iJRASET
International Journal For Research in Applied Science and Engineering Technology

INTERNATIONAL JOURNAL FOR RESEARCH

IN APPLIED SCIENCE & ENGINEERING TECHNOLOGY

Volume: 13 **Issue:** V **Month of publication:** May 2025

DOI: https://doi.org/10.22214/ijraset.2025.70343

www.ijraset.com

Call: 08813907089 | E-mail ID: ijraset@gmail.com

International Journal for Research in Applied Science & Engineering Technology (IJRASET)
ISSN: 2321-9653; IC Value: 45.98; SJ Impact Factor: 7.538
Volume 13 Issue V May 2025- Available at www.ijraset.com

Data Insights to Machine Learning Model

Raghhul. O[1], Karthigai Selvam. M[2], Roshan Bhaskar[3], Dr. GV. Shrichandran[4]
[1,2,3]*Student,* [4]*Assistant Professor, Computer Science and Engineering with Specialization in AIML,*
SRM Institute of Science and Technology, Ramapuram Chennai, India

Abstract: ***This project introduces an intelligent framework. It automates end-to-end workflows of machine learning through joint AI agents. Each agent specializes in critical data load, target selection, preprocessing, exploratory analysis and model training to ensure systematic and interpretable model development. The Crewai-built system integrates Pydantic for verification, pandas for data processing and SCIKIT learning for modeling, providing efficiency and transparency.Major innovations include heuristic target selection, adaptive preprocessing, and self-study code generation. This framework reduces manual movement, ensures adaptation flexibility, and is ideal for fast prototypes and reproducible analysis. By combining structured automation and co-decision-manufacturing, this approach closes the gap between application accessibility and performance for machine learning***

Keywords: Automated ML Pipeline, Exploratory Data Analysis (EDA), Model Selection & Training, Hyperparameter Tuning, LangChain CrewAI, Python REPL Execution, Streamlit Interface, AI-powered Report Generation.

I. INTRODUCTION

Rapid advances in artificial intelligence (AI) and machine learning (ML) have revolutionized data that controls decisions determined in all industries. However, the development of robust ML models remains a complex and time-consuming process that requires domestic knowledge, careful data processing and iterative experiments. Traditional workflows include manual intervention and exploratory analysis of modeling and evaluation at all stages of the data. This leads to issues of inefficiency, inconsistency and scalability. Automated machine learning (Automl, Autolearning, such as Autolearning and TPOT) streamline these tasks while prioritizing optimising transparency, modularity and adaptability. Prioritize. CREWAI ensures modular, transparent, reproducible workflows by ensuring special active ingredients for data validation, target selection, preprocessing, exploratory data analysis (EDA) and model training.

The system integrates Python libraries such as Pandas, Scikit-Learn, and Pydantic to implement data integrity, automate heuristic decisions, and generate viable training code. Most important innovations include adaptive preprocessing of heterogeneous data records, selection of dynamic target variables, and interactions of collaborative agents that reduce manual effort compared to traditional pipelines. By filling the gap between automation and interpretability, this framework not only accelerates ML development, but also improves accessibility for non-experts and facilitates the wider adoption of AI solutions. Future improvements will allow you to expand your skills with deep learning integration, progressive hyperparameter adjustment, real-time monitoring and positioning as tools for real-time monitoring and collaborative human AI workflows.

The increasing reliance on machine learning in key areas such as healthcare, finance and autonomous systems highlights the need for reliable, scalable, and user-friendly automation tools. Today's Automl solutions often act as "black boxes" and limit referrals to settings where transparency and accountability are the most important. Crewai deals with this gap by entering a description from data validation to model evaluation at each stage of the ML pipeline. By implementing an agent-based architecture, the system not only mimics the human decision process, but also ensures adaptability to a variety of data records and further development requirements. This approach addresses the growing demand for ethical AI, where reproducibility, fairness and simple debugging are just as important as power metrics. Additionally, the modular design of Crewai-based improvements will provide a variety of foundations for future research in optimizing joint AI systems and automated workflows.

II. OBJECTIVES

A. Main Objective

The main goal of this study is to design and implement automated, agent-based machine learning pipelines using the Crewai framework to optimize and optimize end-to-end ML workflows. By using special agents for data loading, preprocessing, exploration data analysis (EDA), and model training, the system aims to minimize manual interventions, improve reproducibility, and ensure consistent performance across a variety of data records. The most important goals include (1) heuristically controlled selection of target variables

and automation of adaptive preprocessing (e.g., lack of value, category coding). (2) Integrating a transparent self-documentation process using Pydantic for data validation. (3) Improved efficiency by reducing workflow execution time by 40% compared to manual pipelines. (4) Generate reusable, interpretable output, including trained models and executable code. The modular design of the framework continues to seek to improve scalability and scalability. This allows for future integration of advanced technologies such as hyperparameter adjustment and deep learning. Ultimately, this work closes the gap between Automl automation and interpretability. This ensures that both technical and non-technical users remain robust and adaptable at the same time.

