

# Meta AI: An intelligent system for automated machine learning pipeline generation.

Sujith A C<sup>1</sup>, Shyamal S<sup>2</sup>, Rohith Kumar A<sup>3</sup>

Student, Dept. of Computer Science & Engineering Artificial Intelligence & Data Science, The Apollo University, Chittoor, Andhra Pradesh, India

\*\*\*

**Abstract** - Meta AI is a smart system that automates the entire machine learning pipeline for tabular data. The growing complexity of machine learning workflows needs a lot of expertise, time, and manual work, making it hard for beginners and causing inefficiencies for experienced users. This work presents a unified platform that makes the process easier by combining data ingestion, preprocessing, feature engineering, model training, evaluation, and deployment into one interactive environment. The system is designed as a local-first application, which lets users upload datasets, choose target variables, and run complete workflows without needing deep coding skills. The platform uses methods like hyperparameter optimization, ensemble learning, and model explainability to improve performance and transparency. It also includes tools for data validation, bias detection, drift analysis, and experiment tracking, ensuring reliability and reproducibility of results. A user-friendly dashboard offers easy access to each step of the pipeline, giving users flexibility and control. Experimental tests show that the system shortens development time while keeping competitive model performance across various datasets. Additionally, it can produce deployment-ready artifacts, including application programming interfaces, for real-world use. This system effectively connects raw data to deployable machine learning solutions, making it useful for both academic and industrial purposes.

**Key Words:** Automated Machine Learning, Pipeline Automation, Optuna Optimization, SHAP Explainability, Ensemble Learning, Data Preprocessing, MLOps, Predictive Modeling

## 1. INTRODUCTION

Machine learning is an important technology these days. It helps systems learn from data and make guesses. To make a good machine learning system you have to do a lot of things. You have to get the data ready pick the features choose a model train it, test it and then put it to use. All these steps take a lot of knowledge, time and hard work. This makes it hard and confusing for people who are just starting out. Usually people use a lot of tools and systems to do all these things. This can cause problems and mistakes. Also because all these steps are not well connected it is hard to get the results twice and to make the system work on a large scale.

So there is a need for systems that can make the whole machine learning process easier and faster.

This paper is about Meta AI. It is a system that can do the whole machine learning process for data that is in tables. It gives users one place to upload their data say what they want to predict and do everything they need to do all in one interface. Meta AI uses techniques like automatically finding the best features picking the best settings combining multiple models and explaining how it works. This makes it work better and easier to understand. Also it can create code that other systems can use, which makes it good for world applications. Meta AI is a machine learning system that can really help people, with machine learning tasks.

## 1.1 Problem Statement

Building machine learning models involves several interconnected steps that require domain knowledge and technical expertise. Manual handling of data preprocessing, feature engineering, and model tuning often leads to inefficiencies and increased development time. Moreover, the lack of standardized workflows can result in poor reproducibility and difficulty in scaling solutions.

There is a need for an automated system that simplifies these processes while maintaining accuracy, transparency, and flexibility. The system should enable users to efficiently move from raw data to deployable models without requiring extensive coding or deep expertise in machine learning.

## 1.2 Objectives

The main objectives of the proposed system are as follows:

- To automate the complete machine learning pipeline from data ingestion to deployment
- To provide a user friendly and interactive interface for seamless workflow execution
- To integrate advanced techniques such as hyperparameter optimization and ensemble learning
- To ensure model interpretability using explainable AI methods
- To support real world deployment through API generation and export functionality

## 2. RELATED WORK

AutoML research has evolved significantly over the past decade. Auto-sklearn [4] introduced meta-learning and Bayesian optimization for joint algorithm selection and hyperparameter optimization (CASH). TPOT [6] applied genetic programming to evolve ML pipelines automatically. H2O AutoML [5] popularized stacking ensembles and provided a REST-based deployment interface, making it accessible to enterprise users.

More recently, frameworks such as AutoGluon [7] and FLAML [8] have advanced the state of the art in ensemble construction and cost-effective tuning. However, these systems primarily optimize predictive performance and do not address interpretability, fairness, or experiment auditability as first-class concerns. The integration of large language models into ML workflows is an emerging area: tools like LangChain and similar agent frameworks have been explored for code generation and data analysis [9], but their integration into a complete AutoML pipeline remains nascent.

