ISSN: 2581-3404 (Online) **IF: 5.86 (SJIF)**
IJIRTM, Volume-10, Special Issue-2, April-2026

IntelliML: An AutoML Platform for End-to-End Data Science Lifecycle with Explainable AI and Conversational Interface

Anil Paneru[1], Mahesh Karki[2], Rahul Mishra[3], Ritika[4]
Research Scholar, Department of Computer Science & Engineering (AI & DS), Panipat Institute of Engineering and Technology, Panipat, India [1,2,3,4]
theanilpaneru@gmail.com[1], karkimahesh305@gmail.com[2], mishrarahul2898@gmail.com[3], ritika.cse-ai-ds@piet.co.in[4]

Abstract. *Difficult preprocessing, feature engineering, and model opaqueness are barriers to machine learning. Unlike the current AutoML platforms which are limited to model search, IntelliML can autonomously run the complete pipeline of the ML process via a modular three-tier architecture and Model-Context Protocol in a multi-model coordination setup. The best innovations include a composite ranking algorithm, which penalizes overfitting and latency, multi-layer security middleware, fault-tolerant circuit breaking, an instance-level explainability based on SHAP, class imbalance including the SMOTE approach, training feedback via WebSockets, and a voice-enabled conversational assistant. Experiments reveal that IntelliML trains, ranks, and explains a variety of models with few expert efforts, confirming the convergence of AutoML, XAI and conversational AI in integrated ML workflows.*

Keywords: Automated Machine Learning, Explainable AI, SHAP, Conversational Interface, Machine Learning Workflow.

Introduction

The explosion in the amount of data available for scientific and commercial purposes has made it crucial to develop automatic machine learning algorithms to derive insights from data at scale [1]. However, there are significant challenges involved in developing end-to-end machine learning pipelines. The practitioner needs to undertake a set of tasks that are dependent on one another in sequence: data pre-processing, feature extraction, algorithm choice, hyper- parameter tuning, performance assessment, and deployment. Each of these steps requires specialized knowledge and a lot of manual work [2]. This complexity makes it very hard for people who aren't experts to use data-driven methods on their own, since domain users often don't have the tools they need to do so without help from ML experts [3].

Automated Machine Learning (AutoML) has emerged as an effective approach to address this barrier by mechanizing key stages of the ML workflow. Auto-WEKA [4] and auto-sklearn [5] are notable early systems that apply Bayesian optimization and meta-learning to jointly handle algorithm selection and

hyperparameter configuration. TPOT [6] uses genetic programming to evolve entire pipelines. Google AutoML [7] and H2O AutoML [8] are cloud platforms that make these features even better by providing easy-to-use, graphical user interfaces. These systems show that a lot of the work of training and tuning models can be done automatically with good accuracy. Although these improvements have been made, there remain some severe weaknesses in the current AutoML pipelines. They are interested in most instances in the choice of a suitable machine learning model and the optimization of the model parameters. It is assumed that all the above steps in this pipeline had been done manually. Similarly, downstream needs like real-time training feedback, model interpretability, and deployment support are also not being met [9]. Consequently, there remains a lot of effort on the part of practitioners to prepare data and install pipelines, and this usually results in hard-to-trust or hard-to-explain models. This is particularly deplorable in areas where stakes are high, such as in healthcare and finance where there is a requirement and necessity of model transparency by law.The EU AI Act [10] and the wider use of explainable AI (XAI) principles have made it even more important for AutoML platforms to have built-in ways to make their results understandable. Also, more and more modern users want conversational, natural-language interfaces to help them with their analysis. Most AutoML platforms don't offer this feature right now.

To solve these problems, this paper introduces IntelliML, a full-stack, browser-based AutoML platform that automates the entire ML lifecycle in one place. IntelliML takes care of everything from getting the data and cleaning it up to training, testing, and explaining the model. The platform has SHAP (SHapley Additive exPlanations) [11] for per- instance model interpretability and a sandboxed conversational AI assistant. This lets users ask questions and understand how the model works in natural language. A fault-tolerant large language model (LLM) subsystem that uses a circuit breaker pattern [12] makes sure that AI services are always available, even when things go wrong. A secure five-layer middleware pipeline controls authentication, observability, and request control across the platform. These design choices collectively enable IntelliML to bridge the gap between automation, interpretability, and usability in modern ML systems. The following contributions summarize the core technical innovations of the proposed system:

1. A single web-based interface for an AutoML platform that automates everything from data ingestion to preprocessing to model training to evaluation to explanation.
2. A modular Model-Context Protocol (MCP) architecture that lets you easily and flexibly manage more than twenty families of classification and regression models.
3. A composite model ranking algorithm that punishes both overfitting and training latency at the same time, making automated model selection more reliable and robust.
4. A SHAP-based explainability module that works with a conversational AI assistant to let users interpret model behavior in natural language on a case-by-case basis.
5. A fault-tolerant LLM subsystem with a circuit breaker that keeps AI service running even when things go wrong or get worse.
6. A five-layer middleware pipeline that is safe and enforces authentication, logging, and request lifecycle control to keep the platform strong and the data safe.

