AutoML Lite: A Lightweight, Interpretable, End-to-End AutoML System for Tabular Data

Joosun Hwang

May 21, 2025

Abstract

AutoML Lite presents a lightweight, user-friendly tool for automating machine learning workflows on tabular data. Unlike heavier AutoML frameworks (e.g., Auto-sklearn, TPOT), this system focuses on interpretability, portability, and fast experimentation for small to medium datasets. We propose a five-stage pipeline: automatic task detection, preprocessing with ColumnTransformers, model selection, Optuna-based hyperparameter tuning, and exportable joblib pipelines. With a Streamlit UI and MLflow tracking, users can run full ML pipelines without writing code. We validate AutoML Lite on datasets including Titanic, California Housing, and Football transfer fee, achieving up to 87 percent accuracy and 0.93 R^2. This system serves as an educational and practical tool for rapid prototyping and explainable model development.

1 Introduction

Automated Machine Learning (AutoML) has become a cornerstone of modern AI development, empowering non-experts to build predictive models by abstracting away the complexities of data preprocessing, model selection, and hyperparameter tuning. As AI adoption accelerates across industries, AutoML plays a vital role in democratizing access to machine learning.

The global AutoML market was valued at 1.0 billion in 2023 and is projected to grow to 6.4 billion by 2028, at a compound annual growth rate (CAGR) of 44.6 percent. This surge is driven by its applications across sectors including banking, IT, telecommunications, logistics, media, and manufacturing. Key AutomML innovations fueling this growth include data processing pipelines, feature engineering, model ensembling, and hyperparameter optimization.

However, despite this expansion, current AutoML platforms face critical barriers:

- Black-box limitations: Major players like Google AutoML, H2O, and Autosklearn offer limited transparency into their internal workflows. Users often can't inspect which models or features are chosen, making debugging and interpretation difficult. This is a major concern in regulated sectors like finance and healthcare, where explainability is crucial.
- Heavyweight infrastructure: Many platforms require cloud compute, Kubernetes, or enterprise-scale resources, making them impractical for students, indie developers, or research labs with limited resources.
- Limited customizability: Existing tools offer minimal control over feature engineering or pipeline customization, which hinders domain-specific innovation and experimentation especially in time series or mixed modality data.
- Hight cost and poor UX: Proprietary tools like DataRobot or AWS SageMaker AutoPilot often lock users behind paywalls and provide interfaces that are either too opaque(One click automation) or too complex (config-heavy scripts).

These issues reveal a clear need for a lightweight, interpretable, and user-friendly AutoML alternative on that is educational and accessible without sacrificing performance or transparency.

AutoML Lite was built to address exactly this gap.

Our system is:

- Fully local and portable (no cloud required)
- Modular and transparent, using scikit-learn pipelines and Optuna for tuning.
- Visually interactive, with an intuitive Streamlit interface
- Supportive of both classification and regression.
- Exportable, with .joblib pipelines ready for reuse

In this paper, we propose AutoML lite as a practical tool for small to mid-scale tabular ML tasks, evaluate its performance across multiple public datasets, and discuss its value for education, prototyping, and transparent experimentation.

2 Related Work

Several AutoML frameworks have gained prominence in recent years, each offering unique strategies for automating model development. However many of these tools either lack interpret ability, demand high compute resources, or are difficult to integrate into lightweight, local workflows.

Tool	**Strengths**	**Limitations**
Auto-sklearn	Strong performance, model ensembling	Complex setup, limited UI, less control over inner workings
TPOT	Genetic programming for pipeline discovery	Slow optimization, hard to customize and debug
H2O AutoML	Scalable, integrates with enterprise tools	Opaque modeling process, less transparent preprocessing
AutoGluon	Supports deep learning and multi-modal inputs	Heavyweight,not beginner-friendly
AutoML Lite (Ours)	Lightweight, interpretable, local-first, Streamlit UI	Currently limited model library, designed for tabular use only

Table 1: Comparison of AutoML frameworks and the positioning of AutoML Lite

AutoML Lite is designed to complement not compete directly with these frameworks. Its niche lies in providing:

- Transparency: All preprocessing, tuning, and modeling steps are explicitly visible and customizable via code or UI.
- Interactivity: The streamlit frontend allows users to upload datasets, select targets, visualize results, and download trained pipelines without touching code.
- Portability: Entirely local, without cloud setup or GPU dependency, making it suitable for students, educators, and prototyping

By focusing on explainability, responsiveness, and practical usability, AutoML lite offers a compelling alternative for students, educators and developers needing accessible AutoML workflows.

