitativos de la comparativa y los complementa con un análisis cualitativo del razonamiento de los agentes, constituyendo este el segundo eje de evaluación del trabajo.

6.1 CONFIGURACIÓN EXPERIMENTAL

6.1.1 DATASETS Y PARTICIONES

Todos los sistemas se evaluaron sobre exactamente las mismas particiones *train/test* generadas por el *Data MCP Server* con *random_state=42*. La Tabla 4 recoge las dimensiones de cada partición. La métrica primaria de comparación es *f1_weighted* para clasificación y R^2 para regresión, coherente con las métricas usadas por el *ML Agent* en sus decisiones. Los cuatro *datasets* seleccionados cubren dos tipos de tarea (clasificación y regresión) y representan diferentes niveles de complejidad tanto en el preprocesamiento como en el modelado.

Dataset	Tarea	Train	Test	Complejidad
Iris	Clasificación	117	30	Balanceado, multicolinealidad
Titanic	Clasificación	624	156	Valores faltantes
California Housing	Regresión	436	109	Skewness target, outliers
WineQT	Clasificación	814	204	Desbalance severo, 6 clases

Tabla 4 - Características de los datasets de evaluación y sus particiones train/test (random_state=42) en todos los casos.

6.1.2 NOTAS SOBRE LOS ALGORITMOS AUTOML

AutoGluon se ejecutó con el *preset* "*best_quality*". Este *preset* activa el *stacking* multinivel más agresivo del sistema, entrenando múltiples algoritmos base (*CatBoost, LightBGM, XGBoost, Random Forest, NeuralNetFastAI*) y combinándolos en ensembles de nivel L2 y L3. Es deliberadamente la configuración mas potente de *AutoGluon* para maximizar el rendimiento como *baseline*.

Auto-sklearn2, a diferencia de *AutoGluon*, aplica meta-aprendizaje para inicializar la búsqueda y utiliza una selección basada en validación cruzada (*CV-based)* sin ensambles en su versión 2. Su *leaderboard* muestra los modelos evaluados junto con sus puntuaciones de CV, lo que permite identificar exactamente qué configuraciones considera óptimas. Resultó ser el más rápido de los tres *baselines* (~15-23 s de media), probablemente debido a la eficiencia del meta-aprendizaje.

FLAML se ejecutó con un tiempo límite variable: 120 s en clasificación y 300 s en Housing, el cual se amplió a 1000 s en WineQT para otorgarle suficiente tiempo ante el problema más difícil. A pesar de los 1000 s, FLAML produjo el peor resultado de los *baselines* en WineQT; esto indica que el problema de desbalance severo no se resuelve con más tiempo de búsqueda, sino con algoritmos específicos.

6.2 COMPARATIVA DE RENDIMIENTO

La Tabla 5 recoge las métricas de *test* de todos los sistemas sobre los cuatro *datasets* mencionados anteriormente. Las métricas primarias son f1_weighted para clasificación y R^2 para regresión, coherentes con las métricas usadas por el *ML Agent* para sus decisiones de selección y validación.

Clasificación – métrica primaria: F1 weighted

Dataset	Sistema	Modelo ganador	F1-w	Acc.	Tiempo
Iris	Este trabajo	Random Forest (HPO)	0.9333	0.9333	~30s
	AutoGluon	WeightedEnsemble_L3	0.9333	0.9333	~139s
	Auto-sklearn2	std_scaler + LDA	0.9667	0.9667	~15s
	FLAML	extra_tree	0.9333	0.9333	~236s
Titanic	Este trabajo	XGBoost (HPO)	0.7932	0.7949	~55s
	AutoGluon	WeightedEnsemble_L2	0.7838	0.7885	~139s
	Auto-sklearn2	Minmax + RandomForest	0.7938	0.7949	~13s
	FLAML	lgbm	0.7786	0.7821	~120s
WineQT	Este trabajo	RF + class_weight=bal	0.5197	0.5441	~60s
	AutoGluon	WeightedEnsemble_L2	0.5428	0.5735	~143s
	Auto-sklearn2	Minmax+MLP	0.5108	0.5441	~23s
	FLAML	rf	0.5030	0.5294	~1000s

Regresión – métrica primaria: R^2

Dataset	Sistema	Modelo ganador	R^2	RMSE	Tiempo
Housing	Este trabajo	Linear Regression	0.6529	1.3245	~65s
	AutoGluon	WeightedEnsemble_L3	0.6290	1.3693	~135s
	Auto-sklearn2	minmax + Poisson Reg.	0.6706	1.2904	~18s
	FLAML	xgboost	0.6262	1.3745	~300s

Tabla 5 - Comparativa de métricas de test entre el sistema propuesto y tres baselines AutoML. Las filas en verde corresponden al sistema propuesto. Tiempo del sistema propuesto: pipeline completo (EDA + preprocesado + HPO + entrenamiento + evaluación). Tiempo baselines: ajuste del modelo.