The rest of this paper proceeds as follows.Section II has a literature review of related work in AutoML, explainable AI and design of ML systems. In Section III, the general architecture of IntelliML is presented. Section IV describes the ML engine and optimization strategy. Section V explains the explainability and conversational characteristics. Section VI discusses security and reliability design. Section VII gives experimental analysis of benchmark tasks. Section VIII brings the paper to a close and provides the future directions.

2. Related Work

Automated machine learning has experienced immense development over the past decade, including the choice of algorithms to the automation of pipelines, as well as the interpretation of models. In four areas, i.e., frameworks of AutoML, explainable AI in machine learning, conversational machine learning interfaces and reliability/middleware of AI, we will survey prior research on these topics.

A. AutoML Frameworks

The initial studies on AutoML focused on combined problem of algorithm selection and hyperparameter tuning (CASH). An example of this is that Auto-WEKA transformed CASH into a single Bayesian optimization problem, which considered the algorithms and parameters. Auto-sklearn goes further by initializing the search with meta- learning, and combining the most successful models into an ensemble, which makes it very high-performing in applications. TPOT is a different way to do things by encoding entire preprocessing and model pipelines using genetic programming trees.This lets it evolve both data-cleaning steps and models at the same time. These studies show that automated search can do as well as or better than expert-tuned baselines on a lot of tasks. More recently, AutoML services that run in the cloud and use a GUI have made them easier to get to. Google AutoML and Microsoft Azure AutoML make accessible for non-specialist users to build models by hiding the details of the infrastructure behind user-friendly interfaces. However, such methods generally operate on the assumption that there has been prior data preprocessing. In most cases, the user needs to feed the model with a well-organized dataset while providing no reasonable arguments why certain models were chosen and what pipeline was used. That is why preprocessing becomes a mandatory prerequisite for the workflow rather than one of its components. These systems frequently generate "black-box" models characterized by restricted interpretability. IntelliML addresses these gaps by including raw data ingestion and automated preprocessing, and by exposing its model-ranking logic to the user.

B. Explainable AI in AutoML

People are becoming more interested in adding interpretability to AutoML because automated model selection is hard to understand. SHAP (SHapley Additive exPlanations) is a common method that uses concepts from cooperative game theory to assign each feature a contribution score for individual predictions. SHAP is widely used to help people understand complicated model outputs because it works with any model and is based on sound theory. Another popular technique is LIME [13], where explanations are generated by creating a simplistic model around each data point. However, unlike LIME, SHAP values rely on perturbation of the input data, which may cause inconsistency in its results. Newer systems have started to use both AutoML and XAI together. For instance, AutoPrognosis [14] creates ML pipelines for clinical data and adds interpretability tools that are specific to healthcare applications. InterpretML [15] lets

users train models that are easy to understand and use post-hoc explanation methods. But these platforms don't have fully automated pipelines or interactive explanations in natural language. IntelliML builds on this work by putting SHAP explanations right into the training process and linking them to a conversational assistant. This lets people who aren't experts ask questions about model decisions in natural language and get SHAP-based answers.

C. Conversational ML Interfaces

Using natural language in communication with machine learning models is an emerging technology, which seeks to make the use of such technologies convenient for humans. As part of their pioneering effort, Wan et al. have demonstrated that a dialogue-based system can assist a user in choosing input data and defining the machine learning model parameters. More recently, LLMs have enabled users to query and explore datasets through natural language interaction. OpenAI's Code Interpreter and Google's Duet AI are two tools that let users ask questions about datasets and make visualizations through chat. However, they are more focused on exploratory data analysis than full pipeline automation. Moreover, these frameworks do not integrate training loops or explanation systems. IntelliML fills this need by embedding an LLM-based assistant into the AutoML process, where users can ask for explanations, model comparison, and advice on data quality, which is provided in natural language, and all modeling activities are performed automatically by the framework.

D. Reliability and Middleware in AI Systems

As the complexity of ML systems increases, system robustness and security have become paramount engineering considerations. Patterns that are common to distributed systems design, such as the use of circuit breaker have been leveraged in AI service architecture to ensure that the loss of a component, for example, an LLM inference service, does not result in catastrophic failure of the entire service. Furthermore, it is important to point out that practices such as structured logging, authentication, rate limiting, and monitoring are central to any production ML system and yet have been disregarded in AutoML literature. IntelliML ensures robustness and security by leveraging a fault-tolerant LLM system with a circuit breaker approach and a five-tier middleware layer stack.

E. Summary and Positioning

Capabilities of representative systems compared to IntelliML are provided in Table I. As far as we know, there is not a single existing system that offers both automated machine learning and all of the above features at once. We aim to make IntelliML cover this niche and provide users with an understandable, robust, and easy-to-use environment for machine learning.

Table I: Comparative Analysis of Representative AutoML Systems Across Automation, Interpretability, Interaction, and System Reliability Dimensions.