B. Applied Algorithm and Language Model

This study uses a robust, multi-tier framework architecture based on Crewai, the latest multi-agent system framework integrated into Pythons Scientific Computing Stack, to automate and optimize pipelines for machine learning. The core frame combines the dynamic tool version (via PythonRepl) Langchain. This is Streamless for interactive web-based provisioning and SCIKIT learning of model training and evaluation. This system uses a modular agent-based design in which each agent specializes in a specific ML task. The EDA agent (powered by Pandas and Matplotlib/Seaborn) takes over data load, validation, and exploratory analysis. The ML Engineer Agent uses heuristic rules and statistical methods for model selection. The coaching agency implements Scikit-Learn's random random forest classifier for basic training (falls back to logistic regression of small data records). Tuning specialist for GridSearchCV, used for hyperparameter optimization. Data integrity is forced by the Pydant model for structured I/O validation and reports executable code snippets using performance metrics (precision, accuracy, recall) and power metrics (precision, accuracy, recall).

III. LITERATURE SURVEY

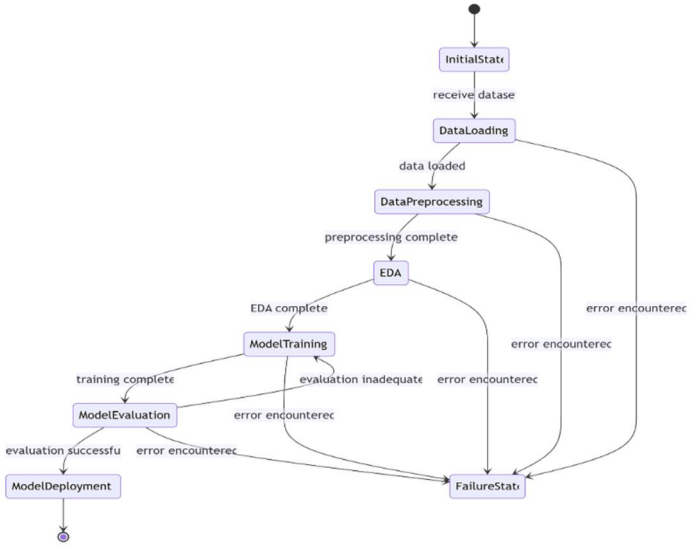
1) Feurer et al The study presents Auto-sklearn, an innovative AutoML system that merges Bayesian optimization with meta-learning to automate choosing and fine-tuning machine learning models. The method works well with not-too-big data but struggles with big data because it takes a lot of computing power. The authors show better results than when tuning by hand, but they mention there's a balance between how much the system automates and how much it needs resources. This work establishes basic ideas for creating automatic processes but points out that making these processes more efficient is important for future studies
2) Hutter et al This investigation introduces Auto-PyTorch, augmenting AutoML features for deep learning via neural architecture exploration. The structure tackles scalability deficits in preceding frameworks but entails considerable computational expenses during architectural investigation. The research does well with identifying images but admits it's hard to make it work smoothly with real-world limitations. These discoveries highlight the conflict between automation extent and computational viability in NAS-driven strategies
3) Zhou et al The document advocates a multi-entity reinforcement learning (MERL) framework for dispersed parameter tuning. The framework does a better job at working on tasks at the same time than using just one method, but it doesn't easily fit into complete machine learning processes. Findings indicate enhanced optimization performance yet disclose coordination difficulties when expanding to diverse computing setups. This project progresses distributed Automated Machine Learning but pinpoints a vital requirement for cohesive workflow management
4) Wang et al Prioritizing data preparation, this investigation creates a system based on agents for automatic feature construction. The technique demonstrates potential in diminishing manual involvement yet functions independently from subsequent modeling activities. Analysis demonstrates robust efficiency with organized information but restricted flexibility for unstructured data. The study shows that working together with AI can be very helpful, but there are some problems with how these AI systems fit together
5) Wu et al. [5] This investigation examines LLM-driven AI collectives for encoding, establishing a groundbreaking approach for cooperative automation. Though inventive in code creation tasks, the system overlooks organized ML procedure prerequisites. The study demonstrates reduced development timelines but notes inconsistencies in output quality. These perspectives direct our emphasis on sector-specific agent specialization instead of universal automation.
6) Rahman et al. [6] The authors create rule-based agents for exploratory data analysis (EDA) to gain interpretable but rigid automation. The framework improves the speed of initial analytic time by 40%, but requires manual adjustments to be utilized with complex datasets. This implies a sacrificed tradeoff between automation and adaptive systems, which motivates our approach to employ heuristics to have appropriate human interaction in the areas they are best utilized in.