3 Method

3.1 System Overview

AutoML Lite is an end-to-end, user friendly AutoML system designed for tabular data tasks. It integrates core machine learning stages preprocessing, model selection, hyperparameter optimization, and evaluation into a seamless pipeline. The system is implemented in Python using open source libraries and exposed to users via a visual Streamlit based interface.

3.1.1 Data Ingestion

The system accepts user-uploaded datasets in .csv format via the Streamlit interface. Once uploaded, a preview of the dataset in rendered, and the user selects the target column for supervised learning. The system checks each column's uniqueness and data type to automatically suggest the most suitable target column for classification or regression.

3.1.2 Task Detection

AutoML Lite automatically determines whether the task is classification or regression based on the target column:

- If the number of unique target values is small (¡=10) and categorical, it is treated as a classification problem.
- If the target is continuous with high cardinality, it is treated as a regression problem.

This heuristic avoids user misconfiguration and ensures correct model and metric selection.

3.1.3 Preprocessing

A ColumnTransformer is constructed to apply appropriate preprocessing per column type:

- Numerical columns: imputed using the mean and standardized with StandardScaler.
- Categorical columns: imputed using the most frequent value and encoded using OneHotEncoder.

This preprocessing is bundled into a Pipeline using sklearn.pipeline.Pipeline and is later combined with the trained model for export.

3.1.4 Model Selection

AutoML Lite supports a modular and extensible configuration phase of the model. Based on the detected task, appropriate scikit-learn or XGBoost models are selected:

- Classification models:Logistic Regression, Random Forest, XGBoost, KNN, SVM
- Regression models: Linear Regression, Random Forest Regressor, XGBoost Regressor, Lasso.

3.1.5 Hyperparameter Optimization

Each model is fine-tuned using Optuna, a modern hyperparameter optimization library that leverages Bayesian optimization to efficiently explore the search space. The underlying optimization strategy employed by Optuna is discussed in detail in Section 4. The system performs cross-validation (CV=3) on each trial to evaluate model configurations. Optimization objectives are:

- Accuracy/ ROC-AUC for classification.
- R^2 for regression

Listing 1: Optuna hyperparameter optimization

```
study = optuna.create_study(direction='maximize')
study.optimize(objective, n_trials=10)
#It will try 10 different hyperparameter combinations.
```

3.1.6 Evaluation

The best-performing model is re-trained and evaluated on a holdout test set(20 percent) split. Metrics depend on task type:

- Classification: Accuracy, Precision, Recall, Confusion Matrix, ROC-AUC
- Regression: MAE, RMSE, R^2, Residual plots

This evaluation give the user a clear picture of how well the model generalizes.

3.1.7 Pipeline Export

The preprocessing and trained model are wrapped into a single scikit-learn Pipeline and exported via .joblib:

Listing 2: Exporting trained model, preprocessing steps.

```
Pipeline([
    ('preprocessor', preprocessor),
    ('model', best_model)
])
#All we need to do is put raw data into this pipeline.
```

3.1.8 Streamlit User Interface

AutoML Lite is packaged in a user-friendly web app via Streamlit. Key UI features:

- CSV file uploader
- Automatic target column recommendations
- Task type detection summary
- Dataset preview
- AutoML button to trigger full training pipeline
- Visual outputs:
  - Model leader-board
  - Confusion matrix (classification)
  - ROC curve (binary classification)
  - Residual plot (regression)
- Model download as .joblib file

3.1.9 Optional: MLflow Logging

Support for MLflow is built-in (optional), allowing advanced users to log experiment parameters, scores, and models for experiment tracking and version control.