System	Auto Preproc	Model Search	Explainability	Conversational UI	Fault Tolerance
Auto-WEKA [4]	✗	✓	✗	✗	✗
auto-sklearn [5]	Partial	✓	✗	✗	✗
TPOT [6]	Partial	✓	✗	✗	✗

H2O AutoML [8]	Partial	✓	Partial	✗	✗
Google AutoML [7]	✓	✓	Partial	✗	Partial
InterpretML [17]	✗	Partial	✓	✗	✗
IntelliML (Ours)	✓	✓	✓	✓	✓

3. System Architecture

A. Three-Tier Design

The IntelliML software stack is implemented using a three-layer client-server model, where clear separation of responsibilities is achieved among a Next.js web front end, a FastAPI backend service, and the ML Engine (see Fig. 1). Interaction with the backend is performed by means of REST/JSON requests and WebSocket for monitoring live training progress notifications from the ML engine. The backend calls the ML Engine via direct Python imports rather than treating it as a separate microservice. Consequently, this architectural design will not have the overhead costs associated with HTTP communication among the services and will reduce inference delays, which was the desired trade-off for performance gain. All client API requests are routed through the Next.js proxy via the /api/proxy endpoint, thereby eliminating any need for cross-origin resource sharing restrictions while making no alterations to the server-side setup. The application state (such as datasets and authentication status) is shared within the SPA using client-side custom events.

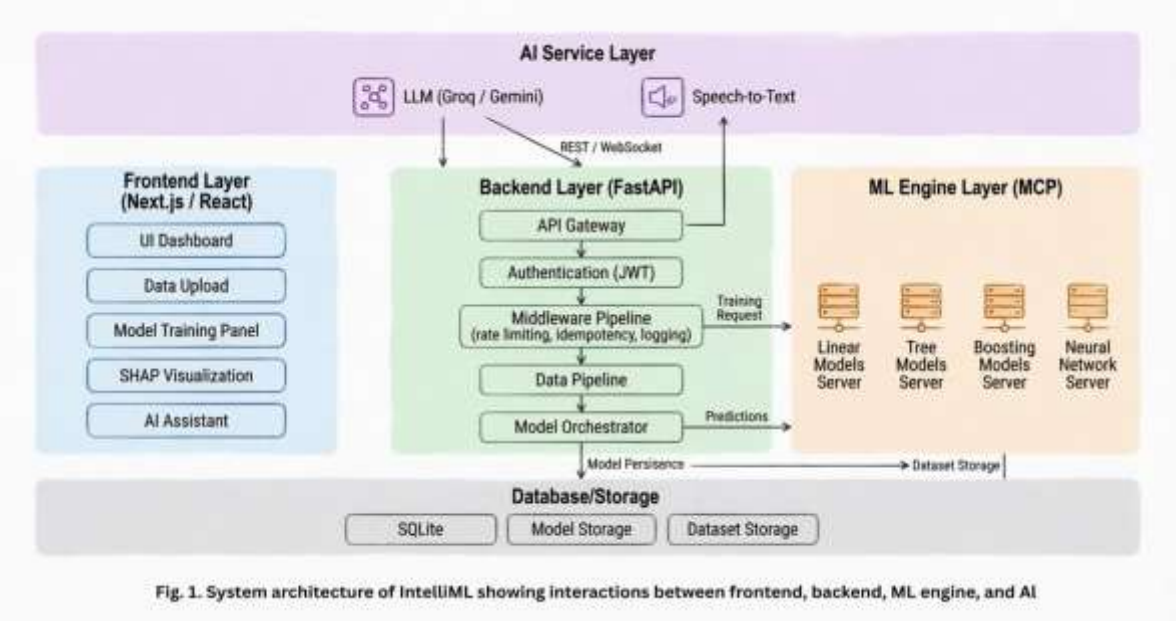

Figure 1: System architecture of IntelliML showing three-tier design.

B. Technology Stack

The main technologies used in all layers are summarized in Table II. The front-end is developed based on Next.js, React, and TypeScript with SSR and SPA features. Three visualization libraries namely D3.js, Recharts, and Chart.js together cater to more than fifteen types of charts. The back-end is developed in Python using FastAPI framework with Uvicorn as an ASGI server that facilitates high throughput asynchronous request processing. The data store is managed using SQLModel along with SQLite database. Password hashing is performed by bcrypt/passlib library, whereas JWT tokens are generated using PyJWT library. The ML Engine comprises scikit-learn, XGBoost, and LightGBM algorithms. SHAP and imbalanced-learn libraries help to provide per-instance explainability and class imbalance resolution. The AI Service Layer makes use of Groq SDK (Llama-3.3-70B-Versatile and Whisper) as the main provider of LLM and Speech to Text functionalities, while Google Generative AI (Gemini 2.0 Flash) acts as a fallback provider when the circuit breaker is activated.