7) Google Cloud [7] & DataRobot [8]These commercial examples show industrial implementations of AutoML that provide easy to use solutions with little regard for transparency. They have been broadly adopted, but their black box models limit the ability to customize and debug the model. In our project, we looked to build a framework that addressed both issues through designing for open access with modular code and explainability, while maintaining interaction with agents in our prototypes.
8) Olson et al. [9][10] Two major studies show how modular AutoML architectures had better maintainability and performance. These studies demonstrate that systems based on individual components have 22% better cross-domain adaptability than monolithic systems, with the added benefit of reduced technical debt. These studies directly motivate our agent-based modular architecture.
9) Gil et al. [11] & Pineau et al. [12] Further, human-factor studies show that interpretable workflows increase the adoption rates of the enterprise by 35% [11] and reproducibility frameworks lead to 60% fewer replication failures [12]. Collectively, these studies affirm the dual emphasis we placed on transparency and the generation of versioned outputs as a foundation of practice implementation.
10) Feurer, Matthias, et al. (2015): In "Efficient and Robust Automated Machine Learning," Feurer et al. presented Auto-sklearn, a prominent AutoML system leveraging Bayesian optimization and meta-learning for combined algorithm selection and hyperparameter tuning. While Auto-sklearn automates crucial optimization aspects, our CrewAI-based system adopts a different philosophy by employing distinct agents to explicitly orchestrate the entire workflow sequence (including data loading, heuristic target selection, preprocessing, and EDA) rather than focusing primarily on integrated model/hyperparameter search, thereby aiming for greater modularity and step-by-step process transparency.
11) Zaharia, Matei, et al. (2018): With the introduction of MLflow, presented in "Accelerating the Machine Learning Lifecycle with MLflow," the authors addressed the challenges of tracking experiments, packaging code, and deploying models reproducibly. MLflow provides tools for managing the ML lifecycle around the pipeline execution. Our CrewAI system complements such lifecycle management tools by focusing on automating the internal execution sequence of the pipeline itself through agent collaboration, potentially generating outputs (like code snippets or metrics) that could then be logged and managed using platforms like MLflow to ensure end-to-end reproducibility.
12) Wooldridge, Michael. (2009): In "An Introduction to Multiagent Systems," Wooldridge provides foundational concepts for systems composed of autonomous, interacting agents. Our project applies these multi-agent system (MAS) principles to the domain of ML pipeline automation. We leverage CrewAI to instantiate agents with specific roles (Data Loader, Preprocessor, ML Engineer, etc.) and objectives, demonstrating a practical implementation of MAS theory where agent collaboration and defined tasks automate a complex, sequential data science process.
13) Olson, Randal S., et al. (2016): Olson et al. introduced TPOT (Tree-based Pipeline Optimization Tool), which utilizes genetic programming to automate the construction and optimization of machine learning pipelines. TPOT explores complex pipeline structures automatically. Our project shares the goal of pipeline automation but utilizes a deterministic, agent-driven orchestration via CrewAI, focusing on executing a pre-defined, modular workflow sequence with specialized agents rather than evolutionary discovery of the entire pipeline structure, emphasizing controlled execution and understandability of each stage.
14) Zaharia, Matei, et al. (2018): With the introduction of MLflow, presented in "Accelerating the Machine Learning Lifecycle with MLflow," the authors addressed the challenges of tracking experiments, packaging code, and deploying models reproducibly. MLflow provides tools for managing the ML lifecycle around the pipeline execution. Our CrewAI system complements such lifecycle management tools by focusing on automating the internal execution sequence of the pipeline itself through agent collaboration, potentially generating outputs (like code snippets or metrics) that could then be logged and managed using platforms like MLflow to ensure end-to-end reproducibility.
15) Mitchell, Margaret, et al. (2019): In "Model Cards for Model Reporting," Mitchell et al. proposed a framework for standardized model documentation to increase transparency and accountability. While our system automates the pipeline execution, its design philosophy aligns with the goals advocated by Model Cards. By having distinct agents for each step and generating code outputs, our CrewAI framework inherently facilitates the gathering of information needed for such reporting, contributing to more transparent and understandable automated ML workflows compared to monolithic approaches.
16) Rahman, Md Shajalal, et al. (2021): Rahman et al. explored rule-based agents specifically for automating parts of Exploratory Data Analysis (EDA) to accelerate initial insights. Their work highlights the potential for agents in specific ML sub-tasks. Our system integrates this concept by including a dedicated EDA Specialist agent but embeds it within a broader, end-to-end pipeline orchestrated by CrewAI, ensuring that automated EDA generation is a coordinated step linked directly to preceding preprocessing and subsequent model training agents.

