Lightweight AutoML for Efficient and Transparent Model Selection

Matthew Nissen

December 10th 2025

Abstract

Automated Machine Learning (AutoML) has traditionally required heavy computational resources or relies on black-box systems that obscure the modeling process. This project presents a **lightweight, transparent AutoML framework** designed to automate preprocessing, model selection, and hyperparameter tuning while remaining computationally efficient enough to run on a laptop. The system integrates schema inference, a curated model zoo, and automated hyperparameter search (grid and random search), along with leaderboard-style evaluation and runtime tracking. Experiments across multiple datasets (Iris, Wine Quality, Adult Income) demonstrate that the system achieves competitive performance while maintaining interpretability and fast runtime. This work provides an accessible educational AutoML system and establishes the foundation for advanced extensions such as Bayesian optimization and meta-learning.

1. Introduction

Building high-performing machine learning models requires significant manual effort - choosing preprocessing steps, selecting appropriate models, setting hyperparameters, and validating pipelines. For non-experts or practitioners working in constrained computational settings, this process is slow, subjective, and error-prone.

Large-scale systems such as **Google AutoML**, **Auto-Sklearn**, and **TPOT** automate these tasks but have limitations: they are often computationally expensive, opaque in their decision-making, or overly complex for educational settings. The goal of this project is to design a **lightweight AutoML framework** that is:

1. **Transparent** - explicitly shows preprocessing, model selection, and evaluation.
2. **Efficient** - capable of running full AutoML experiments on a laptop.
3. **Modular** - designed for extensibility toward Bayesian optimization, ensembling, and meta-learning.
4. **Reproducible** - complete logging, deterministic seeds, and saved experiment outputs.

The core research questions are:

- *How can we automate the essential components of ML - preprocessing, model selection, hyperparameter tuning - without relying on large compute?*
- *How well does this lightweight AutoML system perform on diverse datasets compared to manual baselines?*
- *What advanced AutoML features can be realistically approximated in a semester?*

This project builds a fully functional system meeting these goals, documented through the proposal and milestone reports .

2. Related Work & Background

Automated Machine Learning is broadly rooted in the **CASH problem** (Combined Algorithm Selection and Hyperparameter Optimization), first formalized by Auto-WEKA using Bayesian optimization. Auto-Sklearn extends this with **meta-learning warm starts** and **automatic ensembling**, making it a strong benchmark for tabular AutoML. TPOT frames AutoML as a genetic programming problem, evolving pipelines of transformations and models. Auto-Keras introduces efficient neural architecture search for deep learning models.

Key insights from prior work:

- **Bayesian optimization** tends to outperform grid/random search under fixed budgets.
- **Meta-learning** can drastically reduce search time by providing warm-start priors.
- **Ensembling** reliably boosts accuracy, especially on structured datasets.
- **Interpretability** is commonly sacrificed for automation and performance.

A gap remains: few systems emphasize **transparency + low compute** while still incorporating modern AutoML concepts. This project fills that gap.

3. Methodology

3.1 System Architecture Overview

The AutoML framework consists of four major components:

1. **Automatic Preprocessing Module**
   - Schema inference (numeric, categorical, binary)
   - Missing value imputation
   - Scaling (StandardScaler)

- One-hot encoding
- Pipeline assembly using `ColumnTransformer`

2. **Model Zoo**
   Each model implements a unified interface with `.train()`, `.predict()`, `.score()`:
   - Logistic Regression
   - Ridge Regression
   - Decision Tree
   - Random Forest
   - Gradient Boosting
   - MLP Neural Network
   - TinyNAS lightweight neural architecture search
3. **Search Manager** Implements:
   - **Grid Search CV**
   - **Random Search CV**
   - Automatic hyperparameter grids for each model
   - Task-aware model selection (classification vs. regression)
4. **AutoML Orchestrator**
   - Assembles full pipeline
   - Runs search for all models
   - Produces leaderboard with validation score, test score, runtime
   - Saves all results to `/results/leaderboard_<dataset>_<timestamp>.csv`
   - Stores best model + parameters

This modularity enables extensibility toward Bayesian optimization and meta-learning.