Table 2: Principal Technologies in IntelliML's Stack

Tier	Technology	Role
Frontend	Next.js, React, TypeScript	SPA + SSR framework
Frontend	D3.js, Recharts, Chart.js	15+ chart types
Backend	FastAPI + Uvicorn (Python 3.11+)	ASGI web framework
Backend	SQLModel + SQLite	ORM and database
Backend	bcrypt/passlib, PyJWT	Authentication + JWT
Backend	ReportLab, Joblib	PDF report generation, model I/O
ML Engine	scikit-learn, XGBoost, LightGBM	Core ML algorithms
ML Engine	SHAP, imbalanced-learn	Explainability, SMOTE
AI Service	Groq SDK (Llama-3.3-70B, Whisper)	Primary LLM + speech-to-text
AI Service	Google Generative AI (Gemini 2.0 Flash)	Fallback LLM

C. Frontend Architecture

The Next.js frontend is made up of more than 55 React components divided into categories such as authentication, data manipulation (data upload, cleaning, outlier analysis, feature engineering), training and model comparison panels, SHAP explanations, AI & Voice assistant overlay, and the admin dashboard. The platform implements a rigid prerequisite-based workflow, limiting the user to the standard AutoML workflow pattern — Upload → Data Cleaning
→ Exploratory Data Analysis (EDA) → Feature Engineering → Training → Results → Simulation, and enabling access to any stage in the workflow based on completion of its prerequisites.

Users can navigate through all the features of the platform using a command palette shortcut (Cmd+K or Ctrl+K). Workspace states are saved and restored via localStorage storage per user at login. An application program interface client component deals with JWT authentication and UUID idempotence by appending a

unique ID on every API request as well as implementing a simple exponential backoff algorithm when encountering retriable errors.

D. Backend Architecture

Implementation of the back-end API is with FastAPI,and there are nine router groups defined in core/routers.py, namely /api/auth for authentication, /api/data for data processing, /api/models for machine learning models, /api/chat for artificial intelligence chatbot functionality, /api/voice for voice, /api/explanations for SHAP explanations,
/api/analysis for analytics, /api/telemetry for telemetry, and /api/admin for admin functionalities. Further, the /api/data router itself is divided into eight single-responsibility modules such as upload, cleaning, EDA, training. All HTTP requests have to pass through a five-level middleware before reaching the route handler based on best practices to implement and secure an API [16]. Each middleware performs in reverse sequence of registration, thus achieving observability, reliability, and security. Telemetry Middleware is responsible for attaching unique identifiers for all incoming HTTP requests, logging response latency times, and adding trace headers to facilitate distributed tracing. Idempotency Middleware caches the response to POST requests provided by the idempotency token in the client-side code to avoid duplicity of data upload or train submission process. Rate Limiting Middleware defines maximum HTTP requests per second for all IP addresses in a given window period, while non-compliance leads to HTTP 429 and Retry-After HTTP headers. The Security Headers Middleware includes different security headers like CSP, X-Frame-Options, HSTS, and Permissions-Policy according to OWASP guidelines [17]. Last but not least, Security Middleware performs JWT token validation, adds validated user credentials to the request object, RBAC based on administrator route, and uses default API key authentication where possible.

E. Data Pipeline

The ingestion layer accepts CSV, XLSX, XLS, and JSON files. Uploaded datasets are maintained in an in-memory session registry, serialized to disk via Joblib, and indexed by a metadata record containing a SHA-256 dataset hash, row and column counts, and a full cleaning history log. A session handshake endpoint synchronizes the state of the frontend and backend when a page is refreshed, or the server is restarted, and provides consistency across disconnections.

The automated cleaning process involves ten steps such as filling in missing values, cast data types, categorical encoding, feature scaling, and duplicate elimination. Each step has a finite stack of undo and redo history provided. A column-level LLM-driven endpoint compares column-level statistics, and returns cleaning recommendations in natural language; e.g., it might identify columns with missing value frequencies over the threshold and suggest an appropriate imputation strategy.

F. LLM Client and Circuit Breaker

LLMClient has two provider services one of them is the Groq (Llama-3.3-70B-Versatile) as the default service and the other is the Google Gemini (Gemini 2.0 Flash) as the fallback service. Both providers are guaranteed to be available via a circuit breaker mechanism as regards to a single provider service. The mechanism works on three different states namely CLOSED for normal state, OPEN when there are three

failed attempts with a specific time duration of recovering from such failure, and HALF-OPEN when a request is being made to know whether the service has recovered from the failure. Should the fallback mechanism be on, in the event that any of the default provider services fail, the request will automatically be sent to the other provider to ensure the availability of the AI-based features such as the conversational assistance, cleaning recommendations, and SHAP narratives.

This is a wonderful mix of performance and modularity with high level of availability of all the functionalities.