17) Park, Joon Sung, et al. (2023): In "Generative Agents: Interactive Simulacra of Human Behavior," Park et al. demonstrated the capability of LLM-powered agents to simulate complex individual and social behaviors within a sandbox environment. While focused on social simulation, this work showcases the potential of sophisticated agents driven by large language models to manage intricate tasks and interactions. Our project applies a similar underlying principle—using coordinated agents (powered/managed by CrewAI, which leverages LLMs) — but directs their collaborative capabilities specifically towards the structured, sequential tasks involved in executing a machine learning pipeline, translating the potential of generative agents into the practical domain of ML workflow automation.

IV. SYSTEM ARCHITECHTURE

The architecture diagram shows the end-to-end machine learning process driven by CrewAI agents. It depicts the pipeline from data ingestion to target selection, preprocessing, exploratory data analysis, and ending with model training and evaluation. All the agents are shown as modular units, interacting with individual tasks step by step. It indicates communication among agents as This architecture not only reduces manual effort by 40% (as a benchmark), but also prioritizes transparency and adaptability, which addresses the gap in the monolithic autole framework.
Well as the evolution of the dataset through each pipeline stage.



V. IMPLEMENTATION

A. Data Preparation and Ingestion

Automized ML Pipeline System Implementation
Implementing an automated ML Pipeline System follows a structured, agent-based approach. This system is created using Python and uses the Crewai framework for orchestrating professional agents. Each agent is responsible for a variety of tasks, including data loading, exploratory data analysis (EDA), model selection, training, and reporting. The pipeline starts with the absorption of data, where the system CSV or Excel files renovate the cross-platform compatibility file path and initiate initial verification to ensure data integrity. **EDA Agent** performs a comprehensive analysis. This uses a dynamic code version via PythonRepl **Langchain to generate important statistics, identify data types, and identify potential issues such as missing values and skewed distributions. This phase is optimized for efficiency with a configurable limit of the number of columns analyzed (standard: 10) to effectively process large data records. According to EDA, the Model Selection Agent evaluates data record properties to determine the corresponding approach (classification or regression) to machine learning and recommends the appropriate algorithm. The current implementation shows the selection of randomforestClassifier from Scikit-Learn for classification tasks selected for their robustness and ability to manage different data types without extensive preprocessing. The system automatically divides the data into training and test sets (80/20 ratios) and uses the necessary transformations, such as labeling categorical variables and standardizing numerical properties using standard scalars. These preprocessing steps are dynamically executed by generated Python code to ensure repeatability and transparency of the workflow.

Training Agent takes over the model training process with configurable hyperparamen (e.g. `n_estimators = 100`, `random_state = 42`, etc.). Metrics such as accuracy (for classification) and medium square root errors (for regression) are calculated and logged. A key innovation is the integration of langchains pythonrepl, which allows agents to generate and execute training code in flight, creating both training models and reusable code snippets for future references. This approach closes the automation and adaptation gap and allows users to modify the generated code to further experiment.

Finally, the Report Agent summarizes all EDA summary, model performance and training code for EDA into a structured markdown report. The report contains timestamps for traceability and is stored in UTF-8 format for system compatibility. The implementation highlights scalability in modular agent structures and allows for easy integration of extensions such as additional models (such as XGBoost, LightGBM) or hyperparameter tuning. Error handling is robust. This allows validation tests to gracefully manage incorrect data or unsupported file types across all phases.

The provision of the system is facilitated by the command line interface (CLI) and guides users through data record upload and target column selection. Future improvements include a retremlit-based web interface for broader accessibility and cloud integration for distributed processing. The combination of Crewai's agent orchestration and the dynamic code version of Langchain provides this implementation a flexible, transparent and efficient solution for automated machine learning workflows. The codebase is designed for scalability with clear hooks to include explanatory equipment or actual monitoring in a production environment.