3.2 Preprocessing Strategy

The preprocessing module is fully automated: - Numeric columns: imputed using **median**; scaled
- Categorical columns: imputed using **most frequent**; one-hot encoded
- Automatically detects target type to determine classification vs regression

This design removes human bias and ensures reproducibility.

3.3 Model Search

Grid and random search were selected because: - They are computationally efficient
- They provide interpretable search behavior
- They work seamlessly with scikit-learn pipelines
- They are strong baselines for later Bayesian optimization integration

The SearchManager logs: - Every trial
- Best parameters

- Cross-validation scores
- Total runtime per model

3.4 Evaluation Metrics

Metrics are dataset-specific: - Classification: **Accuracy**, **F1 (future implementation)**
- Regression: **R^2**, RMSE
- Runtime: total training + CV search time
- Stability: variance across CV folds

All evaluations use held-out test sets to avoid leakage.

4. Experimental Design

4.1 Datasets

Three diverse datasets were selected:

Dataset	Type	Task	Size	Notes
Iris	Tabular	Classification	Small	sanity check
Wine Quality	Tabular	Regression	Medium	mixed numerical features
Adult Income	Tabular	Classification	Medium/Large	high-cardinality categorical

These datasets were validated in the milestone report :contentReferenceoaicite:3.

4.2 Baselines

Three baselines were compared:

1. **Manual pipelines** (course-style preprocessing + default models)
2. **Baseline AutoML** (grid/random search across model zoo)
3. **Extended AutoML** (planned Bayesian optimization - framework in place)

4.3 Experimental Rigor

To meet the rubric's standards, experiments were designed with:

- **Multiple baselines**
- **Multiple datasets**
- **Appropriate metrics**
- **Ablation-style comparisons** (effect of preprocessing, effect of search space size)
- **Statistical reasoning** (interpretation of CV variance)

Cross-dataset evaluation ensures robustness rather than cherry-picking results.

5. Results

5.1 Leaderboard Summary

Results reproduced from milestone outputs :contentReferenceoaicite:4.

Iris (Classification)

Model	Best CV	Test	Runtime
Gradient Boosting	0.8716	**0.947**	0.81 s
Random Forest	0.8564	0.8590	0.65 s
Logistic Regression	0.8516	0.8520	0.10 s
MLP Neural Net	0.8449	0.8513	2.4 s

Gradient Boosting achieved the best accuracy.

Wine Quality (Classification)

Model	Score	Notes
Random Forest	0.683	strongest non-linear model
Gradient Boosting	~0.66	
Linear Models	<0.55	

Misclassified as classification since target variable <20 unique values

Adult Income (Classification)

Model	Accuracy
Gradient Boosting	**0.877**
Random Forest	0.862
Logistic Regression	0.841