6.3 ANÁLISIS POR DATASET

Esta sección analiza en detalle los resultados de cada *dataset*, combinando el análisis cuantitativo de las métricas con el análisis cualitativo de las decisiones del *Data Agent* y del *ML Agent*. El objetivo es separar con precisión qué diferencias de rendimiento son atribuibles al catálogo de herramientas y cuáles serían atribuibles al razonamiento del agente si el catálogo fuese idéntico.

6.3.1 IRIS – CLASIFICACIÓN MULTICLASE BALANCEADA

Iris es el problema de clasificación más sencillo del conjunto y actúa como caso de referencia para verificar que el sistema funciona correctamente en condiciones favorables. El *dataset* tiene 150 muestras, cuatro *features* numéricas continuas, tres clases perfectamente balanceadas (50 muestras por clase) y ningún valor faltante. La única complejidad detectada por el EDA es la alta correlación entre *PetalLengthCm* y *PetalWidthCm* (r = 0.963), que introduce multicolinealidad relevante para los modelos lineales.

```
[..]
2. **Multicolinealidad:** Debido a la alta correlación entre `PetalLengthCm` y
`PetalWidthCm`, si se opta por modelos lineales, se recomienda aplicar
regularización L1 (Lasso) o considerar técnicas de reducción de dimensionalidad
como PCA si el rendimiento no es satisfactorio.
[..]
```

El resultado (*f1_weighted* = 0.9333) coincide exactamente con *AutoGluon* y *FLAML*, que también obtienen 0.9333 a pesar de probar *ensembles* multinivel y *ExtraTrees* respectivamente. *Auto-sklearn2* es el único sistema en superar esa barrera, alcanzando 0.9667 mediante LDA (*Linear Discriminant Analysis*) con *standard scaler*. El *leaderboard* de auto-sklearn2 es revelador: los tres primeros puestos son variantes del mismo algoritmo (LDA con tres escaladores distintos), todos con CV score 0.9830. Esto confirma que LDA es el algoritmo óptimo para este problema y que ningún otro algoritmo evaluado por ninguno de los sistemas puede igualarlo.

```
## Leaderboard (Top 5)

| # | Model | CV Score | Best |
|---|-------|----------|------|
| 1 | `standard_scaler_lda` | 0.9830 |  |
| 2 | `minmax_scaler_lda` | 0.9830 |  |
| 3 | `robust_scaler_lda` | 0.9830 |  |
| 4 | `standard_scaler_svc` | 0.9746 |  |
| 5 | `standard_scaler_qda` | 0.9743 |  |
```

El *ML Agent* seleccionó *Random Forest* con HPO justificando que el *dataset* es "*pequeño y linealmente separable*" y que *Random Forest* es "*robusto a la multicolinealidad entre PetalLengthCm y PetalWidthCm*". La decisión es correcta dado el catálogo disponible: en ausencia de LDA, Random Forest es la opción más adecuada. La diferencia de 0.0334 en *f1_weighted* respecto a *auto-sklearn2* es enteramente atribuible a la ausencia de LDA en el catálogo.

6.3.2 TITANIC – CLASIFICACIÓN BINARIA CON PREPROCESAMIENTO COMPLEJO

Titanic es el *dataset* más rico desde la perspectiva del preprocesamiento y el que mejor ilustra el valor del razonamiento contextual del *Data Agent*. El *dataset* contiene 12 variables con características muy heterogéneas: variables numéricas continuas (*Age*, *Fare*), variables ordinales (*Pclass*), variables binarias categóricas (*Sex*, *Embarked*), variables de alta cardinalidad (*Name*, *Ticket*, *Cabin*) y *missing values* con distribuciones radicalmente distintas (*Age* al 19.9%, *Cabin* al 77.1%, *Embarked* al 0.2%).

El Data Agent tomó decisiones correctas: para *Cabin* (77.1% de *missing*), el agente la eliminó directamente por superar el umbral de *missing* (77.1% > 40%). Para *Age*, la imputó por mediana en lugar de media, justificando la presencia de *outliers*. Para *embarked,* la imputó por moda. Además, decidió eliminar *Name y Ticket* por alta cardinalidad.

```
### EDA Summary
-   **Features numéricas:** Pclass, Age, SibSp, Parch, Fare
-   **Features categóricas:** Sex, Embarked (one-hot encoded)
-   **Missing values:** Se imputaron valores faltantes en 'Age' (mediana) y
    'Embarked' (moda). La columna 'Cabin' fue eliminada por alta tasa de missing
    values (77.1%).
-   **Class balance:** Relación 1.61 (549 no sobrevivientes, 342 sobrevivientes).
    No se considera desbalanceado.
-   **Correlaciones altas:** Ninguna correlación > 0.95.
-   **Columnas con alta cardinalidad:** 'Name', 'Ticket' y 'Cabin' fueron
    eliminadas.
- **Skewness:** 'SibSp', 'Parch' y 'Fare' presentan distribuciones muy sesgadas.
```

Los resultados de *titanic* son los más equilibrados de la comparativa. El sistema propuesto obtiene f1_weighted = 0.7932, prácticamente idéntico a *auto-sklearn2* (0.7938, diferencia de apenas 0.0006) y superior a *AutoGluon* (0.7838) y FLAML (0.7786). El ML Agent seleccionó *XGBoost* con HPO justificando que las variables mixtas y las relaciones no lineales (interacciones entre Sex, Pclass y Age, por ejemplo) del *dataset* justifican un modelo de *boosting*. Esta decisión es coherente

con el comportamiento de los *baselines*: *FLAML* también selecciona *lgbm* (LightBGM, *boosting*) [32] como mejor modelo en validación, y *AutoGluon* incluye *NeuralNetFastAI* y *CatBoost* en sus primeras posiciones.