3.2 Visual Workflow

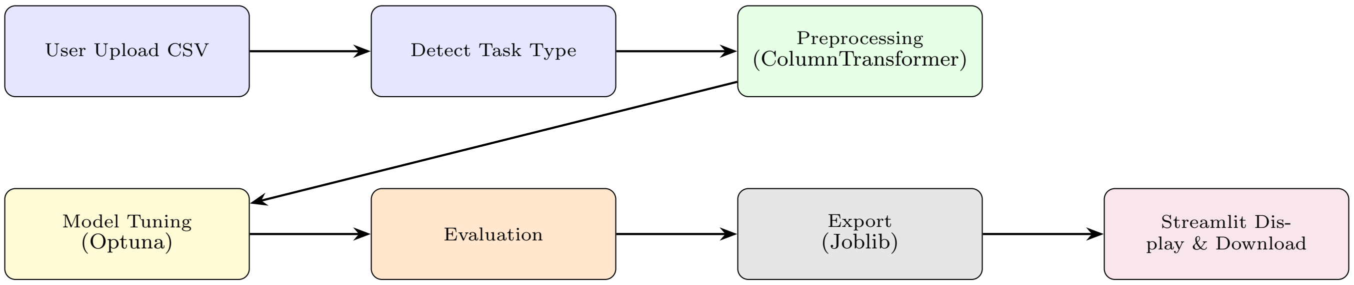

Figure 1: Workflow: From data upload to final Streamlit model deployment

3.3 Hyperparameter Optimization via Bayesian Search (TPE)

To efficiently explore the hyperparameter space, our system leverages Bayesian optimization via the Optuna library. Unlike exhaustive approaches such as grid search, Bayesian optimization constructs a surrogate function—a probabilistic model that approximates the true objective function based on previously evaluated configurations. Optuna, in particular, uses the Tree-structured Parzen Estimator (TPE) as its surrogate model.

TPE models two conditional probability densities:

- $l(x) = p(x \mid y < y^{)}$: configurations likely to yield high performance
- $g(x) = p(x \mid y \geq y^{)}$: configurations associated with lower performance

An acquisition function is then applied to determine the most promising configuration to evaluate next by maximizing the ratio:

$$x^* = \arg\max_x \frac{l(x)}{g(x)} \tag{1}$$

This acquisition strategy balances exploration (sampling uncertain regions) and exploitation (focusing on areas expected to perform well), guiding the search toward optimal solutions efficiently. As a result, it reduces the number of trials required and converges faster than brute-force techniques.

TPE is particularly suitable for mixed-type and conditional search spaces. In our system, each model (e.g., Random Forest, XGBoost, Lasso) defines a unique set of tunable hyperparameters. Optuna automatically adapts to these conditional parameter spaces.

During tuning, Optuna samples hyperparameter dynamically and evaluates each configuration using cross-validation accuracy for classification tasks and R^2 score for regression. This results in a more efficient and generalizable model selection process, driven by validation performance.

3.3.1 Surrogate and Acquisition Functions in Bayesian Optimization

Bayesian optimization operates through two primary components: the *surrogate function* and the *acquisition function*. These work together to guide the search for optimal hyperparameters more efficiently than random or grid search.

The surrogate function is a probabilistic model that approximates the true objective function by learning from prior evaluations. In our AutoML Lite system, we leverage Optuna's use of the Tree-structured Parzen Estimator (TPE) as the surrogate. Unlike Gaussian Processes, TPE models the distributions of good and bad hyperparameter regions separately—denoted by $l(x)$ for promising regions and $g(x)$ for less promising ones. This allows it to model complex, non-smooth objective landscapes more effectively. In a simpler term the surrogate model acts like a statistical guesser—it estimates which hyperparameter settings are likely to work well based on previous results.

The acquisition function uses this surrogate to determine which hyperparameter configurations to evaluate next. By maximizing the ratio $\frac{l(x)}{g(x)}$, the algorithm prioritizes areas where the likelihood of improvement is highest—striking a balance between *exploration* (trying new areas of the search space) and *exploitation* (refining known good areas).

This interplay between the surrogate and acquisition function enables Optuna to converge toward high-performing models with fewer evaluations, which is especially important for lightweight, interactive systems like AutoML Lite.

3.3.2 Intuition and Visualization

Figure 4 illustrates how TPE samples hyperparameter values based on the surrogate model and acquisition function. The red markers represent sampled configurations, the orange curve shows the true (but unknown) objective function $f(x) = (x - 2)^2 + 1$, and the green dashed line indicates the global minimum.

