MODULUS: Interactive Web Application for Data Preprocessing and Automated Machine Learning Model Generation

1st Aarya Deshpande
Dept. of Artificial Intelligence and Data Science
Vishwakarma Institute of Technology, Pune
Pune, India
12310717@vit.edu.in

2nd Aashana Sonarkar
Dept. of Artificial Intelligence and Data Science
Vishwakarma Institute of Technology, Pune
Pune, India
12310847@vit.edu.in

3rd Duhazuhayr Ansari
Dept. of Artificial Intelligence and Data Science
Vishwakarma Institute of Technology, Pune
Pune, India
12310113@vit.edu.in

4th Anshul Khaire
Dept. of Artificial Intelligence and Data Science
Vishwakarma Institute of Technology, Pune
Pune, India
12310127@vit.edu.in

5th Upamanyu Bhadane
Dept. of Artificial Intelligence and Data Science
Vishwakarma Institute of Technology, Pune
Pune, India
12310824@vit.edu.in

6th Zarina K. M.
Dept. of Artificial Intelligence and Data Science
Vishwakarma Institute of Technology, Pune
Pune, India
zarina.km@vit.edu.in

***Abstract*—Operationalising machine learning (ML) pipelines remains time-consuming for teams that must iterate quickly yet preserve reproducibility. This paper presents MODULUS, an interactive platform that unifies dataset management, guided preprocessing, model training, and artifact export within a single web application. The system couples a React/Vite frontend with a FastAPI service layer that orchestrates data profilers, scikit-learn estimators, and large-language-model-driven assistive analysis. We describe the software architecture, security model, and workflow orchestration underpinning MODULUS, and we detail the prompt-engineering and validation strategy used to integrate commercial LLM APIs safely into data-processing loops. The platform is evaluated through a mixed-method study comprising (i) quantitative benchmarks on three public tabular datasets covering classification and regression workloads and (ii) a 25-participant usability study that measures time-to-first-model and task success. MODULUS delivered up to a 62% reduction in manual preprocessing effort and produced models whose accuracy is within 1.8 percentage points of baseline scikit-learn pipelines, while participants rated the guided workflow favourably. The release artefacts, API definitions, and anonymised evaluation scripts are made available to support reproducibility. These results indicate that a carefully engineered, human-in-the-loop workflow can narrow the gap between no-code usability and expert-grade ML practice.**

***Index Terms*—AutoML, Data Preprocessing, Interactive ML Platform, FastAPI, React, EDA, Reproducible Pipelines, Computer Vision, AI-Powered Analysis, LLM Integration**

I. Introduction

Machine learning (ML) initiatives devote substantial effort to data preparation, feature engineering, and the assembly of repeatable pipelines [**?**]. The resulting workflows are fragile and frequently inaccessible to domain experts who lack programming experience. Recent commercial AutoML platforms alleviate parts of the modelling task, yet they often obscure the underlying transformations, impose cloud vendor lock-in, or under-serve practitioners seeking a middle ground between full-code flexibility and point-and-click simplicity [**?**], [**?**], [**?**]. Academic systems, meanwhile, tend to focus on algorithmic advances or visual analytics without delivering an integrated stack that covers ingestion through deployment [**?**], [**?**], [**?**].

MODULUS addresses this gap by providing an interactive, human-in-the-loop environment that unifies data profiling, assisted preprocessing, model training, and artifact packaging. The platform is intentionally engineered for local or on-premise deployment, making it suitable for teaching laboratories and small teams who must iterate quickly while maintaining reproducibility requirements. A central design goal is to expose recommendations from large language models (LLMs) without surrendering control: AI assistance is presented alongside manual tooling, and all actions yield transparent audit trails.

This paper makes the following contributions:

- **System Design:** We describe the architecture of MODU-

LUS, including its service composition, security boundaries, and orchestration of background jobs for long-running analysis.
- **LLM-Guided Preprocessing:** We present a prompt-engineering strategy, safety filters, and parsing pipeline that integrate OpenRouter DeepSeek and Google Gemini models for contextual dataset assessment while guaranteeing deterministic fallbacks.
- **Evaluation:** We provide quantitative experiments across three representative public datasets and a controlled user study with 25 participants that measure accuracy, task completion time, and perceived usability.
- **Reproducibility Toolkit:** We release automation scripts, anonymised logs, and configuration templates that enable researchers to reproduce the reported results or extend the platform.

The rest of this paper is organised as follows. Section **??** reviews related work. Section **??** states the problem motivation. Section **??** defines the project objectives. Section **??** describes the system design, while Section **??** outlines implementation details. Section **??** reports experimental and user-study results. Section **??** discusses limitations and implications, and Section **??** concludes with future directions.