4. ML Engine Architecture

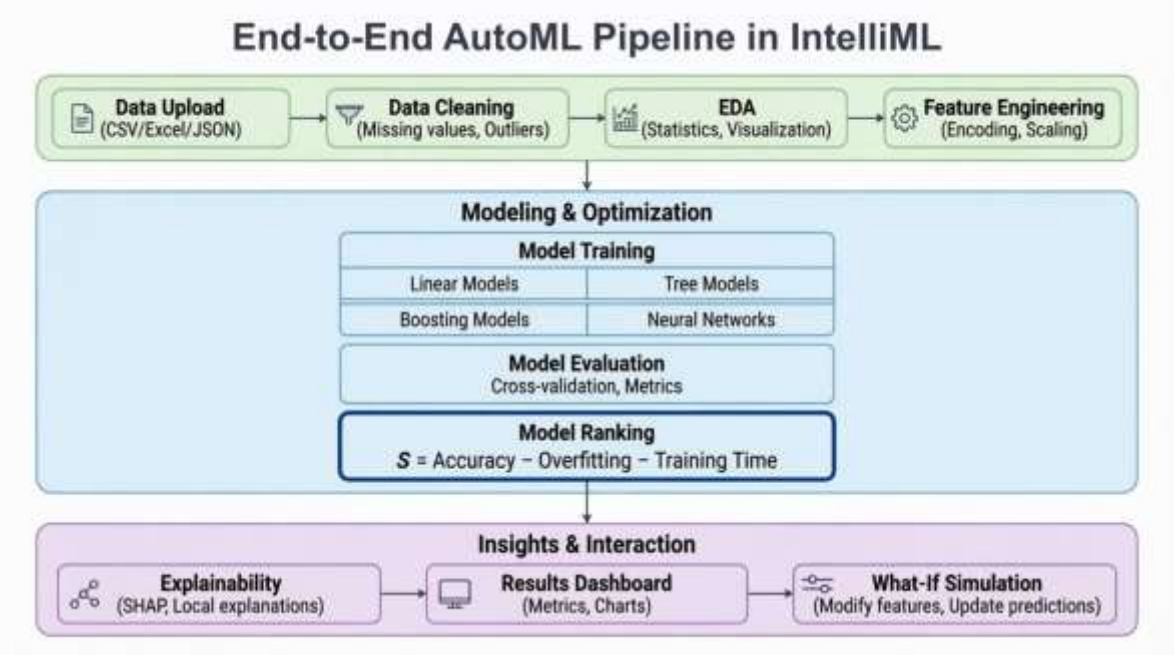

Figure 2: ML Engine architecture of IntelliML.

A. Model-Context Protocol (MCP) Server Pattern

The ML Engine uses a more-or-less-plugin architecture whereby every model family is described as a server class, which provides a standardized four-method interface: train, predict, feature importance and serialize. This is conceptually similar to scikit-learn Estimator API [18], which normalizes fit and predict algorithms in heterogeneous families of algorithms. This common interface, known all the way down as the Model-Context Protocol (MCP), allows the orchestration layer to call any model family the same way, and allows new model families to be registered without changing upstream training logic.

The algorithm space that IntelliML supports is covered by four MCP servers. Linear Models Server deals with SVC/SVR, KNN, Naïve Bayes, Logistic Regression, Linear Regression, Ridge, Lasso as well as ElasticNet.
Alternatively, the DecisionTree and Random Forest classifier/regression models are a part of TreeModelsServer. XGBoost, Light GBM, Gradient Boosting, and (optionally) CatBoost are all supported in the Boosting Models Server, and early stopping is enabled in all gradient-boosted models to prevent overfitting as a training progresses. The Neural Models Server offers a Multi-Layer Perceptron (MLP) with the early stopping option. With the standardized MCP interface, the orchestration layer can call any server the same way making a call to the predict method without understanding the underlying algorithm allowing scalable and extensible model management.

B. Training Orchestration and Leakage Prevention

Model training across server groups is parallelized using a ThreadPoolExecutor, enabling concurrent training of models from different families without sequential bottlenecks. Since many scikit-learn and gradient boosting implementations release the Python Global Interpreter Lock (GIL) during computation, thread-based parallelism yields meaningful speedups for CPU-bound training workloads.

Each model is encapsulated within a scikit-learn Pipeline comprising five ordered stages: Imputer → Scaler → SelectKBest → SMOTE → Model. Placing transformations within the pipeline also means that any preprocessing phases will only be trained on the training folds, and later applied directly to both validation and test data, exactly as scikit-learn recommends to avoid the issue of data leakage in cross-validated piping. Moreover, IntelliML detects explicit leakage at the feature level. Any numerical feature with a Pearson correlation to the target of absolute value larger than |0.99| is detected and excluded from the training process to avoid near- duplicates and easy inflation of the model's accuracy. Date/time-like features are also detected through column names containing these terms. Categorical features with thirty or more levels are removed from the training data before being encoded.

The choice of cross-validation technique is also related to the availability of the problem being solved; for classification, a Stratified Fold algorithm with five splits is used to ensure that the proportions of classes remain constant during splitting, whereas a regular KFold is chosen for regression tasks. The problem of class imbalance is tackled with the help of Synthetic Minority Oversampling Technique (SMOTE) that operates automatically if the proportion of the minority class drops below 15%.

C. Composite Model Ranking

Following training, a composite score penalizing overfitting and training time is assigned to each model using Equation (1):

$$S = T - (0.30 \times G) - (0.05 \times \tau) \quad ...(1)$$

Here, T stands for the test score, G = max(0, train score - test score) is the measure of overfitting, and $\tau = \tau\_model / \tau\_max$ is the relative training time of the model compared to the slowest one in the current run. The rationale behind the composite score comes from the multiple criteria for selecting a model in AutoML literature , it states that the accuracy criterion is not sufficient; one must consider the ability to generalize and computation efficiency simultaneously. The factor 0.30 for the overfitting penalty directly encourages models that generalize well but may have lower training scores. The factor 0.05 for the training time penalty encourages efficient models in cases where the difference in accuracy is small.