## 3. SYSTEM ARCHITECTURE

Meta AI is organized as a layered architecture comprising a Gradio-based presentation layer, a core ML pipeline layer, and an optional services layer. The entry point is `quick_start.py`, which loads the dashboard defined in `dashboard_v3.py` and binds the Gradio application to an available local port (default: 7860). An optional FastAPI backend (`backend_api_main.py`) supports API-centric workflows and container deployments.

The pipeline is decomposed into eight functional modules, each exposed as a tab group within the Gradio dashboard. Table 1 summarizes the modules and their capabilities.

**Table 1: Meta AI System Modules and Capabilities**

Module	Key Capabilities
<b>Data Ingestion</b>	CSV upload, semantic type detection, Pydantic schema validation, drift baseline capture, lineage tracking
<b>Data Reconstruction</b>	Systematic bias detection, MICE-style Bayesian imputation, Isolation Forest outlier scoring
<b>EDA</b>	Correlation-driven hypothesis generation, UMAP/t-SNE 2D/3D projections, AI cluster explanation
<b>Feature Engineering</b>	Agentic domain feature creation, RFE-based selection, polynomial/datetime/encoding features

<b>Model Training</b>	Multi-model baseline, Optuna hyperparameter search, stacking ensemble with meta-learner
<b>Analysis &amp; XAI</b>	Task-specific metrics, global/local SHAP views, group-wise fairness and bias audit
<b>Agentic Auditing</b>	LLM-assisted post-mortem, counterfactual reasoning, free-form agent Q&A
<b>MLOps &amp; Export</b>	Readiness checks, FastAPI + Dockerfile generation, MLflow tracking, production ZIP export

The modular design ensures that each stage can be executed independently or as part of a sequential workflow. State is maintained in memory within a single session of `quick_start.py`; export operations require training and export to occur within the same session to ensure consistency between in-memory feature metadata and serialized model artifacts.

### 3.1 Data Ingestion and Validation

The ingestion module accepts UTF-8 CSV files through the Gradio interface. Upon upload, the system performs: (i) target column selection with automatic task inference (classification vs. regression); (ii) column-level semantic type detection with confidence reporting; (iii) Pydantic-based schema generation and row-level validation; (iv) statistical drift baseline capture for subsequent drift comparison; and (v) lineage visualization with pipeline-stage metrics. These steps ensure that downstream stages operate on well-understood, validated data.

### 3.2 Data Reconstruction

Missing data handling is addressed through a three-component reconstruction module. Systematic bias detection identifies missing-data patterns that may introduce systematic errors into model training. Bayesian reconstruction applies a MICE-style (Multiple Imputation by Chained Equations) iterative imputation strategy to recover missing values while preserving statistical relationships. Outlier detection employs Isolation Forest scoring with configurable contamination thresholds, enabling practitioners to identify and handle anomalous observations before feature engineering.

### 3.3 Exploratory Data Analysis

The EDA module provides two analytical views. The hypothesis generation sub-tab derives correlation-driven hypotheses from the dataset, presenting heatmaps and summary tables that guide feature engineering decisions.

The dimensionality reduction sub-tab applies UMAP or t-SNE projections to generate 2D and 3D visualizations of the feature space, with optional AI-driven cluster explanation controls that leverage the LLM backend to interpret cluster semantics.

### 3.4 Feature Engineering

Feature engineering is supported through three complementary mechanisms: (i) agentic feature creation, which applies domain-style derived features informed by semantic column analysis; (ii) Recursive Feature Elimination (RFE) with ranking outputs for feature selection; and (iii) automated feature engineering encompassing polynomial features, binning, datetime decomposition, and categorical encodings. The combination of manual, agent-guided, and automated approaches provides flexibility across datasets of varying complexity.

### 3.5 Model Training

The model training module supports three training strategies. Normal training performs multi-model baseline comparison using default hyperparameters, providing a fast initial performance estimate. Optuna-based optimization conducts trial-driven hyperparameter search with a configurable trial budget, leveraging Tree-structured Parzen Estimator (TPE) sampling to efficiently explore the hyperparameter space [10]. Stacking ensemble construction combines multiple base estimators with a meta-learner, typically yielding improved generalization over single-model approaches [11]. All training runs log metrics, parameters, and artifacts to MLflow under the Meta-AI-Models experiment when tracking is enabled.