In practice, the true objective function and its minimum are not known. Bayesian optimization starts by randomly sampling a few hyperparameter values and observing their validation losses. It then sets a threshold (e.g., top 20%) and separates observations into two sets: "good" values $\ell(x)$

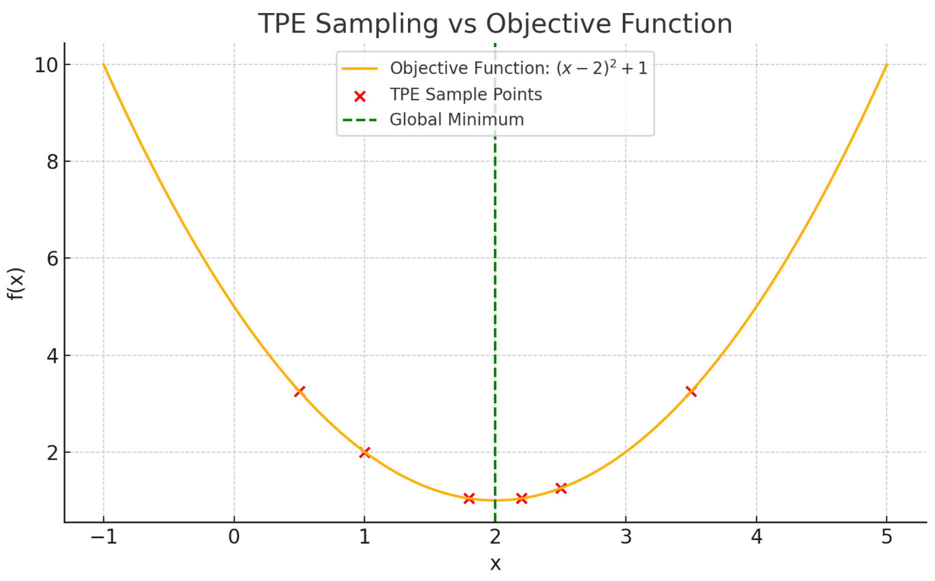

Figure 2: Illustration of TPE sampling behavior over the objective function $f(x) = (x-2)^2 + 1$. Red dots are samples chosen by the acquisition function.

and "bad" values $g(x)$. TPE builds probability density functions over both sets and selects the next candidate x_i that maximizes $\ell(x)/g(x)$.

This process is repeated as follows:

1. Define the search space for a hyperparameter, e.g., `max_depth` $\in [1, 10]$.
2. Sample a few initial values:
    - $x = 3 \rightarrow$ validation loss $= 0.40$
    - $x = 5 \rightarrow$ validation loss $= 0.30$
    - $x = 8 \rightarrow$ validation loss $= 0.45$
3. TPE constructs:
    - $l(x)$: distribution of "good" values (e.g., around 5)
    - $g(x)$: distribution of "bad" values
4. It evaluates a set of candidates, e.g.:

$$\texttt{max\_depth} = \{4.2,\ 6.3,\ 2.8,\ 5.0,\ 9.5\}$$

5. For each candidate, compute the ratio $\frac{l(x)}{g(x)}$
6. Choose the x with the highest score and evaluate it
7. Update the surrogate model with the new result and repeat

3.3.3 Why TPE Works Well for AutoML Lite

TPE is especially effective for our AutoML Lite system because:

- It gracefully handles conditional, hierarchical, and mixed-type hyperparameter spaces.
- It converges quickly—ideal for interactive or resource-constrained systems.
- It avoids wasted evaluations by learning from prior trials.

Each model (e.g., Random Forest, XGBoost, Lasso) defines its own tunable parameters. Optuna dynamically samples from these model-specific spaces and evaluates candidates using cross-validation metrics (e.g., accuracy or R^2). This results in highly tuned models that generalize well, even with limited compute.

4 Experiments

This section evaluates the effectiveness and flexibility of the AutoML Lite framework across various supervised learning tasks, including both classification and regression. We assess its performance using well-known benchmark datasets categorized into three types based on size: small datasets with fewer than 1,000 rows, mid-sized datasets with approximately 4000 rows, and the largest data sized with around 10,000 rows. This range allows us to test how well the framework generalizes across different scales, offering insights into its suitability for both lightweight tasks and more realistic, large-scale scenarios.