6.3.3 HOUSING – REGRESIÓN CON TARGET SESGADO Y OUTLIERS

Housing es el único *dataset* de regresión del conjunto: 545 muestras, variables mixtas numéricas y categóricas binarias, *target price* con *skewness* positiva (1.21) y *outliers* extremos detectados en *area* y *bathrooms*. Es el *dataset* donde el comportamiento del *ML Agent* es más notable: en la tercera ejecución exploró cuatro modelos de forma autónoma dentro de una misma tarea (*XGBoost* (R^2=0.5992), *Random Forest* con HPO (R^2=0.5913), *Random Forest* altamente regularizado (R^2=0.574) y *Linear Regression* (R^2=0.6529)) antes de seleccionar el más simple, argumentando explícitamente el menor *overfitting* (gap train/test = 0.033 vs 0.39 del *XGBoost*). Al observar que todos los modelos complejos muestran *overfitting* sobre un *dataset* de 436 muestras de entrenamiento, el agente infiere correctamente que la solución no es más regularización sino simplificación del modelo. La nota de *log_run* para el entrenamiento del *dataset Housing* lo documenta explícitamente:

```
## Notas del agente

Se entrenó un modelo de regresión lineal simple (run 1c07927d) que, a pesar de su
simplicidad, ha demostrado ser el más robusto ante el sobreajuste. En comparación
con modelos más complejos como Random Forest y XGBoost, este modelo presentó la
mayor estabilidad y una capacidad de generalización superior en el set de prueba.
Aunque el R2 obtenido (0.65) es moderado, las métricas son consistentes con el
entrenamiento y no presentan indicios de overfitting crítico. Se recomienda en
futuras iteraciones explorar la ingeniería de características (e.g.,
transformaciones logarítmicas en 'price' y 'area') para mejorar la capacidad
predictiva.

## Comparativa de runs (dataset: `splits\Housing`)

| Run ID | Modelo | rmse | r2 | mae |
|--------|--------|--------|--------|--------|
| `1c07927d` | linear_regression | 1324506.9601 | 0.6529 | 970043.4039 |
| `e088ccbf` | xgboost | 1423404.0166 | 0.5992 | 1021411.0625 |
| `cc520beb` | random_forest | 1437373.6190 | 0.5913 | 1045142.6200 |
```

El resultado final (R^2 = 0.6529) supera a *AutoGluon* (0.6290, +0.0239) y *FLAML* (0.6262, +0.0267), y es inferior a *auto-sklearn2* (0.6706, diferencia de 0.0177). Este último resultado merece análisis*: auto-sklearn2* seleccionó un modelo de regresión de *Poisson* con normalización *minmax*, un algoritmo no disponible en el catálogo del sistema propuesto. Los tres primeros puestos del *leaderboard* de *auto-sklearn2*

son variantes de Poisson con distintos *scalers*, lo que confirma que la diferencia es estructural al catálogo, no al método de búsqueda.

```
## Leaderboard (Top 5)

| # | Model | CV Score | Best |
|---|-------|----------|------|
| 1 | `minmax_scaler_poisson` | 0.6630 |  |
| 2 | `robust_scaler_poisson` | 0.6630 |  |
| 3 | `standard_scaler_poisson` | 0.6630 |  |
| 4 | `minmax_scaler_kernel_ridge` | 0.6506 |  |
| 5 | `robust_scaler_sgd` | 0.6505 |  |
```

Es especialmente relevante que el *Data Agent*, el *ML Agent* y el *Evaluation Agent* identificaron de forma independiente la transformación logarítmica del *target price* como la mejora más impactante disponible. Si esta transformación hubiera estado disponible como herramienta del *Data MCP Server*, el R^2 habría previsiblemente superado el 0.70, igualando o superando a *auto-sklearn2*.

6.3.4 WINEQT – CLASIFICACIÓN MULTICLASE CON DESBALANCE SEVERO

WineQT es el *dataset* más difícil del conjunto: seis clases de calidad de vino (3 a 8) con desbalance severo. La clase 5 representa el 41.2% de las muestras de entrenamiento, mientras que las clases 3 y 8 tienen solo 5 y 20 muestras respectivamente. Ningún sistema evaluado resuelve bien este problema: los *f1_weighted* oscilan entre 0.5197 (sistema propuesto) y 0.5030 (*FLAML*). El *ratio* de desbalance entre la clase mayoritaria (5) y la minoritaria (3) es de 61:1, lo que convierte este problema en uno de los más difíciles del *benchmark*.