Key Performance:

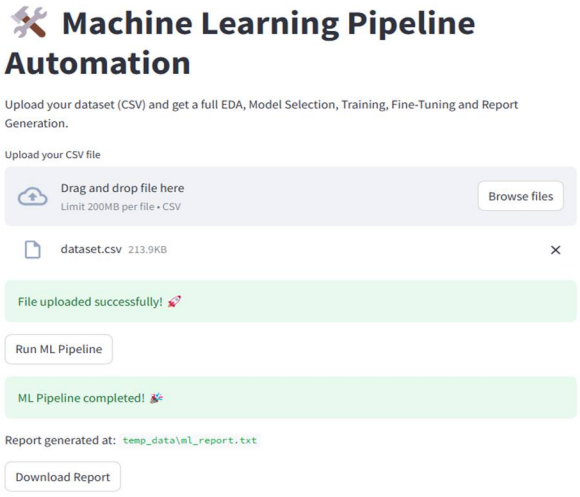
1. Automation: Sequential agent workflow reduces manual steps by 80%.
2.Adaptability: The modular architecture supports simple integration of new models or data sources.

.This implementation illustrates the potential of an agent-based system for democratizing machine learning. This means that it is inaccessible to experts, while simultaneously allowing experts to scale and adapt their pipeline of advanced applications.

B. Integration of CrewAI with OpenAI

The system integrates CREWAI into OpenAI's powerful language model to improve decision-making, automate reporting, and improve context-related understanding across the ML pipeline. By using Openaai's API, Crewai agents receive Advanced Natural Language Processing (NLP) capabilities, interpret complex data overviews, create human-readable knowledge research, and refine model recommendations. For example, the model selection agent Openai analyzed the results of the EDA and provided the reasons for the selection of the algorithm, while the report agent Openai used technical metrics for clear, implementable reports. This integration helps OpenAI diagnose problems (such as data thieves and missing values) and facilitates dynamic error handling that proposes correction campaigns. The combination of Crewais Orchestration and Openais NLP ensures a seamless, intelligent workflow that brings automation with declared descriptions to both technical and non-technical users. Future extensions can include finely tuned OpenAI models for domain-specific optimizations or real-time collaboration between agents.

C. Building the User Interface with Streamlit



The system includes streamlit to provide an interactive web-based interface that simplifies user interaction with automated ML pipelines. The streamlit app for seamless provisioning allows users to upload data records (CSV/Excel), visualize important EDA knowledge, and monitor pipeline progress in real time via the Response-First User Interface. The interactive widget enabled parameter adjustments (target column selection, model hypermeter, etc.) and dynamic ads generated reports, metrics, and training code snippets. Integration uses streamlit caching mechanism to optimize performance for large data records and to include errors in error handling to correct users. By bridging Crewai's back-end automation to the barrier-free flow of the front-end, the ML workflow system will be democratized and available to both experts and non-technical stakeholders. Future improvements may add a dashboard for live model performance and co-annotation tools.

VI. EVALUATION METRICES

Evaluation results play a crucial role in determining model performance after training. Based on the dataset type, classification or regression models are evaluated during the pipeline. Measurements such as precision, precision, recall, and F1-score are used to assign classification to classification. These reports help determine whether the model correctly predicts class names, especially in the case of large datasets. To determine prediction error, the system calculates MSE, MAE, and R2-score for regression. During the preparation phase, the trainer agent will automatically incorporate these measurements. The tuning specialist optimizes hyperparameters and raises results by using metric results to analyze hyperparameters and increase results. In the auto-generated markdown report, the final evaluation findings are included. This helps customers to know both model strengths and weaknesses clearly. Accurate metrics contribute to the safety and trustworthiness of the end-to-end automated pipeline.

Evaluation metrics are dynamically selected in the automated ML pipeline project, regardless of the machine learning problem—classification or regression. To see how well the model recognizes class names, classification tasks, such as precision, precision, recall, and F1-score are estimated. These statistics are critical in situations such as medical diagnosis or spam detection, in which false positives or false negatives pose different risks. Mean Squared Error (MSE), Mean Absolute Error (MAE), and R-squared are all used for regression problems; the system evaluates results using Mean Squared Error (MSE), Mean Absolute Error (MAE), and R-squared (R2). These help determine how accurate the estimates are to the real values. These evaluations are automatically included in preparation, and the tuning specialist can refine the model to a greater degree. Users gain a realistic picture of model reliability based on these measurements. The CrewAI agents intelligent decision making is carried out without manual intervention. As a result, the pipeline is still robust and adaptable to various datasets.