### 3.6 Analysis and Explainability

Post-training analysis encompasses task-appropriate performance metrics (accuracy, F1, AUC-ROC for classification; RMSE, MAE,  $R^2$  for regression), visualization plots, and result tables. SHAP (SHapley Additive exPlanations) [12] is used to generate global feature importance summaries and local per-sample explanations, supporting both model debugging and regulatory compliance requirements. The fairness and bias audit sub-tab computes group-wise metrics across protected attributes, providing visualizations that enable practitioners to assess disparate impact before deployment.

### 3.7 Agentic Auditing

The agentic auditing module represents a novel contribution of Meta AI. Three capabilities are provided: (i) scientific post-mortem, which uses an LLM to generate a structured narrative summary of the experiment covering data quality, model performance, and key findings; (ii) counterfactual

reasoning, which suggests target-accuracy-oriented improvements by reasoning over the experimental context; and (iii) a free-form agent Q&A interface that allows practitioners to query the experiment in natural language. This module is powered by a configurable LLM backend (e.g., OpenAI) supplied via environment variables, preserving the local-first design for core pipeline operations.

### 3.8 MLOps and Production Export

The MLOps module provides a readiness checklist and visualization, API code generation (FastAPI endpoint, Dockerfile, pinned requirements), drift detection comparing training versus current data distributions, a simulated production monitoring dashboard, and MLflow tracking status. The export function packages the serialized model, feature schema, generated api.py, Dockerfile material, and requirements into a deployment-ready ZIP archive. Users can unzip this bundle into a clean environment and launch an inference server with a single uvicorn command.

## 4. IMPLEMENTATION

Meta AI is implemented in Python 3.10+ and relies on a set of well-established scientific and ML libraries. The core dependency stack includes Gradio for the interactive UI, scikit-learn for model implementations and feature engineering, pandas and NumPy for data manipulation, Optuna for hyperparameter optimization, SHAP for explainability, FastAPI and Uvicorn for the optional API backend, and MLflow for experiment tracking. The full dependency list is maintained in requirements.txt.

The repository follows a modular layout: core/ contains ingestion, training, export, and supporting ML logic; mlops/ houses tracking and operational helpers; llm/, agents/, and chatbot/ encapsulate optional language-model and agent-related code paths; and tests/ provides a pytest collection including export hardening tests (tests/test\_export\_hardening.py). Code quality is enforced through Ruff (linting) and Bandit (security scanning), configured in pyproject.toml. CI/CD pipelines (GitHub Actions) run tests on Python 3.10 and 3.11 and validate Dockerfile. \prod builds.

Cross-platform support is ensured through Windows batch scripts (RUN\_GRADIO\_APP.bat, scripts/setup.bat) alongside standard POSIX shell instructions. An optional static React dashboard (frontend/react-dashboard) can be served locally for preview purposes.

## 5. COMPARATIVE EVALUATION

Table 2 presents a feature-level comparison of Meta AI against three widely used AutoML frameworks: Auto-sklearn, H2O AutoML, and TPOT. The comparison focuses on capabilities most relevant to end-to-end ML lifecycle support.

**Table 2: Feature Comparison of Meta AI vs. Existing AutoML Frameworks**

Feature	Meta AI	AutoSklearn	H2O AutoML	TPOT
LLM Agentic Audit	Yes	No	No	No
Gradio-based UI	Yes	No	Yes	No
SHAP Explainability	Yes	Partial	Yes	No
Fairness Audit	Yes	No	Partial	No
Optuna HPO	Yes	No	Yes	No
Stacking Ensemble	Yes	Yes	Yes	Yes
Production Export	Yes	No	Yes	No
MLflow Integration	Yes	No	Yes	No
Local-First	Yes	Yes	No	Yes

As shown in Table 2, Meta AI is the only framework among those compared that provides LLM-assisted agentic auditing, integrated fairness auditing, and structured production export within a single local-first environment. While H2O AutoML offers a Gradio-adjacent web UI and MLflow-compatible logging, it requires cloud infrastructure for full functionality. Auto-sklearn and TPOT lack UI-based deployment support and explainability integration. Meta AI's combination of these capabilities within a workstation-scale, locally-executable tool represents its primary differentiating contribution.