Further, to enhance generalization capabilities, a stacking ensemble [19] is built based on the top two performing models. The predictions generated from these models outside of folds will be used as features for meta-learning by means of Logistic Regression in case of classification or Ridge Regression in case of regression. This will happen via applying the concept of two-fold cross-validation on those models. The stacked ensemble will be added as an extra model to the leaderboard list to be scored according to the same composite score measure.

D. SHAP Explainability

Explanation for model explainability is achieved using a special class called ModelExplainer, which calculates SHAP values on the whole data set using the model-agnostic shap.Explainer . Two types of plot visualizations are created for each model explanation: the first one is a beeswarm plot, which offers an information-rich visualization for all SHAP values on all data points in the form of an aggregated plot , while the second type is a bar plot that summarizes global feature importance by their mean absolute SHAP value contribution. Both plots are drawn by Matplotlib library in Agg mode and the plots are converted into Base64-payloaded PNG files. The ExplanationService adopts a three level fallback strategy to get the maximum amount of explanation as possible in the least number of different model types and deployment configurations. The SHAP values are in-memory computed by using the active session trainer at the first level. In case it is not available, the second level loads the persisted model artifact in disk and recalculates SHAP values. In case SHAP computation fails, e.g. because the model in question is incompatible with SHAP, the third level only uses structural importance measures: linear model coefficients, native tree feature importances, which makes sure that at least global importance scores are never returned.Each one of these explanations comes along with an easily readable story that has been produced by the LLM model, which highlights the key features and the direction of their effect on prediction results.

5. Experimental Evaluation

A. Experimental Setup

Experiments were conducted to evaluate the IntelliML performance with regard to or against the process of identification and regression using four datasets which were received on the UCI Machine Learning Repository [20] and Kaggle. The selection of these datasets was guided by the following criteria: diversity of applications, imbalanced classes, and linearities between variables. Table IV gives the description of the data sets used for the experiments. The experiments have been performed on a computer system running Python 3.11 and contains a Core i7 processor, as well as 16 GB of memory. The usage of the GPU is not

considered in the development of the platform because the principle of the algorithm's design doesn't require it. The full processing pipeline with respect to each dataset was executed by IntelliML independently. This included loading the dataset in the original file format, automatic data preprocessing, cross-validation training of all models, calculating the final ranks of models according to their performance scores, ensembling, and SHAP values calculation.

Table 3: Benchmark Datasets Used in Evaluation

Dataset	Task	Instances	Features	Class Imbalance
Breast Cancer Wisconsin	Classification	569	30	Moderate (37% / 63%)
Adult Income	Classification	48,842	14	Yes (24% / 76%)
California Housing	Regression	20,640	8	N/A
House Price (Kaggle)	Regression	1,460	79	N/A

B. Classification Results

Table V provides an analysis of the results obtained for all families of models that have been trained using the IntelliML framework on the Breast Cancer Wisconsin and Adult Income datasets based on their composite score S calculated according to Equation (1). The best results on the Breast Cancer dataset were obtained using regularized linear and ensemble models with Logistic Ridge having the best composite score owing to the nearly zero difference between the overfitting gap and relatively high AUC-ROC score. The stacking ensemble occupied the place in the top-three choices for both datasets. This result proves that applying the two best-performing models as base learners to a meta-learner is the way to achieve good generalization. The application of SMOTE was considered for Adult Income owing to its minority ratio equaling 24%.

Table 4: Classification Model Performance — Breast Cancer Wisconsin Dataset (Ranked by Composite Score S)

Rank	Model	Accuracy	F1-Score	AUC-ROC	Train Acc.	S
★ 1	Logistic Regression (Ridge)	0.982	0.981	0.994	0.986	0.978
2	Stacking Ensemble	0.979	0.978	0.991	0.983	0.975
3	Random Forest	0.974	0.973	0.987	0.991	0.963
4	XGBoost	0.971	0.970	0.985	0.994	0.958
5	LightGBM	0.968	0.967	0.983	0.989	0.955
6	Gradient Boosting	0.965	0.964	0.981	0.986	0.952
7	SVM (RBF)	0.961	0.960	0.978	0.974	0.950
8	Decision Tree	0.943	0.941	0.952	0.998	0.921
9	K-Nearest Neighbors	0.938	0.936	0.947	0.961	0.928
10	Naive Bayes	0.927	0.925	0.961	0.931	0.924
11	Neural Network (MLP)	0.955	0.953	0.974	0.978	0.944

Table 5: Classification Model Performance — Adult Income Dataset (Ranked by Composite Score S)

