SELA: Tree-Search Enhanced LLM Agents for Automated Machine Learning

Anonymous ACL submission

Abstract

Automated Machine Learning (AutoML) approaches encompass traditional methods that optimize fixed pipelines for model selection and ensembling, as well as newer LLM-based frameworks that autonomously build pipelines. While LLM-based agents have shown promise in automating machine learning tasks, they often generate low-diversity and suboptimal code, even after multiple iterations. To overcome these limitations, we introduce Tree-**S**earch **E**nhanced **L**LM **A**gents (**SELA**), an innovative agent-based system that leverages Monte Carlo Tree Search (MCTS) to optimize the AutoML process. By representing pipeline configurations as trees, our framework enables agents to conduct experiments intelligently and iteratively refine their strategies, facilitating a more effective exploration of the machine learning solution space. This novel approach allows SELA to discover optimal pathways based on experimental feedback, improving the overall quality of the solutions. In an extensive evaluation across 20 machine learning datasets, we compare the performance of traditional and agent-based AutoML methods, demonstrating that SELA achieves a win rate of 65% to 80% against each baseline across all datasets. These results underscore the significant potential of agent-based strategies in AutoML, offering a fresh perspective on tackling complex machine learning challenges.

1 Introduction

Automated Machine Learning (AutoML) is a rapidly evolving field that seeks to automate the process of designing reliable machine learning solutions with minimal human intervention. Traditional AutoML frameworks, such as Auto-WEKA (Thornton et al., 2013), Auto-Sklearn (Feurer et al., 2015, 2020), AutoGluon (Tang et al., 2024b), and H2O AutoML (LeDell and Poirier, 2020), rely on predefined search spaces and routines. These frameworks primarily focus on optimizing hyperparameters and model ensembling to find the best model configuration.

Recently, large language model (LLM)-based agents have emerged as promising tools for automating machine learning tasks by leveraging natural language processing capabilities to generate code. These systems typically begin with a natural language prompt describing the dataset and the problem, after which an LLM generates an end-to-end solution. Early efforts, such as (Zhang et al., 2024), experimented with prompting LLMs to generate machine learning solutions, while (Hong et al., 2024a) introduced agents equipped with hierarchical graph modeling and programmable node generation to address complex and dynamic workflows. Despite these advances, LLM-based solutions often fall short in generating diverse and highly optimized workflows, as their search process remains limited to a single pass or trial. Without iterative refinement or the ability to explore alternative strategies, these solutions often converge to suboptimal results, even when multiple attempts are allowed.

To address the limitations of both traditional AutoML and LLM-based frameworks, we examine the problem-solving strategies of human experts. Unlike traditional AutoML, which relies on predefined search space and prebuilt pipelines, experts dynamically construct workflows by deriving task-specific insights from data. Furthermore, rather than settling on a single attempt, they iteratively experiment, analyze outcomes, and refine each component based on feedback. This adaptive, feedback-driven process enables the exploration of diverse solutions and continuous optimization—an approach that current LLM-based frameworks struggle to replicate.

We propose Tree-**S**earch **E**nhanced **L**LM **A**gents (**SELA**), a novel framework that integrates LLM agents with a dynamic, feedback-driven search pro-

	Dynamic Pipeline	Feature Engineering	Model Training	Model Improvement	Pipeline Optimization
AutoGluon (Erickson et al., 2020)	✗	✗	Fixed models	Multi-layer stacking + bagging	✗
AutoSklearn (Feurer et al., 2020)	✗	✗	Fixed models	Bayes Opt. + meta-learning + ensemble	✗
Data Interpreter (Hong et al., 2024a)	✓	Instinctive	Instinctive	Instinctive	✗
AIDE (Jiang et al., 2025)	✓	Instinctive	Dynamic & diverse	Dynamic & diverse	One-step refinement + LLM
SELA (Ours)	✓	Dynamic & diverse	Dynamic & diverse	Dynamic & diverse	Stepwise MCTS + LLM

Table 1: Comparison of key capabilities across various AutoML methods. *Dynamic* indicates the system's ability to adjust workflows based on intermediate outcomes, allowing it to adapt as new information emerges. *Diverse* refers to employing multiple strategies or methods across tasks, which helps capture varied modeling needs. *Instinctive* means that the system directly relies on the decisions generated by an LLM and heavily depends on the model's inclination.