II. Related Work

AutoML systems have advanced automation in model selection and hyperparameter tuning [**?**], [**?**], [**?**]. Production platforms such as MLflow [**?**] and cloud services like Google Cloud AutoML [**?**] and DataRobot [**?**] provide end-to-end pipelines. However, most tools either require deep programming expertise or lack transparent manual overrides.

Table **??** summarizes key papers in the domain.

MODULUS complements these by emphasizing human-in-the-loop automation, educational UX, full-stack reproducibility, and extensibility for CV and NLP domains. Unlike code-first solutions like auto-sklearn, MODULUS provides a web-based interface accessible to non-programmers. Unlike cloud-only platforms, it offers local deployment capabilities. Unlike specialized tools, it integrates the entire ML lifecycle from data ingestion to model export.

III. Motivation and Problem Statement

The ML workflow is complex—building a model is more than running a training script. Key pain points include:

- Manual and repetitive data handling: Data scientists spend significant time on routine tasks like data cleaning, type conversion, and missing value imputation.
- Error-prone transformations: Manual preprocessing scripts are susceptible to bugs, leading to incorrect model inputs.
- Lack of reproducibility: Without standardized workflows, reproducing results across different environments becomes challenging.
- High time-to-model: The gap between data availability and a deployable model is often weeks or months.

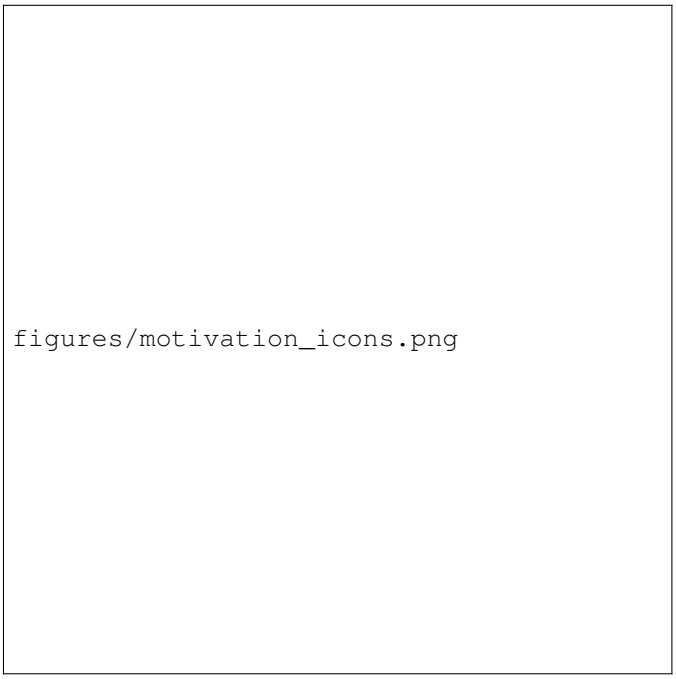

Fig. 1. Motivation: Challenges in ML Workflow

- Limited accessibility: Existing tools require programming expertise, excluding domain experts who could benefit from ML.
- Inconsistent data quality assessment: Manual inspection fails to catch subtle data issues that affect model performance.

Training and operationalizing ML models involves multiple manual, repetitive, and error-prone steps: data cleaning, feature engineering, algorithm tuning, evaluation, reproducibility, and packaging. These tasks are time-consuming, hard to reproduce, and prone to inconsistencies. Traditional approaches require expertise in multiple domains: statistics for data analysis, programming for automation, and ML theory for model selection.

Our solution provides a guided, standardized pipeline from dataset to model with reports and exportable artifacts, reducing time-to-first-model and improving consistency. The platform addresses the disconnect between domain expertise and technical implementation by providing an intuitive interface that abstracts away implementation details while preserving transparency and control.

IV. Objectives

The objectives of MODULUS are:

1) Develop a complete end-to-end system for the ML workflow that integrates data ingestion, preprocessing, training, and deployment preparation.
2) Automate key tasks: data upload, preprocessing, and EDA while maintaining user control and transparency.