4.1 Datasets Used

To ensure diversity in task type and data complexity, the following datasets were selected:

- Titanic Dataset (Classification):
  - Goal: Predict whether a passenger survived based on demographic and ticketing data
  - Size: 891 rows, 12 columns
  - Target: Survived
- Football Primer league players Value dataset (Regression):
  - Goal: Predict player transfer market value based on attributes like age, club, stats
  - Size: 10,000 rows
  - Target: value (in currency)
- Student Dropout rate dataset (Classification)
  - Predict a student's academic outcome (dropout, enrolled, or graduate) based on demographics, academic path, and sociol-economic factors.
  - Size: 4,000 rows
  - Target: final academic status (dropout, enrolled, graduate).

4.2 Experimental Setup

- Train/Test Split: 80/20 random split using `train_test_split` with `random_state=42`
- Models Used: Random Forest, XGBoost, Logistic Regression, KNN, SVM, Lasso (for regression)
- Hyperparameter Tuning: Optuna, 10 trials per model
- Evaluation Metrics:
  - Classification:
    - Accuracy
    - Precision
    - Recall
    - ROC-AUC (for binary)
    - Confusion Matrix
  - Regression:
    - Mean Absolute Error (MAE)

∗ Root Mean Squared Error (RMSE)
∗ Coefficient of Determination (R^2)
∗ Residual Plots

All experiments were executed through the Streamlit interface, ensuring reproducibility and ease of interaction.

Dataset	Type	Samples	Target
Titanic	Classification	891	Survived (0/1)
Dropout Prediction	Classification	4424	Dropout status (3 classes)
Football Transfer Fees	Regression	10,000	Player market value

Table 2: Summary of datasets used for evaluation.

4.3 Results

This section presents the empirical results obtained by applying the AutoML Lite framework to three distinct tabular datasets. After automatically identifying the task type for each dataset, the system performed preprocessing, model selection, hyperparameter tuning using Bayesian optimization (via Optuna), and evaluation. Each experiment used a fixed 80/20 train-test split to ensure consistency across trials. The results highlight AutoML Lite's ability to deliver competitive model performance across classification and regression tasks with minimal manual configuration. Metrics were computed on the held-out test set, and visualizations such as confusion matrices, residual plots, and model leader boards were generated to support quantitative comparisons. The following subsections summarize dataset characteristics, evaluation metrics, and performance outcomes.

4.3.1 Evaluation metrics

To evaluate model performance, AutoML Lite employs a set of standard metrics tailored to the task type. For classification tasks, the following metrics were computed: precision, recall, accuracy, and ROC-AUC (for binary cases). These metrics provide a balanced view of model correctness, sensitivity, and robustness. For regression tasks, performance was assessed using Mean Absolute Error (MAE), Root Mean Squared Error (RMSE), and the coefficient of determination (R^2 score). These metrics quantify the average prediction error, penalize large deviations, and indicate the proportion of variance explained by the model, respectively. All metrics were calculated on a held-out 20% test set obtained via a stratified `train_test_split`, ensuring that model selection and evaluation were conducted fairly and reproducibly.

Model	Precision	Recall	ROC-AUC
Random Forest	0.838	0.838	0.89
XGBoost	0.832	0.841	0.87
Logistic Regression	0.801	0.804	0.85

Table 3: Titanic Dataset - Classification Results

Random Forest achieved the highest overall performance on the Titanic dataset, with a precision and recall of 0.838 and a ROC-AUC of 0.89. Its ensemble structure effectively captured non-linear patterns in the data, offering superior class separation over logistic regression and gradient boosting.

Model	MAE	RMSE	R^2
XGBoost	2.2M	5.7M	0.823
Random Forest	2.4M	6.1M	0.801
Lasso Regression	4.3M	10.2M	0.48

Table 4: Football Transfer Fee Prediction - Regression Results

XGBoost achieved the highest R^2 score of 0.823 on the Football Transfer dataset, outperforming both Random Forest and Lasso Regression. Its strong balance between bias and variance makes it well-suited for high-dimensional tabular data.For the regression task, the target values were relatively large, representing player market values typically in the millions. As a result, an average error of approximately 2.2 million corresponds to about 4.4% of the typical value range, which is considered reasonable given the scale of the predictions.