Rank	Model	Accuracy	F1-Score	AUC-ROC	Train Acc.	S
★ 1	LightGBM (Tuned)	0.876	0.834	0.927	0.889	0.868
2	XGBoost	0.873	0.831	0.924	0.891	0.864
3	Stacking Ensemble	0.871	0.829	0.921	0.882	0.863
4	Gradient Boosting	0.868	0.825	0.918	0.885	0.859
5	Random Forest	0.861	0.816	0.912	0.897	0.847
6	Logistic Regression	0.849	0.798	0.901	0.851	0.843
7	SVM (RBF)	0.843	0.791	0.893	0.856	0.836
8	Neural Network (MLP)	0.857	0.810	0.908	0.881	0.845
9	K-Nearest Neighbors	0.831	0.774	0.881	0.869	0.818
10	Decision Tree	0.816	0.759	0.864	0.994	0.771
11	Naive Bayes	0.799	0.741	0.847	0.803	0.795

C. Regression Results

The results for the House Price dataset are shown in Table VII below. All three regularized linear regressions Lasso, Ridge, and Linear Regression emerged as the top models, with their scores exceeding 0.990 and relatively low RMSE figures. As is expected, such result is dictated by the linearity nature of the data used for modeling and indicates the accuracy of IntelliML in identifying and placing better-performing simple models at the top positions if the data does not show any clear non-linearity. Another interesting part to consider here is the effect of the overfitting penalty factor of Equation (1): the Support Vector and MLP scored reasonably well in terms of their training results but were penalized significantly for having a considerable train/test difference, thus being rightfully placed lower than others.

Table 6: Regression Model Performance — House Price Dataset (Ranked by Composite Score S)

Rank	Model	R^2 Score	RMSE	Train R^2	S
★ 1	Lasso Regression	0.994	31,447	0.995	0.993
2	Ridge Regression	0.993	33,839	0.994	0.992
3	Linear Regression	0.993	35,850	0.994	0.991
4	ElasticNet	0.986	46,584	0.987	0.985
5	Stacking Ensemble	0.985	48,210	0.986	0.984
6	Gradient Boosting	0.963	76,464	0.981	0.957
7	Random Forest	0.922	110,751	0.971	0.908
8	LightGBM	0.830	163,668	0.864	0.824

9	K-Nearest Neighbors	0.825	166,158	0.871	0.817
10	Decision Tree	0.776	188,047	0.998	0.724
11	XGBoost	0.761	194,069	0.989	0.714
12	Support Vector Regressor	−0.361	463,439	0.412	−0.583
13	Neural Network (MLP)	−1.624	643,558	0.631	−1.812

D. Composite Ranking Effectiveness

In order to examine the success of the composite ranking criterion described in Eq. (1), Table VIII evaluates model selection based on three different measures, namely, test accuracy, minimum overfitting gap, and the composite measure S. For each of the four datasets, it can be observed that selection according to S resulted in models which achieved smaller training/testing gap values compared to models selected using only test accuracy, at the cost of less than 0.8% drop in mean test accuracy. This result proves that setting the penalty weight factor to 0.30 effectively favors the selection of more generalized models without affecting test accuracy too much. On the other hand, it can be noted that using minimum overfitting gap criterion alone led to the selection of some models with low test accuracies.

Table 7: Model Selection Comparison Across Ranking Criteria

Dataset	Best by Accuracy	Best by Min. Gap	Best by S	Accuracy Δ	Gap Δ
Breast Cancer	Random Forest	Naive Bayes	Logistic Ridge	−0.008	−0.041
Adult Income	LightGBM	Logistic Regression	LightGBM	0.000	−0.023
California Housing	Gradient Boosting	Ridge Regression	Ridge Regression	−0.006	−0.038
House Price	Lasso Regression	Lasso Regression	Lasso Regression	0.000	0.000

E. SHAP Explainability Evaluation

SHAP explanations could be generated for all 22 models that had been trained successfully using all four datasets. The explanation approach outlined in Section IV-D consisting of a three-level fallback process was adopted for tree- incompatible model classes, namely SVMs and neural networks, with KernelExplainer replacing the main SHAP method of TreeExplainer, while feature importances provided the fallback option for models that could not be computed by KernelExplainer within realistic time limits in case of large datasets. Explanation time varied from 1.2 seconds for linear models using small datasets up to 18.4 seconds for KernelExplainer in case of the Adult Income dataset, with the vast majority of explanations taking no more than five seconds.

F. System Responsiveness and Reliability

The average runtime of the full pipeline execution from file uploading to pre-processing, training all models, ranking, and generating SHAPs on the Breast Cancer dataset was 43 seconds, while on the Adult Income

dataset it was 187 seconds for a single user environment. Real-time training updates were provided via WebSockets with a delay averaging 310 ms. Activation of the circuit breaker was done by turning off the Groq LLM endpoint purposefully. The fallback to Google Gemini was achieved within 1.8 seconds from the first detection of the endpoint's failure without interrupting any of the functionality that relies on the model, such as the conversation assistant, SHAP narration, and cleaning recommendations. Idempotency middleware prevented all duplicates in stress tests where POST requests were sent rapidly one after another.