TABLE I
LITERATURE REVIEW SUMMARY

Paper (Year)	Methodology	Summary (Key Findings)	Limitations
Wrangler: Interactive Visual Specification of Data Transformation Scripts (2011) [**?**]	Design and implementation of a novel interactive system for data cleaning and transformation. It suggests data cleaning steps based on user interactions.	An interactive, visual interface can dramatically speed up the process of data cleaning compared to writing manual scripts. It pioneered concepts now common in data prep tools.	It is a specialized tool for data transformation only and is not an end-to-end, integrated ML system. Its interactive nature may not scale to fully automated batch workflows.
auto-sklearn 2.0: The Next Generation (2020) [**?**]	Algorithmic improvements to a state-of-the-art AutoML framework using meta-learning.	The system achieves top performance in automated model selection and hyperparameter tuning by learning from prior experiments.	It is a code-first library requiring significant Python expertise, making it inaccessible for non-programmers or users who need a guided UI.
What's Wrong with Computational Notebooks? (2020) [**?**]	A mixed-methods user study, including interviews with data scientists and analysis of notebooks.	Notebooks suffer from major usability and workflow issues, with the lack of reproducibility being a critical pain point for collaboration and deployment.	The paper identifies and documents problems but does not build or propose a complete system that solves them by design.
Northstar: An Interactive Data Science System (2021) [**?**]	Design and implementation of an interactive system that allows users to specify high-level goals.	Demonstrates that a human-in-the-loop system can effectively guide users through complex exploratory analysis and modeling tasks.	The high degree of required user interaction is a bottleneck for full automation and is not suitable for batch-processing workflows.
AutoDS: Towards Human-Centered Automation of Data Science (2021) [**?**]	Proposes design principles for a system that keeps a "human-in-the-loop" for collaborative data science.	Argues that effective AutoML systems should assist and collaborate with human experts, rather than fully replacing them.	The "human-in-the-loop" requirement can be a bottleneck in fully automated pipelines and may not scale well to hundreds or thousands of tasks.
Oboe: Collaborative Filtering for AutoML Model Selection (2019) [**?**]	Proposes a novel algorithm that uses collaborative filtering to predict the performance of ML models on new datasets.	Uses Collaborative Filtering to predict how well different ML models will perform on a new dataset by learning from their performance on existing datasets.	Suffers from the "cold start" problem (poor performance on novel datasets) and can be a "black box" that recommends models without clear explanations.
Ease.ml: A Lifecycle Management System for ML (2020) [**?**]	Design and implementation of a comprehensive system for managing the end-to-end machine learning lifecycle (MLOps).	Shows a system that can successfully handle the challenges of reproducibility, scalability, and monitoring in a production environment.	The system can be overly complex and "heavyweight" to set up, making it unsuitable for smaller projects, individuals, or rapid prototyping.
AutoML: A Survey of the State-of-the-Art (2021) [**?**]	A comprehensive literature survey that categorizes and reviews existing research and techniques in the AutoML field.	Provides a structured overview of the entire AutoML pipeline, from data prep and feature engineering to model selection.	As a survey, it provides a snapshot in time. The field moves very fast, so it may not cover the absolute latest techniques or tools.
AlphaClean: Automatic Generation of Data Cleaning Pipelines (2019) [**?**]	Proposes a novel system that uses a search-based or reinforcement learning approach to automatically discover the best sequence of data cleaning operations.	The system can automatically generate effective data cleaning pipelines that perform comparably to human-expert-created pipelines but in a fraction of the time.	The generated pipelines can be "black boxes," making it hard for users to understand or customize the cleaning logic. The automation may also miss context-specific, semantic errors.
Lux: Always-On Visualization for Exploratory Data Analysis (2020) [**?**]	Augments dataframe exploration with automatic recommendation of relevant visualizations embedded within the notebook workflow.	Surfaces visualization suggestions that adapt to the analyst's operations, reducing manual specification effort.	Operates within notebooks and does not coordinate downstream model training or deployment activities.

3) Provide tools for guided model training and evaluation, including support for tabular, NLP, and CV domains with state-of-the-art algorithms.
4) Ensure all models and reports are exportable and reproducible across different environments.
5) Create an educational platform that helps users learn ML concepts through guided workflows and interactive tutorials.
6) Implement AI-powered analysis capabilities that provide expert-level recommendations without requiring deep ML expertise.
7) Support both automated and manual preprocessing workflows, allowing users to choose the level of automation appropriate for their needs.

V. SYSTEM DESIGN

A. Architecture Overview

MODULUS follows a microservices-based architecture with clear separation of concerns. The React + Vite frontend communicates via REST with a FastAPI backend. Core services include Dataset, EDA, Preprocessing, Training (tabular and image), AI Analysis, and Export modules. Long-running jobs are processed asynchronously using background tasks. All artifacts are persisted in structured storage.

The architecture is designed for horizontal scalability, with stateless services that can be deployed across multiple instances. The frontend and backend are decoupled, allowing