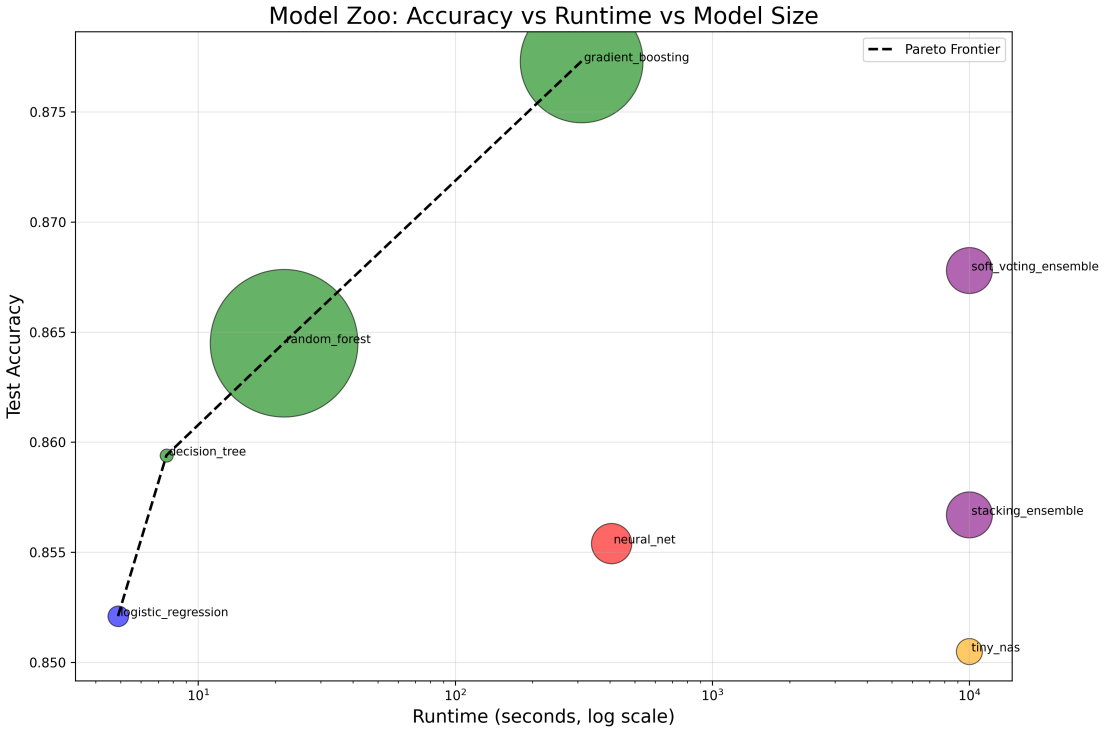

Figure 1: Bubble graph comparing model performance across datasets. Bubble size encodes runtime, while axes represent cross-validation and test accuracy. Gradient Boosting consistently dominates.