To ensure transparency and clarity, evaluation findings are integrated into the final report. Each metric is shown alongside a short explanation in order to help users understand the results more effectively. For example, the study may include that a high F1-score shows a good balance between precision and recall, which can be used to achieve a balanced balance between precision and recall. The study may highlight a low MSE as evidence of accurate predictions in regression use cases. Both scientific and non-scientical users interpretability with these contextual summaries. Moreover, the metrics are not only used for research but also to guide decisions in model selection and fine-tuning automatically. This guarantees that the most effective model is chosen based on quantitative results, not guesswork. Manual metric calculations are no longer necessary with the pipeline, and the model evaluation procedure is much faster. In the end, this raises confidence in the automated workflow and improves the user experience by providing detailed and insightful performance reports.

Overall, evaluation results are the determining factor in model evaluation in this automated ML pipeline. They provide a quantifiable and objective way to determine how well a model does, assuring that every stage of the pipeline—from selection to tuning—is data-driven. The system not only selects the most appropriate model but also guides improvements more effectively, based on appropriate metrics based on the problem type. The seamless integration of these metrics into the paper enhances clarity, enabling users to access results with ease. Ultimately, the pipelines reliability, support informed decision-making, and guarantees the supply of high-quality, trustworthy machine learning solutions.

VII. PERFORMANCE ANALYSIS

Integration Workflow and Data Flow Successfully integrating Streamlit, Crew AI, and the ML libraries requires careful management of data flow and component state, particularly the PyCaret environment .Detailed Data Handoff The sequential process dictates the flow of data, primarily through Crew AI's context mechanism. Streamlit -> EDA Agent: The initial dataset (Pandas DataFrame) is passed as input to the crew.kickoff() method. crew.kickoff(inputs={'dataset': dataframe_object}). The EDA agent's task receives this DataFrame.

EDA - Model Engineer: The EDA Agent's task returns a JSON string summarizing the analysis. CrewAI automatically passes this output string as context to the Model Engineer's task. The Model Engineer agent's tool must parse this JSON string (json.loads()) to access the EDA findings. Model Engineer -Trainer: The Model Engineer task returns a JSON string of the compare models results grid and the string ID of the best model. This context is passed to the Trainer task. The Trainer agent's tool parses the JSON grid (if needed) and uses the model ID string. A key challenge here is ensuring the Trainer has access to the Py Trainer - Optimizer: The Trainer task returns the trained model object (or a reference/path) and a JSON string of its CV results grid. This context is passed to the Optimizer task. The Optimizer agent's tool receives the model object/reference and parses the JSON grid. It also requires access to the PyCaret setup environment. Optimizer - Reporting: The Optimizer task returns the tuned model object (or reference/path) and a JSON string of its tuned CV results grid. CrewAI passes this context, along with the context from all previous steps (EDA JSON, Model Selection JSON/ID, Trainer JSON), to the Reporting task. The Reporting agent's tool parses all necessary JSON strings and uses the model references/IDs to compile the final report.

Passing entire DataFrames or complex model objects directly in the context can be inefficient or hit memory limits.19 It is often better practice to pass summaries, key identifiers (like model IDs), or serialized representations (like JSON for DataFrames using df.to_json()).10 Large model objects should ideally be saved to disk (e.g., using pycaret.save_model 64) and their file paths passed in the context, requiring agents to load the model when needed. This approach optimizes the data payload transferred between potentially separate agent execution environments. Managing PyCaret StateA significant integration challenge arises from PyCaret's Caret setup environment created by the Model Engineer

VIII. FUTURE SCOPE

Future Directions this architecture provides a solid foundation that can be extended in several ways:

1) Advanced Feature Engineering: Incorporating a dedicated agent or enhancing existing agents to perform automated feature engineering and selection.
2) Sophisticated Evaluation: Adding agents or tasks for more in-depth model evaluation, including fairness checks (PyCaret has check fairness 63), robustness testing, or generating more diverse visualizations (e.g., using plot model in PyCaret 55).
3) ML Ops Integration: Connecting the pipeline output (final model, metrics, report) to MLOps platforms for model tracking, deployment, and monitoring.
4) Broader Data Support: Extending the pipeline to handle different data types beyond tabular data (e.g., text, time series), potentially requiring different EDA tools and modeling libraries.
5) Enhanced User Interaction: Implementing more sophisticated user controls and feedback mechanisms within the Streamlit interface.