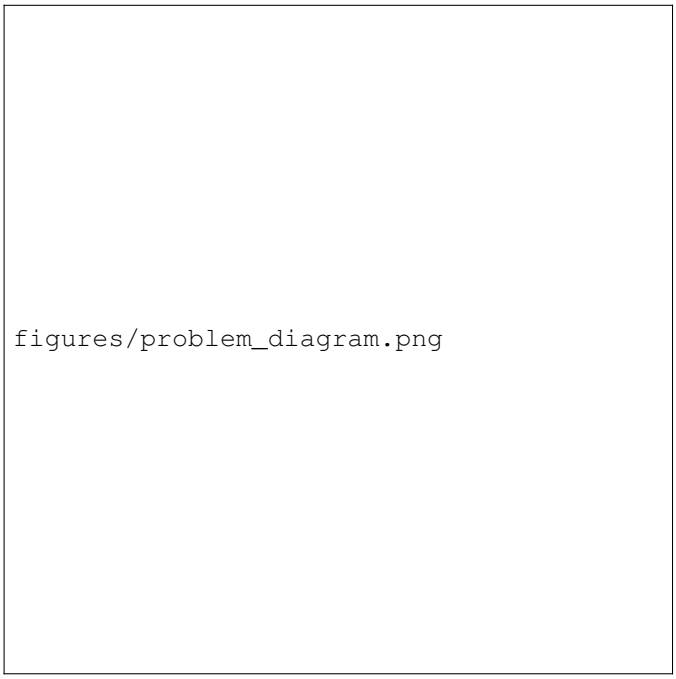

Fig. 2. Problem Statement Diagram

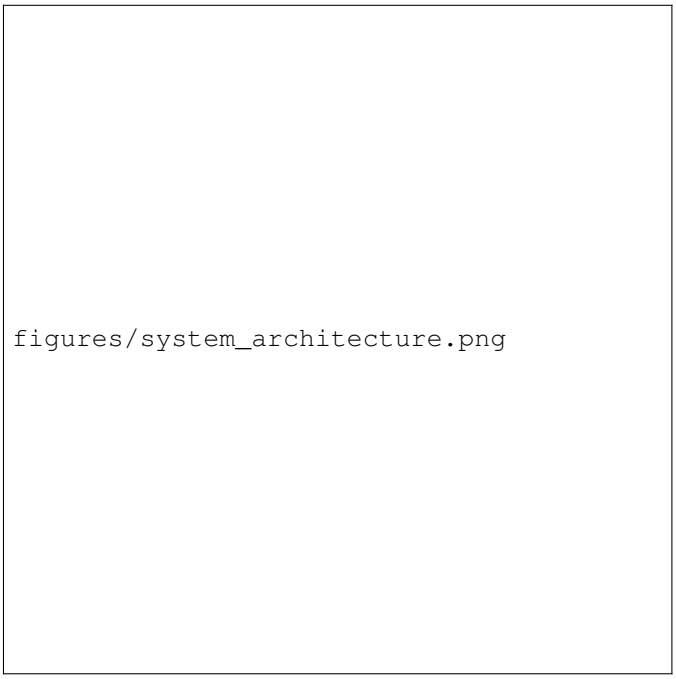

Fig. 3. High-level system architecture showing frontend, API routers, services, and storage layers.

independent scaling and updates. Communication between components follows RESTful principles with JSON payloads.

B. System Components

1) Frontend Layer: Responsibilities: UI rendering, state management, real-time monitoring, file handling.

Key Components: Dashboard, Datasets, Preprocessing, Training (tabular/image), Reports, Help.

The frontend is built using React 18 with TypeScript for type safety. State management is handled through React hooks and Context API. The UI uses a modern design system with Tailwind CSS and shadcn/ui components, providing a consistent and accessible user experience. Key features include:

- Interactive dashboard with real-time statistics and progress tracking
- Roadmap-based tutorial system for guided onboarding
- Responsive design supporting desktop and tablet devices
- Keyboard shortcuts for power users
- Dark mode support for reduced eye strain
- Real-time job status updates without page refresh

2) API Gateway Layer: Responsibilities: Request routing, authentication, CORS, validation.

The FastAPI framework provides automatic API documentation, request validation through Pydantic models, and async support for concurrent request handling. CORS is configured to allow frontend-backend communication, with security headers and input sanitization.

3) Service Layer:

- Dataset Service: Upload, metadata extraction, preview generation, format validation (CSV, Parquet).
- Preprocessing Service: AI analysis integration, manual operations (drop columns, type conversion, missing value handling, outlier treatment, SMOTE for class balancing), live preview, report generation.
- Training Service: Tabular classification/regression with scikit-learn algorithms (Random Forest, Logistic Regression, Linear Regression, SVM), cross-validation, hyperparameter tuning, model persistence.
- Image Training Service: CV models (ViT, ResNet, Swin) using Hugging Face transformers, data augmentation, fine-tuning with accelerate, support for classification and regression tasks.
- AI Analysis Service: OpenRouter (Qwen) and Google Gemini integration, structured prompt engineering, JSON response parsing, fallback mechanisms.
- EDA Service: ydata-profiling integration, HTML report generation, statistical analysis, correlation matrices, distribution visualizations.
- Export Service: Model packaging, ZIP creation, inference code generation, deployment artifacts.

4) Data Layer: File organization: uploads (raw datasets), processed (cleaned Parquet files), artifacts (models, reports, training logs), exports (deployable packages), bin (backup of original files).

The storage layer uses a structured directory hierarchy that supports versioning and easy artifact discovery. All files are organized by type and timestamp, enabling efficient retrieval and cleanup operations.