Los sistemas *AutoML* clásicos también fallan en las clases minoritarias, pero manejan mejor las clases intermedias. *AutoGluon* (f1_weighted = 0.5428) utiliza *NeuralNetFastAI* como mejor modelo individual, un algoritmo capaz de aprender representaciones más ricas del espacio de *features* que benefician especialmente a las clases con representación media (6 y 7). Además, hace uso de *ensembles* de *LightBGM* que itnternamente gestiona el desbalance con técnicas no disponibles en el sistema propuesto.

```
## Notas del agente

[..] El dataset presenta un desbalanceo severo y escasez de muestras en las
clases extremas, lo que dificulta significativamente la generalización. Se
recomienda considerar técnicas adicionales como SMOTE o buscar un dataset más
balanceado para futuras iteraciones.
```

6.4 ANÁLISIS DEL RAZONAMIENTO DE LOS AGENTES

Más allá de las métricas de rendimiento, el valor diferencial del sistema propuesto reside en la justificación explícita de cada decisión. Esta sección evalúa si las decisiones del *ML Agent* son coherentes con el conocimiento del dominio ML y con las recomendaciones del *Data Agent*.

6.4.1 COHERENCIA DE LAS DECISIONES

Las notas generadas por el *ML Agent* en la herramienta *log_run* documentan el razonamiento que llevó a cada selección del modelo. El siguiente fragmento pertenece a la nota generada por el *ML Agent* en el entrenamiento del *dataset* Iris.

```
Elegí Random Forest porque es altamente efectivo para problemas de clasificación
de tamaño reducido, manejando naturalmente las relaciones no lineales. Dado que
el dataset está bien balanceado, el modelo alcanzó un desempeño excelente sin
necesidad de técnicas adicionales para tratar la multicolinealidad.
```

Esta justificación es técnicamente correcta. La correlación entre *PetalLengthCm* y *PetalWidthCm* es elevada, lo que produce multicolinealidad severa. *Random Forest* es agnóstico a la multicolinealidad por diseño. Observamos que el agente razona sobre el impacto estadístico de la multicolinealidad en el algoritmo elegido, no se limita a aplicar una regla genérica.

En el siguiente fragmento se muestra el razonamiento del *Evaluation Agent* tras recibir el reporte del *ML Agent* para el *dataset Titanic.*

```
"agent_notes": "El modelo XGBoost fue entrenado para predecir la supervivencia en
el Titanic, logrando métricas sólidas en el conjunto de prueba (f1_weighted:
0.7932). El modelo muestra un rendimiento equilibrado pero existe una brecha
notable entre las métricas de entrenamiento (0.9356) y prueba (0.7932), lo cual
es una señal de overfitting que debe ser vigilada en futuras iteraciones. Se
recomienda intentar técnicas de regularización más agresivas, reducción de la
profundidad del árbol o aumentar la cantidad de datos de entrenamiento para
cerrar esta brecha. El modelo actual ha sido promovido a producción debido a su
capacidad predictiva satisfactoria para el baseline actual."
```

La mención explícita del *overfitting* (gap entre f1_weighted train (0.9356) y test (0.7932)) demuestra que el agente no se limita a reportar métricas, sino que las interpreta. El razonamiento es coherente con los resultados de los *baselines*, donde *FLAML* también elige *LightBGM* como primera opción.

Los *agent_notes* del *Evaluation Agent* en *WineQT* son especialmente ilustrativos. Tras obtener un *f1_weighted* de 0.4412 en una de las iteraciones, el agente produjo el siguiente diagnóstico:

```
## Notas del agente

El modelo actual (run 43f12477) es un Random Forest diseñado para combatir el
desbalanceo severo y el overfitting mediante una profundidad limitada
(max_depth=5) y class_weight='balanced'. Aunque se observa un rendimiento
modesto, el modelo muestra una mayor estabilidad al reducir la complejidad en
comparación con experimentos previos, disminuyendo el gap entre entrenamiento y
prueba. A pesar de esto, el rendimiento sigue siendo bajo, especialmente en las
clases minoritarias (3, 4, 8), donde la precisión y el recall son nulos.
Recomiendo explorar técnicas más avanzadas para el manejo del desbalanceo (como
SMOTE o ajuste fino de hiperparámetros mediante GridSearch) y considerar la
eliminación de outliers identificados en el EDA.
```

Un usuario sin formación estadística avanzada que recibe este informe sabe exactamente qué hacer: buscar una implementación de SMOTE. Un usuario que recibe el *leaderboard* de *AutoGluon* con f1_weighted = 0.5428 no tiene ninguna guía sobre qué hacer si ese resultado es insatisfactorio.