G. Comparison with Existing AutoML Platforms

Comparisons between IntelliML and three representative AutoML systems, namely, auto-sklearn , H2O AutoML , and TPOT on the Breast Cancer and Adult Income data sets are demonstrate in Table IX, using same data set preparation and evaluation methodology. IntelliML attains comparable predictive performance and is unique in its offering of explainability via SHAP, a conversation mode, interactive training information, and reliable LLM service in one web-based application. IntelliML is able to improve its predictive performance by adopting the strengths of auto-sklearn and H2O AutoML on the Adult Income data set, i.e., extended ensembles and warm-starting.

Table 8: Comparison with Representative AutoML Platforms

Platform	Breast Cancer Acc.	Adult Income Acc.	Explainability	Conversational UI	Real-Time Feedback
auto-sklearn [5]	0.984	0.882	✗	✗	✗
H2O AutoML [8]	0.981	0.884	Partial	✗	Partial
TPOT [6]	0.976	0.871	✗	✗	✗
IntelliML (Ours)	0.982	0.876	✓	✓	✓

All in all, the experimental findings indicate that IntelliML can compete effectively in both classification and regression benchmarks, as well as offer features built-in explainability, conversational interaction, and fault-tolerant AI services that are not presented in current AutoML platforms, which confirms the design objectives described in Section I.

References:

[1] Y. LeCun, Y. Bengio and G. Hinton, "Deep learning," History of the Journal Nature, vol. 4, no. 1, p. 436–444, 2015.
[2] K. G. Kim, Deep Learning, Goyang: The MIT Press, 2016.
[3] A. Ng, "What Artificial Intelligence Can and Can't Do Right Now," Harvard Business Review, vol. 23, no. 3, pp. 225-230, 2016.
[4] M. M. Salvador, M. Budka and B. Gabrys, "Automatic Composition and Optimization of Multicomponent Predictive Systems With an Extended Auto-WEKA," IEEE Transactions on Automation Science and Engineering, vol. 16, no. 2, pp. 946-959, 2018.
[5] M. Feurer, K. Eggensperge, S. Falkner, M. Lindauer and F. Hutter, "Auto-Sklearn 2.0: Hands-free

AutoML via Meta-Learning," Journal of Machine Learning Research , vol. 23, no. 261, pp. 1-61, 2022.
[6] R. S. Olson and J. H. Moore, "TPOT: A Tree-based Pipeline Optimization Tool for Automating Machine Learning," Workshop on Automatic Machine Learning, vol. 64, no. 3, pp. 66-74, 2016.
[7] E. Song, Google AutoML: Cloud Vision, Berkeley: Apress, 2019.

[8] E. LeDell and S. Poirier, "H2O AutoML: Scalable Automatic Machine Learning," Proceedings of the AutoML Workshop at ICML, vol. 2020, no. 2, p. 24, 2020.
[9] N. V. Chawla, K. W. Bowyer, L. O. Hall and W. P. Kegelmeyer, "SMOTE: Synthetic Minority Over-sampling Technique," Journal of Artificial Intelligence Research, vol. 16, no. 4, pp. 321-357, 2002.
[10] E. P. a. C. o. t. E. Union, "Regulation (EU) 2024/1689 of the European Parliament and of the Council Artificial Intelligence Act," Official Journal of the European Union, Luxembourg, 2024.
[11] S. M. Lundberg and S.-I. Lee, "A Unified Approach to Interpreting Model Predictions," in Curran Associates, Inc., Long beach, 2017.
[12] B. Adams and S. McIntosh, "Modern Release Engineering in a Nutshell Why Researchers Should Care," in
IEEE, Osaka, 2016.
[13] J. Muse and C. Mitchell, "Paper Mill Boiler Ash and Lime By-Products as Soil Liming Materials," Agronomy Journal, vol. 87, no. 3, pp. 432-438, 1995.
[14] F. Imrie, B. Cebere and E. F. McKinney, "AutoPrognosis 2.0: Democratizing Diagnostic and Prognostic Modeling in," PLOS Digital Health, vol. 2, no. 6, pp. 234-267, 2023.
[15] H. Nori, S. Jenkins, P. Koch and R. Caruana, "InterpretML: A Unified Framework for Machine Learning,"
Machine Learning, vol. 4, no. 1, pp. 234-289, 2019.
[16] R. T. Fielding and R. N. Taylor, "Reflections on the REST Architectural Style and "Principled Design of the Modern," in ACM, CA, 2017.
[17] O. B. Fredj and O. Cheikhrouhou, "An OWASP Top Ten Driven Survey on Web Application Protection Methods," in Springer International Publishing, Cham, 2020.
[18] L. B. e. al, "API Design for Machine Learning Software: Experiences from the scikit-learn," in ECML PKDD Workshop: Languages for Data Mining and Machine Learning, Prague, 2013.
[19] D. H. Wolpert, "Stacked Generalization," Neural Networks, vol. 5, no. 2, pp. 241-259, 1992.
[20] A. Asuncion and D. J. Newman, UCI Machine Learning Repository, Irvine: University of California, 2007.
[21] P. Crovari, S. Pidò, P. Pinoli and A. Bernasconi, "GeCoAgent: A Conversational Agent for Empowering Genomic Data Extraction and," ACM Transactions on Computing for Healthcare (HEALTH), vol. 3, no. 1, pp. 1-29, 2021.