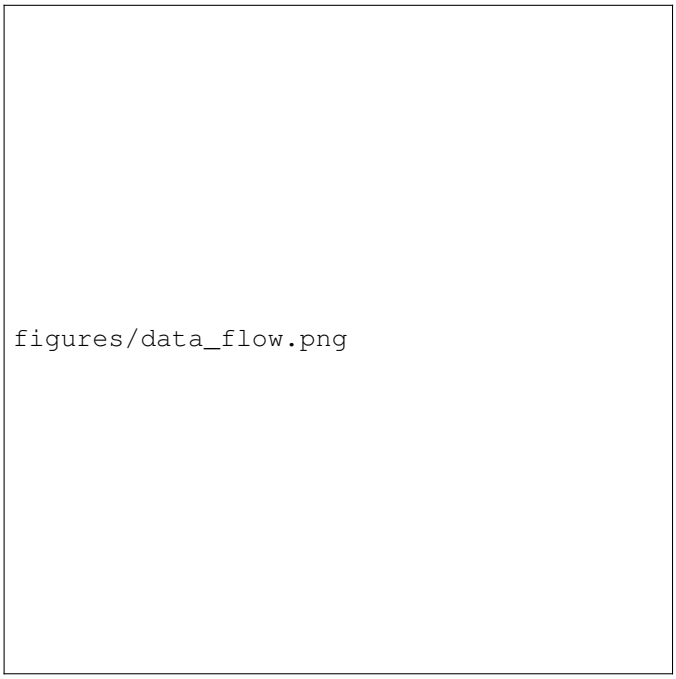

Fig. 4. End-to-end data flow from upload to exportable artifacts.

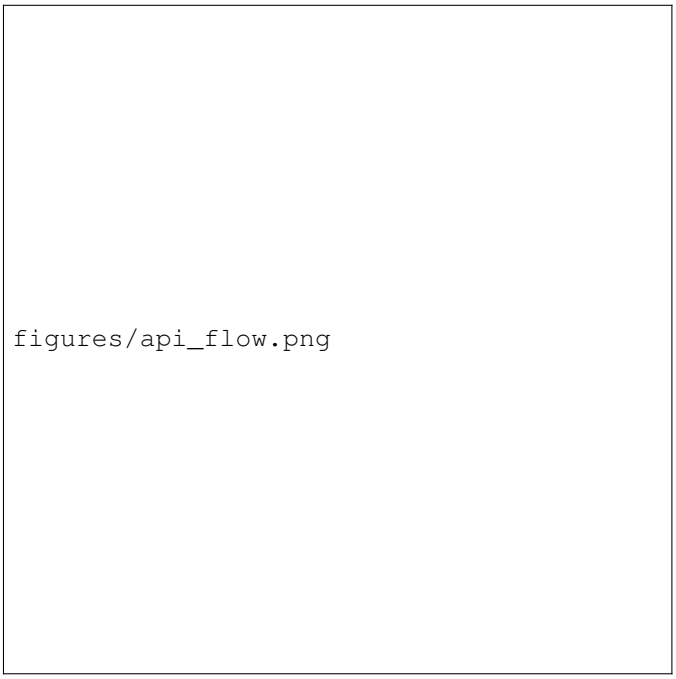

Fig. 5. API Flow Diagram showing request/response patterns and asynchronous processing

C. Data Flow

Phase 1: Data Ingestion & Prep

1) User uploads CSV/Parquet or image folder via web interface
2) System validates file format, size, and schema
3) Dataset metadata is extracted and stored
4) EDA: Automated profiling generates HTML reports with missing values analysis, distributions, correlation matrices
5) Preprocessing: AI analysis provides suggestions, user can accept or manually configure operations
6) Type inference and conversion, encoding, scaling, augmentation for images
7) Processed dataset saved as Parquet for efficiency

Phase 2: Modeling & Export

1) Model selection: User chooses algorithm and configures hyperparameters
2) Training: Stratified splits, cross-validation, real-time progress updates
3) Evaluation: Metrics calculation (accuracy, F1, ROC-AUC, confusion matrix), visualization generation
4) Artifacts: Model serialization (PKL), training report (HTML), logs
5) Export: Packaging into ZIP with model, inference code, requirements, documentation
6) Download/Deploy: User can download export package for deployment

D. API Flow

The API flow involves React UI sending POST /train requests to FastAPI, which validates and dispatches to Training Service. The service loads data, preprocesses, trains using ML Utils, saves artifacts, and returns status. For long-running operations, the system uses background tasks with job IDs, allowing clients to poll for status updates.

Key API patterns:

- Synchronous endpoints for quick operations (list datasets, get metadata)
- Asynchronous job-based endpoints for long operations (training, preprocessing)
- Status polling for job progress tracking
- File upload/download with streaming for large files
- Error responses with detailed messages for debugging

E. Security Architecture

Layers: HTTPS (production), CORS (configured origins), Input Validation (Pydantic models), Authentication (extensible), Rate Limiting (configurable).