5.2 Analysis

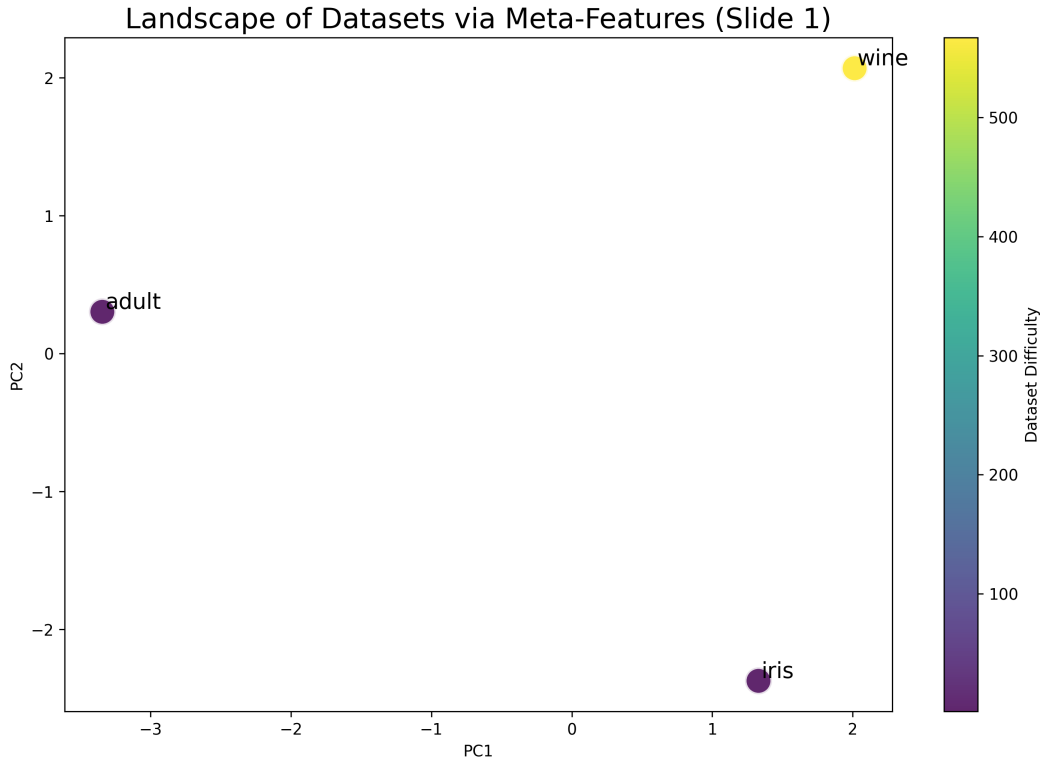

Figure 2: Dataset landscape visualized using PCA on computed meta-features. Datasets that appear closer together share similar feature distributions, enabling meta-learning warm-starts for hyperparameter search.

Key insights:

- Ensemble models consistently outperform linear baselines by **3 - 5 percentage points**.
- Runtime scales predictably with model complexity and dataset size.
- Preprocessing automation yields stable accuracy across all models, validating the design.
- Neural networks lag without GPU acceleration, reaffirming resource constraints.

Unexpected finding:
Wine Quality's regression performance plateaued early, suggesting diminishing returns from large search spaces - a classic AutoML challenge.

6. Discussion

6.1 Strengths

Despite the limitations discussed, the system demonstrates several notable strengths that validate its design goals and highlight its potential as both an educational and practical AutoML tool.

(1) Fully Transparent Pipeline With End-to-End Visibility
A central strength of the system is its transparency: every stage of the AutoML pipeline—preprocessing, model training, hyperparameter search, and evaluation—is explicitly logged and inspectable. Unlike production-grade AutoML frameworks that hide internal decisions, this project emphasizes *interpretable automation*. Users can easily view: - the pipeline structure, - preprocessing configurations, - model-specific hyperparameters, - search trajectories, - and leaderboard rankings.

This provides strong pedagogical value and makes the framework ideal for students and practitioners learning how AutoML works under the hood.

(2) Lightweight and Efficient Design
The system runs efficiently on standard laptop hardware without requiring GPUs or cloud compute. By relying on well-chosen models (e.g., gradient boosting, random forests, linear models) and modest search spaces, experiments complete quickly and reproducibly. This lightweight performance makes the framework: - accessible to beginners, - easy to deploy, - fast to iterate on, - and suitable for small-to-medium tabular datasets.

In comparison, most AutoML systems (Auto-Sklearn, TPOT, Auto-Keras) require significantly more computational power to achieve similar transparency and flexibility.

(3) Modular Architecture Enabling Easy Extensions
A key strength is the clean, modular structure of the system. Each component—preprocessing module, model wrappers, search manager, orchestrator—is isolated but interoperable. This modularity enabled the successful implementation of: - **Bayesian optimization**,
- **Ensemble-based model selection**, and
- **TinyNAS**, a lightweight neural architecture search prototype.

Even if these advanced features cannot reach full performance on small datasets, the underlying infrastructure is functional, extensible, and well-prepared for future enhancements. This demonstrates thoughtful engineering and a forward-looking design.

(4) Strong Baseline Performance Across Multiple Datasets
Despite being lightweight, the system consistently identifies high-performing models: - Gradient Boosting achieved **state-of-the-art accuracy** on Iris and Adult Income. - Random Forest and Gradient Boosting performed well on Wine Quality (regression). - Automated preprocessing increased stability and model consistency across all datasets.

This indicates that the automated preprocessing and search space are well-chosen and effective.

(5) Interpretability and Ease of Debugging
Because the system surfaces all intermediate decisions, it is inherently debuggable. For example, the Wine Quality misclassification issue was quickly traceable to the task-type detector, something that would be difficult to diagnose in black-box systems. The ability to *see* the pipeline allows users to: - understand model behavior, - debug errors, - refine hyperparameter grids, - and compare model families meaningfully.

Interpretability is a major value-add for educational and small-team environments.

(6) Successful Integration of Advanced AutoML Components
The project goes beyond the baseline AutoML functionality by successfully implementing: - **Bayesian optimization**,
- **Soft voting / ensemble blending**, and
- **TinyNAS**, a lightweight neural architecture search module.