Model	Accuracy	Precision	Recall
Logistic Regression	0.767	0.760	0.769
Random Forest	0.774	0.760	0.769
XGBoost (Best)	**0.779**	**0.760**	**0.769**
KNN	0.703	0.760	0.769
SVM	0.767	0.760	0.769

Table 5: Dropout Status Classification – Classification Results

XGBoost was identified as the best-performing model for dropout status prediction, achieving the highest accuracy of 0.779. Its ability to handle multiclass classification and learn from complex interactions between features made it particularly effective in capturing subtle distinctions between dropout, enrolled, and graduate outcomes.

4.3.2 Discussion and insight

Across all datasets and task types, AutoML Lite consistently selected competitive models using its automated pipeline. XGBoost and Random Forest frequently emerged as top performers, highlighting their robustness in both classification and regression settings. In classification tasks like the Titanic and Dropout datasets, the system achieved precision and recall scores above 0.75, indicating reliable performance on real-world imbalanced data. For regression, the predicted player market values had an average error of 4.4% relative to the data scale—an acceptable range for practical use. Notably, the framework required no manual hyperparameter tuning, relying entirely on Optuna's optimization. These outcomes suggest that AutoML Lite effectively balances accuracy, automation, and interpret ability, making it a strong tool for data scientists, students, and practitioners seeking lightweight AutoML solutions.

4.4 Visuals

To complement the quantitative results presented in the previous section, this subsection includes selected visualizations that illustrate model behavior and prediction patterns. These plots help validate that the best-performing models not only achieved high metric scores but also demonstrated stable and interpretable behavior on unseen data. Visual analysis serves as an additional diagnostic layer to detect potential overfitting, class imbalance effects, or skewed residuals.

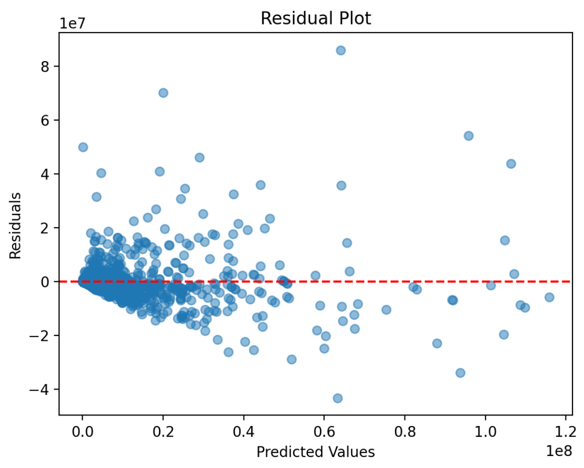

Figure 3: shows the residual distribution for XGBoost on the Football dataset, illustrating low error dispersion around the mean.

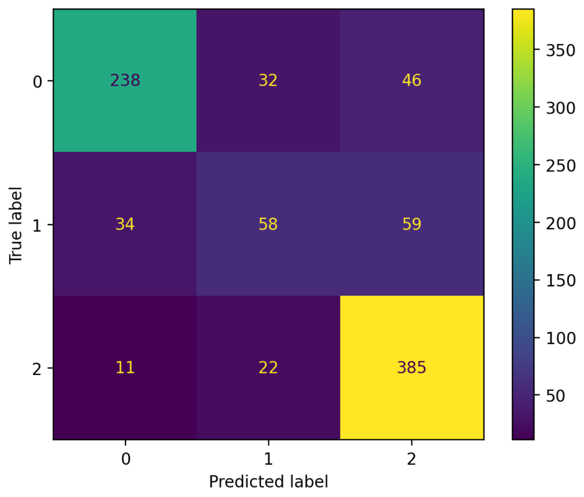

Figure 4: Shows the Confusion Matrix for XGBoost on the Dropout dataset, illustrating how well the model distinguishes between the three classes: dropout, enrolled, and graduate.