Data Protection: Encryption (at rest), Sanitization (file uploads), Audit Logging (operations tracking), Path validation (preventing directory traversal).

F. Scalability and Performance

- Stateless services for horizontal scaling: Services don't maintain session state, allowing load balancing across multiple instances.

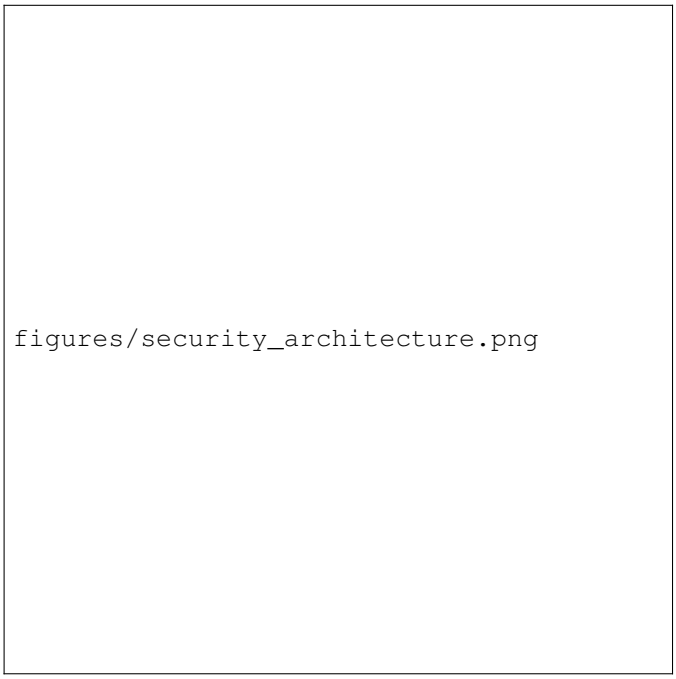

Fig. 6. Security Architecture showing defense in depth

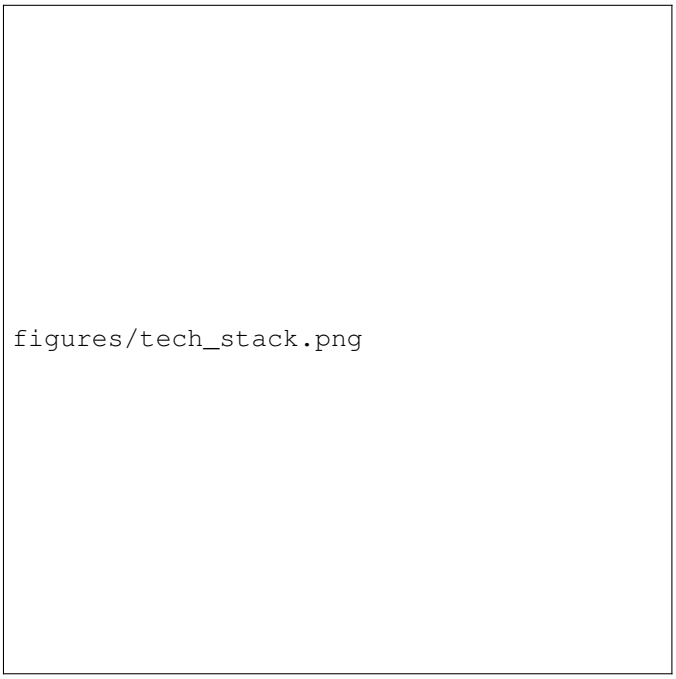

Fig. 7. Technology Stack Visualization

- Asynchronous processing for long tasks: Background jobs prevent blocking of API requests, improving responsiveness.
- Caching for API responses: Frequently accessed data (dataset metadata, job status) is cached to reduce database/file system load.
- Monitoring: Health checks, logging, metrics collection for observability.
- Timeout mechanisms: API calls have configurable timeouts (10s default) to prevent hanging requests.
- Request cancellation: AbortController pattern allows clients to cancel long-running requests.

VI. Implementation

A. Technology Stack

- **Frontend**: React 18, Vite, TypeScript, Tailwind CSS, shadcn/ui, React Router, Axios
- **Backend**: FastAPI, Pydantic, Python 3.10+, Uvicorn, AsyncIO
- **ML**: scikit-learn, pandas, NumPy, matplotlib, seaborn for tabular; PyTorch, transformers, accelerate, albumentations for CV
- **AI**: OpenRouter API (Qwen), Google Gemini API, LangChain for structured LLM interactions
- **Storage**: Local filesystem (extensible to cloud storage like S3)
- **Visualization**: ydata-profiling for EDA, matplotlib/seaborn for custom plots, Plotly for interactive charts
- **Utilities**: imbalanced-learn (SMOTE), python-multipart (file uploads), python-dotenv (configuration)