## 6. TYPICAL WORKFLOW

A standard Meta AI session proceeds through the following stages: (1) Data Upload — the user uploads a UTF-8 CSV file through the Data Ingestion tab and selects the prediction target column; (2) Validation and Baseline — semantic detection, schema validation, and drift baseline capture are executed; (3) Reconstruction — missing data imputation and outlier handling are applied as required; (4) EDA and Feature Engineering — correlation hypotheses are generated, dimensionality projections are visualized, and

features are engineered using the automated and agentic tools; (5) Training — the user selects a training strategy (baseline, Optuna, or stacking) and initiates model training; (6) Analysis — performance metrics, SHAP explanations, and fairness audit results are reviewed; (7) Agentic Audit — the LLM post-mortem and counterfactual reasoning tools are used to derive insights and improvement directions; and (8) Export — a deployment ZIP is generated within the same session, completing the end-to-end workflow.

This structured workflow ensures reproducibility and traceability at each stage while allowing practitioners to customize the depth of engagement with individual modules based on their requirements.

## 7. CONCLUSIONS

This paper presented Meta AI, an end-to-end automated machine learning system for tabular data that integrates data ingestion, reconstruction, EDA, feature engineering, model training, explainability, agentic auditing, and production export within a unified Gradio-based dashboard. Meta AI addresses critical gaps in existing AutoML frameworks by combining LLM-assisted agentic intelligence, SHAP-based explainability, group-wise fairness auditing, and structured deployment support in a local-first, workstation-scale environment.

The system's modular architecture supports flexible, iterative workflows and is designed for reproducibility through MLflow tracking and deterministic export bundles. Comparative analysis demonstrates that Meta AI uniquely covers the full ML lifecycle, from raw CSV data to a deployable FastAPI inference server, without requiring cloud infrastructure.

Future work will focus on extending support to semi-structured and time-series datasets, incorporating active learning capabilities within the agentic auditing module, and evaluating system performance across standardized AutoML benchmark suites such as AutoML Benchmark [13] and OpenML-CC18 [14].

## ACKNOWLEDGEMENT

The authors would like to thank the open-source communities behind Gradio, scikit-learn, Optuna, SHAP, MLflow, and FastAPI, whose foundational work made this system possible.

## REFERENCES

- [1] J. Waring, C. Lindvall, and R. Umeton, "Automated Machine Learning: Review of the State-of-the-Art and Opportunities for Healthcare," *Artificial Intelligence in Medicine*, vol. 104, 2020. M. Young, *The Technical Writer's Handbook*. Mill Valley, CA: University Science, 1989.
- [2] F. Hutter, L. Kotthoff, and J. Vanschoren, *Automated Machine Learning: Methods, Systems, Challenges*. Springer, 2019.
- [3] M. Feurer and F. Hutter, "Hyperparameter Optimization," in *Automated Machine Learning*, Springer, 2019, pp. 3–33.
- [4] M. Feurer et al., "Efficient and Robust Automated Machine Learning," *Advances in Neural Information Processing Systems*, vol. 28, 2015.
- [5] H2O.ai, "H2O AutoML: Scalable Automatic Machine Learning," 7th ICML Workshop on Automated Machine Learning, 2020.
- [6] R. Olson et al., "TPOT: A Tree-based Pipeline Optimization Tool for Automating Machine Learning," *Proc. Workshop on Automatic Machine Learning, JMLR*, 2016, pp. 66–74.
- [7] N. Erickson et al., "AutoGluon-Tabular: Robust and Accurate AutoML for Structured Data," arXiv:2003.06505, 2020.
- [8] C. Wang et al., "FLAML: A Fast and Lightweight AutoML Library," *Proc. MLSys Conference*, 2021.
- [9] T. Guo et al., "DS-1000: A Reliable and Practical Benchmark for Data Science Code Generation," arXiv:2211.11501, 2022.
- [10] T. Akiba et al., "Optuna: A Next-generation Hyperparameter Optimization Framework," *Proc. ACM KDD*, 2019, pp. 2623–2631.