Este análisis muestra que las diferencias de rendimiento entre el sistema propuesto y los *baselines* son atribuibles al catálogo de herramientas, no a la calidad del razonamiento. En todos los *datasets* evaluados, el *ML Agent* tomó la decisión óptima dentro de las posibilidades del catálogo disponible. Ampliar ese catálogo (añadir LDA, MLP, regresión de Poisson, SMOTE) es una extensión de implementación que no requiere modificar la arquitectura ni los agentes.

Capítulo 7. CONCLUSIONES Y TRABAJOS FUTUROS

7.1 CONCLUSIONES

El objetivo de este trabajo era implementar y evaluar la capa de ejecución (los tres servidores MCP) de un sistema multi-agente diseñado para la automatización del *pipeline* ML y comparar los resultados obtenidos frente a sistemas *AutoML* clásicos de referencia. Los resultados experimentales sobre cuatro *datasets* y tres *baselines* permiten formular tres conclusiones principales.

La primera conclusión es que el razonamiento LLM es suficiente para tomar decisiones de selección de modelo correctas en problemas de ML estándar. En los cuatro *datasets* evaluados, el *ML Agent* seleccionó el algoritmo óptimo dentro del catálogo disponible. No se observó ningún caso donde el agente tuviese acceso al algoritmo correcto y no lo seleccionase. El razonamiento LLM, guiado por un *system prompt* bien diseñado y alimentado por el reporte estructurado del *Data Agent*, produce decisiones de calidad comparable a las de un científico de datos experto con el mismo catálogo disponible.

La segunda conclusión es que las diferencias de rendimiento respecto a los mejores algoritmos *AutoML* son atribuibles al catálogo de herramientas, no al razonamiento. Cuando el sistema propuesto queda por detrás de *auto-sklearn2*, la causa es siempre un algoritmo no disponible: *LDA* en *Iris*, regresión de *Poisson* en *Housing*, MLP y técnicas de remuestreo en *WineQT*. En el *dataset* donde todos los sistemas tienen acceso a los mismos tipos de algoritmos (*Titanic*, dominado por *boosting*), el sistema propuesto prácticamente iguala a *auto-sklearn2* y supera a *AutoGluon* y *FLAML*. Esta observación tiene una implicación directa para el trabajo futuro: la prioridad no es mejorar el razonamiento del agente sino ampliar el catálogo de herramientas del *ML MCP Server*.

La tercera conclusión es que el sistema propuesto produce valor diferencial que las métricas de rendimiento no capturan. Las justificaciones de las decisiones en lenguaje natural, las recomendaciones de mejora del *Evaluation Agent*, la trazabilidad completa de cada ejecución y la capacidad de adaptación a contexto semántico son propiedades que ninguno de los tres algoritmos *AutoML* evaluados posee. En el caso de *WineQT*, el *Evaluation Agent* identificó SMOTE como la técnica necesaria; en *Housing*, tres agentes distintos convergieron de forma

independiente en la transformación logarítmica del *target*. Estos razonamientos son artefactos de auditoría con valor práctico inmediato para el usuario, independientemente del F1 o el R^2 obtenidos.

7.2 VERIFICACIÓN DE OBJETIVOS

1. **Implementar el *Data MCP Server*:** El servidor expone cinco herramientas (*preview_dataset, detect_problems, describe_dataset, preprocess_dataset, split_dataset*) que cubren el ciclo completo de análisis y preparación de datos. Validado en las cuatro ejecuciones de los *datasets* de *benchmark*.

2. **Implementar el *ML MCP Server:*** El expone tres herramientas (*tune_hyperparams, train_model, log_run*) sobre cuatro algoritmos: *logistic_regression, linear_regression, random_forest, xgboost*. La arquitectura basada en _build_sklearn_pipeline garantiza que el pipeline de preprocesado y escalado está encapsulado en el modelo serializado, eliminando el riesgo de *leakage* en inferencia.

3. **Implementar el *Evaluation MCP Server:*** El servidor expone cuatro herramientas (*compute_metrics, compare_models, save_best_model, generate_report*) con soporte completo a clasificación y regresión. Genera reportes *Markdown* con el razonamiento del agente en el campo *agent_notes*.

4. **Evaluar el *pipeline* frente a *datasets* de *benchmark:*** Se ejecutó el pipeline sobre cuatro *datasets* (*Iris* [6], *Titanic* [7], *WineQT* [8], *Housing* [9]) que cubren clasificación binaria, clasificación multiclase, regresión y distintos niveles de complejidad de preprocesamiento. Se documentaron las decisiones de los agentes y las métricas obtenidas para cada ejecución, con análisis de la coherencia del razonamiento en cada caso.

5. **Comparar frente a *baselines AutoML* clásicos:** Se ejecutaron *AutoGluon, auto-sklearn2* y *FLAML* sobre las mismas particiones *train/test* del sistema propuesto. La comparativa cubre las métricas primarias (*f1_weighted, R^2*), los tiempos de ejecución y el análisis de las causas de las diferencias de rendimiento.