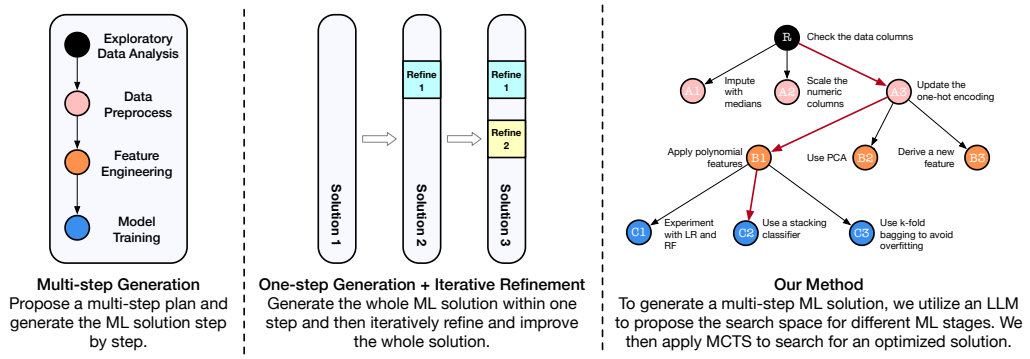

Figure 1: SELA's abstraction compared to other agent-based AutoML frameworks. There are two main types of agent-based approaches to AutoML problems. The first approach (Hong et al., 2024a) divides a machine learning task into multiple stages, proposing a plan for each stage, and generating and executing code step by step according to the plan, with no refinement after the solution is completed. The second (Jiang et al., 2025) generates the entire solution in one step and iteratively refines it as a whole. SELA integrates both approaches, enabling stage-wise planning while iteratively exploring better solutions at each stage level.

cess for automated machine learning. Our framework combines a task-specific insight proposer, which dynamically explores machine learning configurations, with a modified Monte Carlo Tree Search (MCTS) selection algorithm, designed to efficiently navigate complex search spaces by prioritizing deeper levels earlier. As shown in Figure 1, SELA combines stage-wise planning, where each phase (e.g., Exploratory Data Analysis, Data Preprocessing, Feature Engineering, and Model Training) is handled sequentially, with iterative refinement to continuously improve solutions. Through this combination, SELA iteratively optimizes machine learning workflows, much like experts refining their approach based on feedback and experimentation.

We rigorously evaluated SELA using 20 diverse datasets from the AutoML Benchmark (Gijsbers et al., 2024), comparing its performance against both traditional AutoML systems and agent-based AutoML approaches. The results demonstrate that SELA consistently delivers superior performance across a wide range of machine learning tasks, validating its effectiveness and adaptability.

To summarize, our research makes the following contributions:

1. We introduce a feedback-driven, task-specific insight proposer that enables LLM agents to dynamically explore machine learning configurations, iteratively optimizing solutions across multiple experimental rounds.

2. We propose a modified MCTS selection algorithm that adapts to computation-heavy scenarios, prioritizing the exploration of deeper levels of the search tree earlier to enhance efficiency and solution quality.

3. We compare agent-based and traditional AutoML approaches, highlighting the flexibility and performance advantages of agentic methods in machine learning.

2 Related Works

Tree Search and Its Integration with LLMs Tree search algorithms have significantly advanced problem-solving in artificial intelligence, with Monte Carlo Tree Search (MCTS) emerging as a

leading technique. These algorithms have been successfully applied across various domains, including robotics (Wu et al., 2015; Clary et al., 2018; Best et al., 2019), chemistry (Segler et al., 2018), and gaming (Silver et al., 2016, 2017), where MCTS is used to navigate vast solution spaces and solve complex problems. More recently, research has focused on integrating tree search with Large Language Models (LLMs) to enhance reasoning and decision-making. Studies such as Krishnamurthy et al. (2024) and Dwaracherla et al. (2024) explored LLMs' capacities for efficient exploration, while Tang et al. (2024a) and Hui and Tu (2024) developed strategies for exploiting previously learned knowledge. Zhou et al. (2024) and Chi et al. (2024) applied MCTS for planning with external or self-evaluated feedback, while Feng et al. (2023); Wang et al. (2024a) adapted AlphaZero-style tree search to LLM-based tasks. However, the combination of LLMs and MCTS can be computationally intensive and time-consuming. Investigating more efficient strategies for using MCTS with LLMs presents a promising direction.

Advances and Limitations in AutoML Systems Automated Machine Learning (AutoML) frameworks were introduced to reduce the need for expert knowledge in designing machine learning pipelines. Early AutoML efforts, such as (Thornton et al., 2013; Olson and Moore, 2016; Jin et al., 2019; Feurer et al., 2020; Erickson et al., 2020; LeDell and Poirier, 2020; Wang et al., 2021; Jin et al., 2023; Tang et al., 2024b), focused primarily on automating key pipeline components like hyperparameter optimization, model selection, stacking, and ensembling. These frameworks achieved notable progress by integrating meta-learning and hyperparameter search strategies to automatically select and tune machine learning models.