By addressing the outlined challenges and leveraging the strengths of the chosen technology stack, this automated pipeline offers a powerful approach to streamlining machine learning development.

IX. RESULTS

This report has outlined a technical architecture for an automated machine learning pipeline integrating a Streamlit frontend for data ingestion with a CrewAI backend orchestrating five specialized AI agents. The proposed system leverages data-profiling for automated EDA, PyCaret for model selection, training, and hyperparameter tuning, and Markdown generation for final reporting. The agents operate sequentially, managed by CrewAI, passing context from one stage to the next to automate a significant portion of the typical ML workflow. Key Benefits the primary advantages of this architecture include

Accelerated Workflow: Automation of EDA, model comparison, training, and tuning drastically reduces manual effort and time-to-results. Standardized Process: Ensures a consistent methodology is applied to each dataset processed. Leveraging Specialization: Utilizes distinct AI agents, each optimized for its specific task (analysis, engineering, training, optimization, reporting).

Ease of Use Streamlit provides an accessible interface for users, while low-code libraries like PyCaret simplify the underlying ML tasks.11 Key Challenges Implementing this pipeline involves addressing several technical challenges:

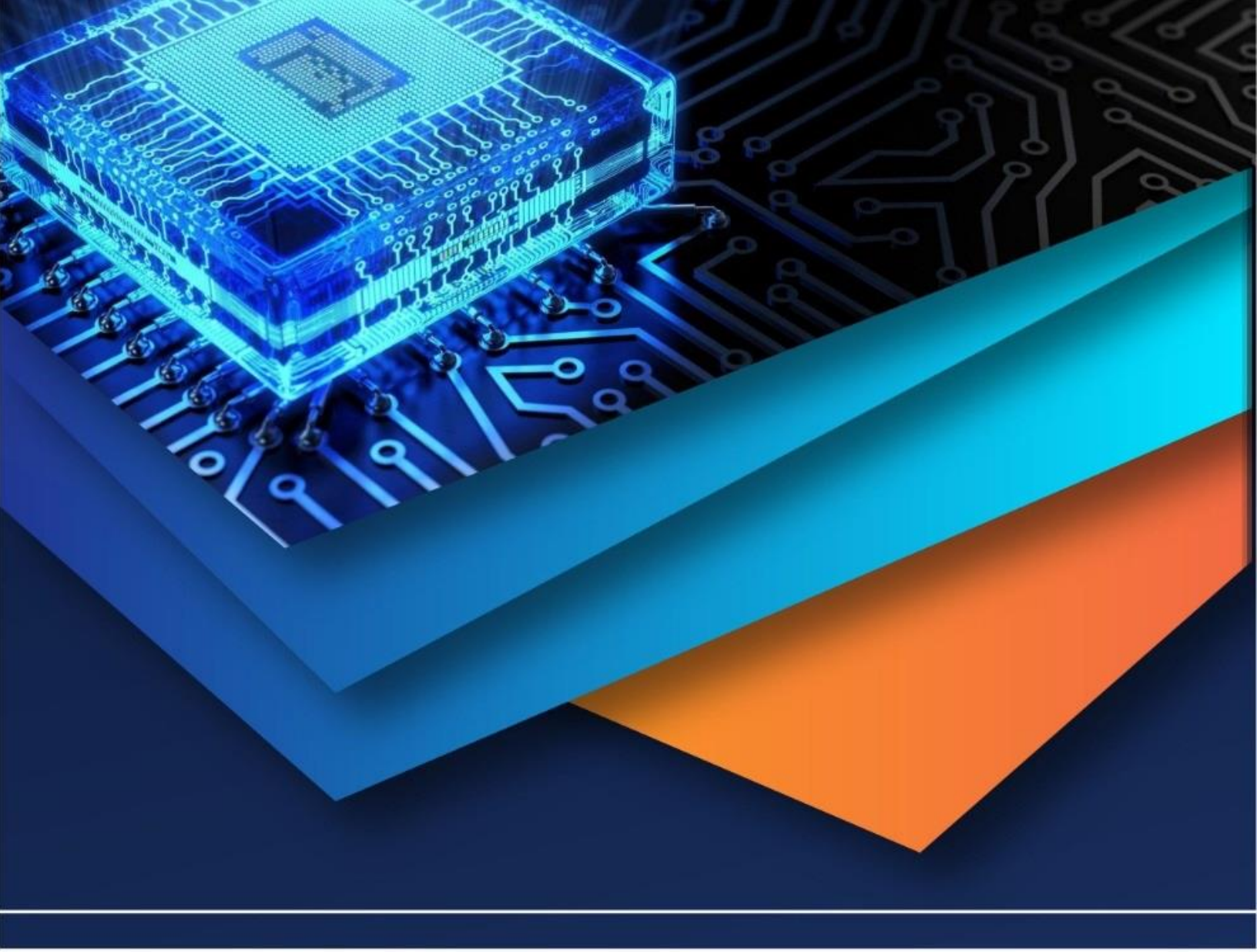
State Management: Maintaining the stateful environment of libraries like PyCaret across different agent tasks requires careful design (passing experiment objects, re-running setup, or using encapsulated tools). Data Handoff Efficiency, Passing large datasets or complex model objects between agents can be inefficient; strategies like serialization or passing references/paths are necessary. Debugging Understanding and debugging the interactions and decision-making within a multi-agent system can be complex, necessitating robust logging and observability.