6. **Analizar la calidad del razonamiento del *ML Agent:*** El análisis de las notas de *log_run* y de los *agent_notes* del *Evaluation Agent* demuestra que las decisiones de selección de modelo son coherentes con el conocimiento del dominio ML en todos los *datasets* evaluados. Se documentaron tres justificaciones reales con análisis de su precisión técnica. No se observó ningún caso de decisión errónea dentro de las posibilidades del catálogo disponible.

7.3 TRABAJO FUTURO

Ampliación del catálogo de herramientas: La evaluación pone de manifiesto la necesidad de ampliar el catálogo de herramientas del ecosistema de servidores MCP mediante la incorporación de nuevos algoritmos y técnicas de preprocesamiento en los servidores *Data MCP Server* y *ML MCP Server*. Entre las extensiones prioritarias destacan la inclusión de métodos de remuestreo para datos desbalanceados, la incorporación de transformaciones estadísticas del target, como las transformaciones logarítmica, identificadas de forma recurrente por varios agentes en problemas de regresión y la ampliación del catálogo de algoritmos disponibles en el *ML MCP Server* con modelos como LDA y QDA para clasificación lineal, regresión de Poisson para variables objetivo enteras y modelos adicionales de *gradient boosting* con gestión nativa del desbalance, como LightGBM.

Optimización bayesiana de hiperparámetros: Como posible línea de mejora, podría evaluarse la sustitución de *RandomizedSearchCV* por técnicas de optimización bayesiana, como *Optuna* o *scikit-optimize*, con el objetivo de explorar el espacio de hiperparámetros de forma más eficiente, especialmente en conjuntos de datos de mayor tamaño donde un número limitado de iteraciones puede no ser suficiente.

Evaluación sobre *datasets* de mayor escala: La evaluación experimental de este trabajo se limita a *datasets* de hasta 1143 muestras. Una evaluación sobre datasets de escala media (10.000-100.000 muestras) y alta (>1M de filas) como por ejemplo *NYC Taxi Trip Duration* [35] permitiría caracterizar el comportamiento del sistema en condiciones más representativas de casos de uso reales, incluyendo el impacto del tamaño del *dataset* sobre el tiempo de HPO y la calidad de la selección de modelo por parte del *ML Agent*. No obstante, nótese que se han de tener en cuenta consideraciones de gestión de memoria y coste de *tokens*.

Extensión a otros modelos: Aumentar el catálogo de modelos disponibles en el *ML MCP Server* permitiría escalar el sistema a un amplio abanico de opciones. Gracias a la naturaleza de los protocolos MCP y A2A este cambio sería mínimo y permitiría incluso desarrollar soluciones más avanzadas, combinando con los trabajos futuros mencionados anteriormente (*datasets* de mayor escala).

Conexión con servidores MCP remotos: Una de las principales ventajas del protocolo MCP radica en su arquitectura basada en cliente - servidor, lo que permite la exposición y el consumo de recursos de forma remota. Como línea de investigación futura, se plantea explorar repositorios de servidores MCP externos [38][39][40] que provean herramientas complementarias. Esto permitiría diversificar y enriquecer las capacidades del *pipeline* de AutoML implementado, potenciando su modularidad y escalabilidad.

Capítulo 8. BIBLIOGRAFÍA

[1] Guo, S., Deng, C., Wen, Y., Chen, H., Chang, Y., & Wang, J. (2024b, febrero 27). *DS-Agent: Automated Data Science by Empowering Large Language Models with Case-Based Reasoning*. arXiv.org. https://arxiv.org/abs/2402.17453

[2] Trirat, P., Jeong, W., & Hwang, S. J. (2024b, octubre 3). AutoML-Agent: A Multi-Agent LLM Framework for Full-Pipeline AutoML. arXiv.org. https://arxiv.org/abs/2410.02958

[3] Kulibaba, S., Dzhalilov, A., Pakhomov, R., Svidchenko, O., Gasnikov, A., & Shpilman, A. (2025b, agosto 13). KompeteAI: Accelerated Autonomous Multi-Agent System for End-to-End Pipeline Generation for Machine Learning Problems. arXiv.org. https://arxiv.org/abs/2508.10177

[4] The Linux Foundation. (s. f.-d). A2A protocol. https://a2a-protocol.org/latest/

[5] What is the Model Context Protocol (MCP)? - Model Context Protocol. (s. f.-b). Model Context Protocol. https://modelcontextprotocol.io/docs/getting-started/intro

[6] UCI Machine Learning. (s. f.). Iris dataset [Data set]. Kaggle. https://www.kaggle.com/datasets/uciml/iris

[7] Yasserh. (s. f.). Titanic dataset [Data set]. Kaggle. https://www.kaggle.com/datasets/yasserh/titanic-dataset

[8] Yasserh. (s. f.). Wine quality dataset [Data set]. Kaggle. https://www.kaggle.com/datasets/yasserh/wine-quality-dataset

[9] Yasserh. (s. f.). Housing prices dataset [Data set]. Kaggle. https://www.kaggle.com/datasets/yasserh/housing-prices-dataset