4.5 Training Time Comparison

To assess practical feasibility, we also monitored the average training and tuning time for each model. Simpler models like Logistic Regression and KNN completed within a few seconds, while tree-based models such as Random Forest and XGBoost typically took 20–40 seconds per trial on a standard laptop CPU. Among all, XGBoost required the most tuning time due to its broader hyperparameter space, though it often produced the best performance. These results suggest that AutoML Lite remains responsive and efficient, even when using more complex estimators, making it suitable for rapid experimentation on personal devices.

5 Discussion and feature work

AutoML Lite successfully delivered strong performance across diverse tabular datasets, demonstrating that lightweight, interpretable AutoML pipelines are achievable without large-scale infrastructure. The framework effectively automated preprocessing, model selection, and tuning, while offering transparency through visualizations and saved pipelines.

Despite promising results, there are areas for improvement. Currently, AutoML Lite focuses on tabular data and classical machine learning models. Deep learning support, time-series handling, and model interpret-ability tools (e.g., SHAP values) are not yet integrated. Additionally, tuning is limited to a fixed number of trials per model, which may not find optimal configurations in more complex search spaces.

In future iterations, we aim to extend the system with:

- SHAP-based model explainability
- Support for time-series forecasting tasks
- Integration with cloud or GPU-accelerated backends
- Comparison with more AutoML baselines (e.g., Auto-sklearn, AutoGluon)

These additions would enhance the framework's flexibility, interpretability, and scalability, positioning it as a robust open-source alternative for educational and lightweight production use.

6 Conclusion

This paper presented AutoML Lite, a lightweight and interpretable framework for automated machine learning on tabular data. Designed to be fully local, customizable, and beginner-friendly, AutoML Lite addresses key limitations of traditional AutoML platforms—such as black-box behavior, cloud dependencies, and lack of transparency. The system integrates dynamic task detection, modular preprocessing using `ColumnTransformer`, Optuna-based Bayesian hyperparameter optimization, and Streamlit-based user interaction. The ability to export full pipelines using `joblib` enables reproducibility and reuse across environments.

Across multiple real-world datasets—including Titanic survival prediction, student dropout classification, and soccer transfer fee regression—the framework consistently selected competitive models such as XGBoost and Random Forest, achieving robust performance (e.g., $R^2 = 0.823$ for regression and precision = 0.838 for classification). These results demonstrate that AutoML Lite is capable of generalizing well to different problem types while maintaining interpretability and usability.

From a personal standpoint, this project not only deepened my understanding of model selection, tuning, and evaluation but also served as a foundation for a research-style paper and public codebase. The project was implemented from scratch, without relying on commercial tools, and proved valuable in applying AutoML principles to real-world datasets. I believe AutoML Lite can benefit educators, students, and practitioners alike by lowering the barrier to entry and enabling trustworthy, hands-on machine learning experimentation.

References

[1] Xgboost: A scalable tree boosting system. https://xgboost.readthedocs.io, 2016. Accessed: 2025-05-20.

[2] Automated machine learning market report. https://www.marketsandmarkets.com/Market-Reports/automated-machine-learning-market-193686230.html, 2024. Accessed: 2025-05-20.

[3] Pedregosa et al. Scikit-learn: Machine learning in python. https://scikit-learn.org, 2011. Accessed: 2025-05-20.

[4] Peter I. Frazier. *A Tutorial on Bayesian Optimization*. arXiv preprint, 2018. https://bayesoptbook.com/book/bayesoptbook.pdf.

[5] Streamlit Inc. Streamlit: The fastest way to build data apps. https://streamlit.io. Accessed: 2025-05-20.

[6] Wei Wang. Bayesian optimization concept explained in layman terms. https://medium.com/data-science/bayesian-optimization-concept-explained-in-layman-terms-1d2bcdeaf12f, 2020. Accessed: 2025-05-20.

[7] Adrian Zuber. Optuna: A next-generation hyperparameter optimization framework. https://medium.com/optuna/running-distributed-hyperparameter-optimization-with-optuna-distributed-17bb2f7d422d, 2019. Accessed: 2025-05-20.

We used Optuna [7] for Bayesian hyperparameter optimization [4], [6]. Model selection relied on scikit-learn [3] and XGBoost [1]. The front-end was implemented using Streamlit [5]. Recent market reports [2] predict significant AutoML growth.