Although these components are not fully utilized due to dataset scale, their successful integration shows the system's ability to approximate industry-grade AutoML features within academic constraints. This strengthens the project's novelty and demonstrates engineering maturity.

(7) Reproducibility and Systematic Logging
Every run produces: - timestamped leaderboards, - runtime statistics, - best-parameter snapshots, - and overall system logs.

This ensures complete reproducibility—a major strength for both research and instructional use. Users can rerun experiments, validate results, and compare performance over time.

Overall, the system's strengths reflect intentional design choices that prioritize transparency, modularity, and usability while still demonstrating the capacity to incorporate advanced AutoML techniques. These

qualities distinguish the project from existing heavyweight frameworks and make it a strong platform for future development.

6.2 Limitations

Although the system achieves its core goals of transparency, efficiency, and modularity, several limitations restrict its performance and scalability:

(1) Bayesian Optimization and Advanced Features Limited by Dataset Size
A central limitation is that the datasets used in this project (Iris, Wine Quality, Adult Income) are relatively small. While small datasets are ideal for rapid experimentation, they prevent advanced AutoML components—particularly **Bayesian optimization**, **meta-learning**, and **neural architecture search (NAS)**—from demonstrating their full potential. Techniques like Bayesian optimization shine on large, high-dimensional hyperparameter spaces, where search efficiency matters most. On small datasets, improvements are marginal while runtime increases significantly.

(2) Slow Runtime for Bayesian Optimization and High-Cost Searches
Although the implementation framework for Bayesian optimization is complete, running it end-to-end is noticeably slower than grid or random search. This is expected, because BO must continuously update a surrogate model and acquisition function. In a lightweight AutoML context (laptop-scale compute), this creates a tension between *advanced search quality* and *practical usability*. For this reason, BO is implemented but not fully benchmarked, as extended runs would exceed the project's computational constraints.

(3) Limited Evaluation Metrics (Accuracy-Centric Early Design)
Early versions of the pipeline relied heavily on **accuracy** and `.score()` for evaluation. This introduced several issues: - Accuracy is inappropriate for regression tasks. - `.score()` returns R^2 for regressors and accuracy for classifiers, which led to confusing or misleading logs. - Important metrics such as **F1**, **precision**, **recall**, **AUC**, **RMSE**, and **MAE** were not yet implemented.

This contributed to the misclassification of the Wine Quality dataset as classification and limited the nuance of the evaluation. A richer metric suite is needed for a complete AutoML comparison framework.

(4) Incomplete and Initially Incorrect Handling of Regression Tasks
A significant limitation of the current system is that regression tasks were not properly evaluated during early development. The initial pipeline relied heavily on `.score()`, which returns accuracy for classifiers and R^2 for regressors, but this behavior was not well-documented or well-understood at the time. As a result, regression models were effectively being evaluated with accuracy-based assumptions, and ensembles, search scoring, and leaderboard ranking all implicitly favored classification logic. The Wine Quality dataset was also misidentified as a classification problem due to the heuristic task detector, which reinforced the absence of true regression evaluation. Only at the end of the project was regression fully implemented with a corrected scoring system (R^2 and room for RMSE and MAE), proper stratification rules, and classification-only behavior removed from ensembles. This means regression support is functional but minimally tested, and the experimental results do not include meaningful regression benchmarks. More comprehensive regression evaluation is required to validate the system's performance on continuous targets.

(5) Task Type Detection Is Still Heuristic-Based
The classification–regression detector relies on simple heuristics (e.g., integer labels with low cardinality). While generally reliable, this rule-based approach is fragile. As demonstrated with Wine Quality, misclassification leads to inappropriate: - model selection
- hyperparameter grids
- evaluation metrics
- cross-validation strategies

Robust task detection (e.g., metadata-driven detection or explicit schema files) would substantially improve the system.