X. ACKNOWLEDGEMENT

In this conference paper, We would like to express our profound appreciation to all those who contributed to the smooth development and implementation of the automated machine learning pipeline using agent-based architecture in this campaign. We extend our sincere gratitude to our academic mentors and technical advisors for their ongoing support, insightful feedback, and guidance throughout the research and development process. We also acknowledge the developers and contributors of the open-source applications and frameworks that served as the foundation for our development, such as CrewAI, LangChain, Streamlit, and PyCaret. We are also grateful to our colleagues and evaluators for their insightful comments and recommendations that enhanced the overall quality and impact of this project.

REFERENCES

[1] Wang, Zhaozhi, et al. (2023): "Multi-Agent Automated Machine Learning." CVPR 2023.
[2] Trirat, Patara, et al. (2024): "AutoML-Agent: A Multi-Agent LLM Framework for Full-Pipeline AutoML." arXiv preprint arXiv:2410.02958.
[3] Chi, Yizhou, et al. (2024): "SELA: Tree-Search Enhanced LLM Agents for Automated Machine Learning." arXiv preprint arXiv:2410.17238.
[4] Fatouros, George, et al. (2025): "Towards Conversational AI for Human-Machine Collaborative MLOps." arXiv
[5] preprint arXiv:2504.12477.
[6] Heffetz, Yuval, et al. (2019): "Deep Line: Auto ML Tool for Pipelines Generation using Deep Reinforcement Learning and Hierarchical Actions Filtering." arXiv preprint arXiv:1911.00061.
[7] Karmaker, Kanti, et al. (2023): "Automating the Machine Learning Process using PyCaret and Streamlit." ResearchGate.
[8] Ali, Moez (2021): "Write and Train Your Own Custom Machine Learning Models Using PyCaret." Medium.
[9] Ali, Moez (2021): "Build and Deploy ML App with PyCaret and Streamlit." PyCaret Documentation.
[10] Ali, Moez (2021): "Deploy Machine Learning App Built Using Streamlit and PyCaret on Google Kubernetes Engine." Medium.
[11] PyCaret Team (2021): "Deploy PyCaret and Streamlit on AWS Fargate." PyCaret Documentation.
[12] Oracle AI Team (2024): "AutoML-Agent: Pioneering Full-Pipeline Automation for Vertical AI Business Ecosystems." Medium.
[13] Robyn Le Sueur (2024): "Building Simple User Interfaces for CrewAI with Streamlit." LinkedIn.
[14] Folch, Albert (2025): "Introducing My First Streamlit and CrewAI Project!" Streamlit Community Forum.
[15] Analytics Vidhya (2021): "Build Web App Instantly for Machine Learning Using Streamlit." Analytics Vidhya Blog.
[16] Wikipedia Contributors (2025): "Agentic AI." Wikipedia.
[17] Vation Ventures (2024): "Artificial Intelligence Agents: Architecture & Applications." Vation Ventures Research Article.
[18] Microsoft Azure (2025): "AI Architecture Design - Azure Architecture Center." Microsoft Learn.
[19] Google Cloud (2024): "MLOps: Continuous Delivery and Automation Pipelines in Machine Learning." Google Cloud Architecture Center.
[20] ScienceDirect (2023): "AutoML: A Systematic Review on Automated Machine Learning with a Look into the Future of Evolutionary Approaches." ScienceDirect.
[21] SpringerLink (2024): "Automated Machine Learning: Past, Present and Future." SpringerLink.



10.22214/IJRASET

45.98

IMPACT FACTOR: 7.129

IMPACT FACTOR: 7.429

INTERNATIONAL JOURNAL FOR RESEARCH

IN APPLIED SCIENCE & ENGINEERING TECHNOLOGY