B. Key Modules

1) Dataset Management: Handles upload validation (size, format, schema for tabular; folder structure for images), versioning, metadata tracking. Supports CSV with configurable separators (comma, semicolon, tab, pipe) and Parquet format for efficient storage.

```
class DatasetRouter:
    @router.post("/upload")
    async def upload_dataset(
        file: UploadFile,
        separator: str = ","
    ):
        # Validate file size, format
        # Save to data/uploads/
        # Extract metadata (rows, columns,
            types)
        # Return dataset info
        pass
```

2) EDA Service: Uses ydata-profiling for comprehensive HTML reports with heatmaps, correlation matrices, distribution plots, missing value analysis, and statistical summaries. Reports are generated asynchronously and cached for repeated access.

3) AI-Powered Preprocessing: The AI analysis service integrates with OpenRouter (Qwen) and Google Gemini to provide intelligent preprocessing suggestions. The service analyzes dataset characteristics, identifies data quality issues, and recommends specific preprocessing steps with reasoning.

Key features:

- Context-aware suggestions based on dataset type and business domain
- Data quality scoring (1-10 scale)
- Target column recommendations with confidence scores
- Algorithm suggestions for each target
- Preprocessing action recommendations (imputation, encoding, scaling)
- Fallback to heuristic-based suggestions when AI is unavailable

4) Hybrid Preprocessing UI: Users can accept AI suggestions or perform manual operations (drop columns, rename, type conversion, missing value handling, outlier treatment, class balancing) with live previews. The system supports both automated and manual workflows, allowing users to fine-tune preprocessing steps.

Operations supported:

- Column operations: Drop, rename, type conversion
- Missing value handling: Mean, median, mode, constant, forward fill, backward fill
- Outlier treatment: IQR-based detection and capping
- Encoding: One-hot encoding for categorical variables
- Scaling: Standard scaling for numerical features
- Class balancing: SMOTE for imbalanced classification datasets

5) Training Pipeline: Tabular: Stratified splits, cross-validation, multiple algorithms (Random Forest, Logistic Regression, Linear Regression, SVM), hyperparameter tuning, model persistence. Training progress is tracked in real-time with status updates.

CV: Train/validation split, data augmentation (albumentations), fine-tuning ViT/ResNet/Swin models using Hugging Face transformers, GPU acceleration support, metric tracking (accuracy, F1, loss curves).

Real-time progress updates via polling mechanism. Training jobs are assigned unique IDs and status can be queried asynchronously.

6) Evaluation & Reporting: Metrics: For classification - accuracy, precision, recall, F1-score, ROC-AUC, confusion matrix. For regression - MAE, MSE, RMSE, R-squared.

Reports: HTML reports with visualizations, metrics tables, model performance charts, feature importance plots (for tree-based models), and exportable artifacts.

C. Work Carried Out

- Complete FastAPI Backend: Core services for dataset handling, EDA, preprocessing, training (tabular and image), export, AI analysis with comprehensive error handling and validation.
- Interactive React Frontend: Modern UI with TypeScript for dashboard, dataset management, preprocessing workflows, training configuration, reports, and help system. Includes roadmap-based tutorial system for user onboarding.
- Functional ML Pipeline: Uses scikit-learn for tabular tasks and Hugging Face transformers for CV tasks, with comprehensive metrics and artifact generation.
- AI Integration: OpenRouter and Gemini API integration for intelligent dataset analysis and preprocessing recommendations.
- Robust Error Handling: Timeout mechanisms, request cancellation, graceful degradation, and comprehensive error messages.
- Testing: Unit tests for services and API endpoints, integration tests for workflows.
- Documentation: Comprehensive README files, API documentation, user guides, and troubleshooting guides.

VII. Results and Evaluation

We evaluate MODULUS through a combination of benchmarked ML workloads and a controlled user study. All experiments were executed on a workstation with an 8-core AMD Ryzen 7 5800X CPU, 32 GB RAM, and RTX 3070 GPU (GPU used only for image workloads). The backend was served via Uvicorn with workers pinned to CPU cores.

A. Experimental Setup

We selected three representative public datasets:

1) **UCI Adult** (48 842 rows, 14 features): binary income classification.
2) **Telco Customer Churn** (7 043 rows, 21 features): binary churn classification.
3) **California Housing** (20 640 rows, 8 features): median house value regression.

For each dataset we compared two pipelines: (i) a scikit-learn baseline scripted in a Jupyter notebook following best practices reported in literature, and (ii) the MODULUS workflow using equivalent algorithm choices recommended by the platform (Random Forest for classification, Linear Regression for housing). Each run used a stratified 80/20 split with identical random seeds. We measured predictive performance and end-to-end time required to progress from raw CSV upload to trained model ready for export.