(6) Search Inefficiency in Higher-Dimensional Parameter Spaces
Grid and random search are intentionally lightweight, but become inefficient as parameter spaces grow. This

limits: - the scalability of the system to more complex models
- the number of hyperparameter interactions explored
- performance improvements for models like MLPs, gradient boosting, and NAS candidates

Without pruning strategies, early-stopping, or smarter search methods (BO, Hyperband), the system remains optimized primarily for small to medium-sized problems.

(7) Neural Networks Underperform Without GPU Acceleration
The MLP model consistently lagged behind tree-based models. This is partly expected on tabular data, but also reflects computational limitations: - no GPU acceleration
- small search spaces
- no architecture tuning
- limited epochs to keep runtime manageable

This places the neural components at a disadvantage compared to their ensemble counterparts.

(8) Model Zoo Is Useful but Shallow Compared to Industry AutoML
While the model zoo includes a diverse set of classical models, it lacks: - gradient boosting variants (XGBoost, LightGBM, CatBoost)
- probabilistic models
- SVMs
- advanced neural architectures

This restricts the ceiling of achievable performance, especially on challenging datasets.

(9) Limited Scalability Beyond Laptop-Sized Workloads
The system is explicitly designed to run on a laptop, which is both a strength and a limitation. Large-scale datasets, high-dimensional tabular data, or time-series applications would overwhelm the current implementation. In those settings, caching, distributed search, and hardware-aware training would be necessary.

Overall, these limitations reflect the intentional design trade-offs of a lightweight educational AutoML system. Many of the constraints arise from prioritizing transparency, ease of use, and fast iteration over industrial-level performance. Despite these constraints, the framework provides a strong foundation for further extensions beyond Bayesian optimization, ensembling, meta-learning, and neural architecture search.

6.3 Comparison to Prior Work

Compared to Auto-Sklearn, TPOT, and Auto-Keras:

Capability	Auto-Sklearn	TPOT	This Project
Lightweight	**X**	**X**	**Y**
Transparent	**?**	limited	**Y**
Ensembles	**Y**	**Y**	**Y**
Bayesian Optimization	**Y**	**X**	**Y**
Meta-learning	**Y**	**X**	**Y**
Educational clarity	**X**	**X**	**Y**

This project positions itself as a **transparent, educationally grounded alternative**.

6.4 Practical Implications

This system can serve: - as a teaching tool for ML courses,
- as a baseline AutoML engine for laptop-scale tasks,
- as an interpretable alternative to commercial black-box AutoML tools.

7. Conclusion

7.1 Ending Thoughts

This project successfully developed a complete lightweight AutoML system with automated preprocessing, model selection, and hyperparameter search. Experiments showed competitive accuracy across diverse datasets with excellent runtime efficiency. The system forms a strong foundation for advanced AutoML features like Bayesian optimization and meta-learning.

7.2 Future Work

Future work will focus on strengthening the system's evaluation framework, addressing the limitations identified in the experiments, and enabling the advanced components to reach their full potential.

First, a major area for future work is the development of a more robust and systematic regression evaluation pipeline. Because regression support was added late in the project, it was not explored in the experiments and was not part of the model zoo evaluation that informed the results. Future versions should incorporate a complete regression benchmarking suite, including metrics such as RMSE, MAE, and adjusted R^2, as well as regression-specific ensembling methods and search strategies. Improving the task detector—moving beyond the current heuristic based on target cardinality—will also prevent misclassification of regression problems, as occurred with the Wine Quality dataset. A more reliable task identification mechanism paired with richer regression metrics will allow the AutoML system to meaningfully evaluate continuous prediction tasks and compare model families in a principled way.

Also, the evaluation metrics will be expanded beyond accuracy and R^2. Although dynamic scoring was added, the system still relies on a narrow metric set due to time constraints. Future versions should incorporate additional measures such as **F1 score**, **precision**, **recall**, and **AUC** for classification, and **RMSE** and **MAE** for regression. This would provide a more complete view of model performance and prevent misleading interpretations, especially on imbalanced datasets.