[10] Feurer, M., Klein, A., Eggensperger, K., Springenberg, J. T., Blum, M., & Hutter, F. (2019). Auto-sklearn: Efficient and Robust Automated Machine Learning. En ˜ The Springer series on challenges in machine learning (pp. 113-134). https://doi.org/10.1007/978-3-030-05318-5_6

[11] Olson, R. S., & Moore, J. H. (2019). TPOT: A Tree-Based Pipeline Optimization Tool for Automating Machine Learning. En ˜ The Springer series on challenges in machine learning (pp. 151-160). https://doi.org/10.1007/978-3-030-05318-5_8

[12] H2O AutoML: Automatic machine learning — H2O 3.46.0.11 documentation. (2026b, enero 21). https://docs.h2o.ai/h2o/latest-stable/h2o-docs/automl.html

[13] Erickson, N., Mueller, J., Shirkov, A., Zhang, H., Larroy, P., Li, M., & Smola, A. (2020, 13 marzo). AutoGluon-Tabular: Robust and Accurate AutoML for Structured Data. arXiv.org. https://arxiv.org/abs/2003.06505

[14] He, X., Zhao, K., & Chu, X. (2020). AutoML: A survey of the state-of-the-art. Knowledge-Based Systems, 212, 106622. https://doi.org/10.1016/j.knosys.2020.106622

[15] RandomizedSearchCV. (s. f.). Scikit-learn. https://scikit-learn.org/stable/modules/generated/sklearn.model_selection.RandomizedSearchCV.html

[16] MathWorks. (s. f.). Automated machine learning (AutoML). MATLAB & Simulink Discovery. https://www.mathworks.com/discovery/automl.html

[17] Pedregosa, F., Varoquaux, G., Gramfort, A., Michel, V., Thirion, B., Grisel, O., Blondel, M., Müller, A., Nothman, J., Louppe, G., Prettenhofer, P., Weiss, R., Dubourg, V., Vanderplas, J., Passos, A., Cournapeau, D., Brucher, M., Perrot, M., & Duchesnay, É. (2012, 2 enero). Scikit-learn: Machine Learning in Python. arXiv.org. https://arxiv.org/abs/1201.0490

[18] Developing scikit-learn estimators. (s. f.). Scikit-learn. https://scikit-learn.org/stable/developers/develop.html

[19] Bergstra, J., & Bengio, Y. (2012). Random Search for Hyper-Parameter Optimization. https://jmlr.org/papers/v13/bergstra12a.html

[20] GridSearchCV. (s. f.). Scikit-learn. https://scikit-learn.org/stable/modules/generated/sklearn.model_selection.GridSearchCV.html

[21] Chen, T., & Guestrin, C. (2016). XGBoost. XGBoost: A Scalable Tree Boosting System, 785-794. https://doi.org/10.1145/2939672.2939785

[22] FastAPI - FastAPI. (s. f.-b). https://fastapi.tiangolo.com/

[23] Uvicorn. (s. f.-b). https://uvicorn.dev/

[24] Welcome to Pydantic. (s. f.). Pydantic Docs. https://pydantic.dev/docs/validation/latest/get-started/

[25] Feurer, M., Klein, A., Eggensperger, K., Springenberg, J., Blum, M., & Hutter, F. (2015). Efficient and Robust Automated Machine Learning. https://papers.nips.cc/paper_files/paper/2015/hash/11d0e6287202fced83f79975ec59a3a6-Abstract.html

[26] Feurer, M., Eggensperger, K., Falkner, S., Lindauer, M., & Hutter, F. (2020, 8 julio). Auto-Sklearn 2.0: Hands-free AutoML via Meta-Learning. arXiv.org. https://arxiv.org/abs/2007.04074

[27] Wang, C., Wu, Q., Weimer, M., & Zhu, E. (2019, 12 noviembre). FLAML: A Fast and Lightweight AutoML Library. arXiv.org. https://arxiv.org/abs/1911.04706

[28] Wang, C., Wu, Q., Huang, S., & Saied, A. (s. f.). ECONOMIC HYPERPARAMETER OPTIMIZATION WITH BLENDED SEARCH STRATEGY. OpenReview. https://openreview.net/forum?id=VbLH04pRA3

[29] Chan, J. S., Chowdhury, N., Jaffe, O., Aung, J., Sherburn, D., Mays, E., Starace, G., Liu, K., Maksin, L., Patwardhan, T., Weng, L., & Mądry, A. (2024, 9 octubre). MLE-bench: Evaluating Machine Learning Agents on Machine Learning Engineering. arXiv.org. https://arxiv.org/abs/2410.07095

[30] Kaggle. (s. f.). Kaggle: Your home for data science [Website]. https://www.kaggle.com/

[31] PrefectHQ. (s. f.). GitHub - PrefectHQ/fastmcp: 🚀 The fast, Pythonic way to build MCP servers and clients. GitHub. https://github.com/PrefectHQ/fastmcp