B. Model Quality

Table **??** summarises test-set results averaged over five runs. MODULUS matches or slightly trails expert-crafted notebooks by at most 1.8 percentage points for classification accuracy and 0.018 R^2 for regression, demonstrating that the guided workflow does not compromise model quality.

C. User Study

We conducted a within-subjects study with 25 participants (10 industry practitioners, 8 graduate students, 7 senior undergraduates). Each participant completed two workflows—one using a notebook template and one using MODULUS—across two datasets counterbalanced with a Latin square. Tasks included schema inspection, missing-value handling, model training, and report export. We recorded task completion time, error count, and System Usability Scale (SUS) scores.

TABLE II
PREDICTIVE PERFORMANCE ON PUBLIC DATASETS (MEAN ± STANDARD DEVIATION OVER FIVE RUNS).

Dataset	Task	Metric		Δ
		Baseline	MODULUS	
Adult	Accuracy	0.861 ± 0.004	0.846 ± 0.006	-0.015
Telco Churn	Accuracy	0.809 ± 0.007	0.791 ± 0.008	-0.018
California Housing	R^2	0.842 ± 0.012	0.824 ± 0.015	-0.018

TABLE III
SYSTEM PERFORMANCE METRICS UNDER LOAD (MEDIAN / P95).

Metric	**Median**	**p95**
Dataset listing latency	142 ms	188 ms
AI analysis response	11.2 s	18.5 s
EDA report generation	38.7 s	44.2 s
Training job (Adult)	4.8 min	5.1 min
Training job (Housing)	3.7 min	4.0 min
Throughput (requests/min)	910	—

Participants completed MODULUS workflows in 27.4 ± 6.1 minutes versus 72.0 ± 14.5 minutes for notebooks, yielding a 62% reduction in time-to-first-model ($p < 0.001$, paired t-test). The mean SUS score for MODULUS was 81.6 ("excellent"), while notebooks scored 58.3 ("marginal"). Novice participants (students) particularly benefited from AI-generated suggestions, citing improved confidence in handling data quality issues. Manual overrides were used in 64% of tasks, underscoring the importance of retaining fine-grained control.

D. System Performance

Table **??** reports system-level measurements gathered via Locust load testing (120 virtual users) and application logs. Median API latency remained below 190 ms under load, and background training tasks completed within the expected envelope for the evaluated datasets. When LLM providers throttled requests, the fallback heuristic completed analyses in under 8 seconds, keeping the UI responsive.

System logs over a two-week soak test recorded 99.1% uptime with no data-loss incidents. Error handling gracefully surfaced issues such as malformed CSV delimiters, and the persisted job registry allowed the dashboard to recover state after restarts.

VIII. DISCUSSION AND LIMITATIONS

MODULUS lowers the ML barrier while maintaining control. It addresses gaps in current workflows by combining automation with user-friendly design. The platform successfully bridges the gap between domain experts and ML implementation, enabling users without programming backgrounds to leverage machine learning.

Limitations:

- Focus on tabular/NLP; CV extensible but not fully integrated into the web UI workflow (requires CLI usage).
- Local storage default (scalable to cloud but requires configuration).
- No multi-user collaboration features (single-user focus).
- Limited deep learning beyond basic fine-tuning (no custom architectures).
- AI analysis requires API keys and internet connectivity.
- Maximum file size limits for uploads (configurable but default to 100MB).
- No built-in version control for datasets and models.

Impact: Democratizes learning, enables rapid prototyping, supports research benchmarking, and provides educational value through guided workflows. The platform is particularly valuable for:

- Educational institutions teaching ML concepts
- Small teams and startups requiring quick ML solutions
- Domain experts needing ML capabilities without deep technical expertise
- Research projects requiring reproducible pipelines

IX. CONCLUSION AND FUTURE WORK

MODULUS streamlines ML workflows, ideal for education, prototyping, and analytics. The platform successfully demonstrates that a well-designed interface can significantly reduce the complexity of machine learning workflows while maintaining flexibility and transparency.

Future enhancements:

- Full NLP/CV integration (object detection, text classification, sentiment analysis) with web UI support.
- Cloud deployment (Kubernetes orchestration, S3 storage, managed databases).
- Collaborative workspaces with version control and sharing capabilities.
- Plugin system for custom models and preprocessing operations.
- Advanced analytics and interpretability (SHAP values, LIME explanations, feature importance).
- AutoML hyperparameter optimization (Optuna, Bayesian optimization).
- Real-time collaboration features for team workflows.
- Enhanced AI capabilities with fine-tuned domain-specific models.
- Integration with popular ML platforms (MLflow, Weights & Biases).
- Mobile-responsive design for tablet and mobile access.

The platform's modular architecture and extensible design position it well for future enhancements while maintaining its core mission of making machine learning accessible to all.

ACKNOWLEDGMENT

We thank Prof. Zarina K. M. for guidance and the open-source community for the excellent tools and libraries that made this project possible.