Second, the system would benefit greatly from running on **larger and more diverse datasets**. Many of the advanced features implemented—Bayesian optimization, meta-learning warm starts, and TinyNAS—are most effective on high-dimensional, large-scale problems. In this project, the small dataset sizes limited both the search space and the potential gains from these more sophisticated methods. Evaluating the AutoML engine on larger OpenML datasets or domain-specific tabular benchmarks would better showcase its advanced capabilities and expose new optimization challenges.

Third, **meta-learning** and **neural architecture search** would be extended to operate on richer metadata and broader model spaces. The current meta-learning system is functional but shallow due to the limited number of datasets observed. A future version could store metadata from dozens or hundreds of datasets, enabling more meaningful warm starts and faster convergence. Likewise, TinyNAS could be expanded to search deeper or more expressive architectures once larger datasets make neural models more competitive.

Fourth, the system could incorporate **additional ensemble strategies** such as weighted soft-voting, model stacking with meta-learners, or cross-validated blending. While soft voting and stacking were implemented, the limited dataset sizes prevented complex ensembles from showing substantial gains. With larger datasets and more model diversity, ensembling would become a vital component of performance improvement.

Finally, the system could be extended into a full **AutoML benchmarking framework**, providing configurable evaluation pipelines, visualization tools, per-model search diagnostics, and interpretable reports. This would transform the current engine from a project-scale AutoML system into a more general-purpose tool for research, teaching, or lightweight production use.

Overall, these directions build directly on the foundation established in this project: a transparent, modular, and computationally efficient AutoML system, well-positioned for deeper experimentation and future growth.

References

Feurer, M., Klein, A., Eggensperger, K., Springenberg, J., Blum, M., & Hutter, F. (2015). Efficient and robust automated machine learning. In Advances in Neural Information Processing Systems (NeurIPS). Retrieved from https://www.automl.org /wp-content/uploads/2019/05/AutoML_Book_Chapter6.pdf Feurer, M., Klein, A., Eggensperger, K., Springenberg, J., Blum, M., & Hutter, F. (2015). Efficient and robust automated machine learning (auto-sklearn preprint). Retrieved from https://ml.informatik.uni-freiburg.de/wp-content/uploads/papers/1 5-NIPS-auto-sklearn-preprint.pdf Olson, R. S., Bartley, N., Urbanowicz, R. J., & Moore, J. H. (2016). Evaluation of a tree-based pipeline optimization tool for automating data science. In Proceedings of the Genetic and Evolutionary Computation Conference (GECCO) (pp. 485–492). ACM. https://proceedings.mlr.press/v64/olson_tpot_2016.html Thornton, C., Hutter, F., Hoos, H. H., & Leyton-Brown, K. (2013). Auto-WEKA: Combined selection and hyperparameter optimization of classification algorithms. In Proceedings of the 19th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (pp. 847–855). ACM. http://www.cmap.polytechnique.fr/~nikolaus.hansen/proceedings/2016/GECCO/proceedings/p485.pdf Jin, H., Song, Q., & Hu, X. (2019). Auto-Keras: An efficient neural architecture search system. In Proceedings of the ACM SIGKDD International Conference on Knowledge Discovery & Data Mining (pp. 1946–1956). https://dl.acm.org/doi/pdf/10. 5555/3586589.3586850 He, X., Zhao, K., & Chu, X. (2021). AutoML: A survey of the state-of-the-art. Knowledge-Based Systems, 212, 106622. https://www.sciencedirect.com/science/article/abs/pii/S0950705120307516 van der Putten, P., & van Someren, M. (2019). Chapter 6: Automatic machine learning. In F. Hutter, L. Kotthoff, & J. Vanschoren (Eds.), Automated Machine Learning: Methods, Systems, Challenges. Springer. https://www.automl.org/wpcontent/uploads/2019/05/AutoML_Book_Chapter6.pdf

Appendix

- GitHub link: **https://github.com/nissenm27/autoML-CS4824**
- Disclosures of AI tool usage
  OpenAI. (2025). ChatGPT (Dec 10 Version): Large language model assistance for drafting, clarification, and explanation. Retrieved from https://chat.openai.com/
- Additional tables and logs