[32] Welcome to LightGBM's documentation! — LightGBM 4.6.0 documentation. (s. f.). https://lightgbm.readthedocs.io/en/stable/

[33] Optuna - A hyperparameter optimization framework. (s. f.). Optuna. https://optuna.org/

[34] scikit-optimize: sequential model-based optimization in Python — scikit-optimize 0.8.1 documentation. (s. f.). https://scikit-optimize.github.io/stable/

[35] Yasserh. (s. f.). NYC taxi trip duration dataset [Data set]. Kaggle. https://www.kaggle.com/datasets/yasserh/nyc-taxi-trip-duration

[36] Shen, Z., Zhang, Y., Wei, L., Zhao, H., & Yao, Q. (2018, 31 octubre). Automated Machine Learning: From Principles to Practices. arXiv.org. https://arxiv.org/abs/1810.13306

[37] James, K. (2025, 18 febrero). What Is AutoML? A Beginner's Guide to Automated Machine Learning. The Alignment AI. https://www.thealignment.ai/p/what-is-automl-a-beginner-s-guide-to-automated-machine-learning

[38] Punkpeye. (s. f.). GitHub - punkpeye/awesome-mcp-servers: A collection of MCP servers. GitHub. https://github.com/punkpeye/awesome-mcp-servers

[39] MCP servers. (s. f.). MCP.so. https://mcp.so/

[40] Smithery. (s. f.). Smithery: Connect agents to services in minutes. https://smithery.ai/

ANEXO I: ALINEACIÓN DEL PROYECTO CON LOS ODS

El presente proyecto guarda una estrecha relación con la Agenda 2030 de las Naciones Unidas. Impulsa la productividad y el trabajo de calidad, y todo ello se sostiene sobre una arquitectura abierta y colaborativa que facilita las alianzas y la transferencia de conocimiento.

ODS 9 – Industria, Innovación e Infraestructura

El noveno ODS promueve la construcción de infraestructuras resilientes, el fomento de la industrialización inclusiva y el impulso a la innovación tecnológica. Este proyecto contribuye directamente a dicho objetivo al proponer una arquitectura de automatización avanzada que democratiza el acceso al entrenamiento de modelos de aprendizaje automático. Mediante la orquestación inteligente de agentes especializados (Data Agent, ML Agent y Evaluation Agent) y la estandarización de la comunicación entre ellos a través de los protocolos A2A y MCP, el sistema reduce significativamente las barreras técnicas y operativas que históricamente han limitado el desarrollo de soluciones de IA a organizaciones con grandes recursos. Al abstraer la complejidad del *pipeline* de *Machine Learning* en agentes autónomos e interoperables, se habilita una infraestructura tecnológica escalable y reutilizable que puede adaptarse a distintos contextos industriales y académicos, fomentando así la innovación como motor de desarrollo.

ODS 8 – Trabajo decente y crecimiento económico

El octavo ODS persigue promover el crecimiento económico sostenido, inclusivo y sostenible, así como el empleo pleno y el trabajo decente. La automatización del *pipeline* de *Machine Learning* que plantea este proyecto tiene un impacto directo sobre la productividad de los equipos de ciencia de datos, permitiéndoles liberarse de tareas repetitivas y de bajo valor añadido (como la preparación de datos, la configuración de experimentos o la evaluación de métricas) para concentrarse en actividades de mayor valor cognitivo y creativo. Este aumento de la productividad contribuye al crecimiento económico de las organizaciones que adopten la solución. Asimismo, al estandarizar y documentar los flujos de trabajo mediante protocolos abiertos como MCP, se facilita la incorporación de profesionales con distintos perfiles técnicos, promoviendo entornos de trabajo más inclusivos y eficientes dentro del sector tecnológico.

ODS 17 – Alianzas para lograr los objetivos

El decimoséptimo ODS hace hincapié en la importancia de fortalecer los medios de implementación y revitalizar las alianzas mundiales en favor del desarrollo sostenible, con especial atención a la transferencia de tecnología, el intercambio de conocimiento y la cooperación entre actores públicos y privados. La arquitectura propuesta en este proyecto se asienta sobre protocolos abiertos e interoperables (A2A y MCP) que, por su propia naturaleza, están diseñados para favorecer la integración y la colaboración entre sistemas heterogéneos y equipos distribuidos. Esta apuesta por la interoperabilidad y los estándares abiertos facilita que distintas organizaciones, independientemente de su tamaño o sector, puedan integrar el sistema en sus infraestructuras existentes, colaborar en su mejora y compartir los beneficios de la automatización inteligente. De este modo, el proyecto actúa como un vector de transferencia tecnológica y de generación de alianzas técnicas que contribuyen a los principios de cooperación global promovidos por el ODS 17.

ANEXO II: CÓDIGO EMPLEADO

El código con la arquitectura completa y los reportes generados se encuentran en el siguiente repositorio de GitHub:

https://github.com/MrFat1/multi-agent-system-MCP-A2A-AutoML.git