Recently, there has been growing interest in leveraging LLMs within AutoML systems to enhance pipeline flexibility. Studies such as (Hollmann et al., 2024; Li et al., 2024) applied LLMs to automate feature engineering, while Liu et al. (2024) introduced LLMs for hyperparameter tuning. In addition, Luo et al. (2024) proposed embedding LLMs at each stage of the machine learning workflow. Despite these advancements, traditional AutoML systems, which consist of pre-defined search space and procedure, face challenges in adapting to unique datasets or specific task requirements.

LLM Agents for Dynamic ML Pipelines LLM-based agents provide dynamic solutions for complex machine learning tasks. Hong et al. (2024a,b) introduced LLM agents with hierarchical graph modeling and programmable node generation, creating adaptable pipelines for diverse data scenarios. Zhang et al. (2024) showed LLMs' ability to interpret structured inputs and apply past experiences to new tasks. Guo et al. (2024) introduced a data science agent leveraging case-based reasoning but struggled with generating solutions from scratch due to reliance on existing codebases. Jiang et al. (2025) proposed an iterative approach where pipelines are generated and refined incrementally.

SELA combines stage-wise planning and iterative refinement, allowing autonomous exploration and generation of machine learning solutions from the ground up. This provides greater flexibility and control, enabling optimized solutions at each stage. Table 1 compares the functionalities of various AutoML systems.

3 Method

As illustrated in Figure 2, SELA consists of three key components: an LLM-based insight proposer, an MCTS-based search module, and an LLM agent for executing experiments. First, the LLM generates insights from the problem description and dataset, defining a search space. The search module then organizes this space into a tree structure and uses MCTS to explore promising paths. At each cycle, the selected path is passed to the LLM agent, which translates it into an executable pipeline, conducts the experiment, and feeds back the results to refine future searches. This iterative process continues until the termination criterion is met. We show the pseudo code in Algorithm 1 and explain each component in the following sections.

3.1 Insight Proposal and Search Space Creation

To explore various machine learning strategies, SELA employs an insight proposer that generates diverse methods tailored to different stages of the workflow. Each insight suggests a technique or combination of methods aimed at improving performance, such as feature engineering or model training strategies. The proposer takes as input the problem description p and dataset d, generating insights λ for each stage of the process, which are stored in an insight pool Λ. We break down the

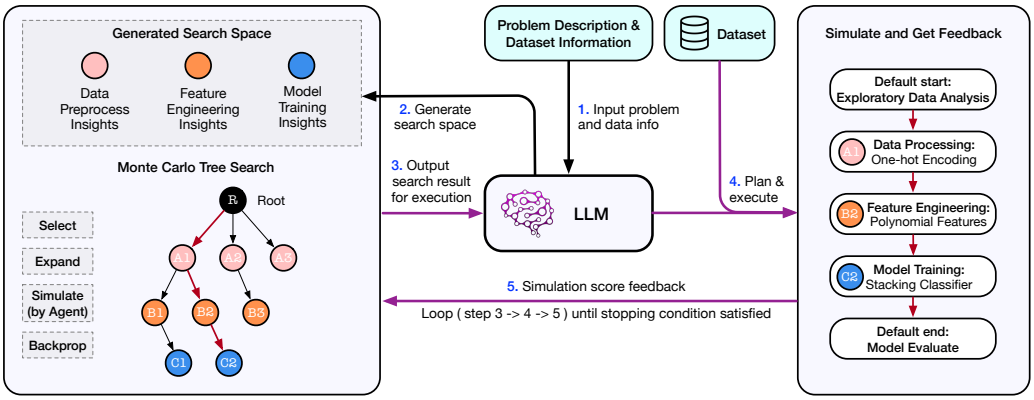

Figure 2: SELA's pipeline operates as follows: The system begins by inputting the problem description and dataset information into the LLM, which generates a search space of potential solutions, encompassing data preprocessing, feature engineering, and model training. The search module, powered by Monte Carlo Tree Search (MCTS), explores this space by selecting, expanding, and simulating potential configurations. The LLM agent then simulates the selected configuration by planning, coding, and executing the experiment. Feedback from the simulation is fed back into the search module, where it is used in the backpropagation step to refine future searches. This iterative process continues until a predefined stopping criterion is met, resulting in an optimized experimental pipeline.

process into five stages: Exploratory Data Analysis (τ_1), Data Preprocessing (τ_2), Feature Engineering (τ_3), Model Training (τ_4), and Model Evaluation (τ_5), collectively denoted as T.

$$\text{InsightProposer}(p, d, M) \rightarrow \Lambda \tag{1}$$

$$\Lambda := \{\lambda_i^\tau \mid \tau \in T, i = 1, \ldots, m\} \tag{2}$$

Figure 3 shows how the insight proposer uses dataset information to generate a task-specific insight, which is then converted into a feature engineering code snippet.

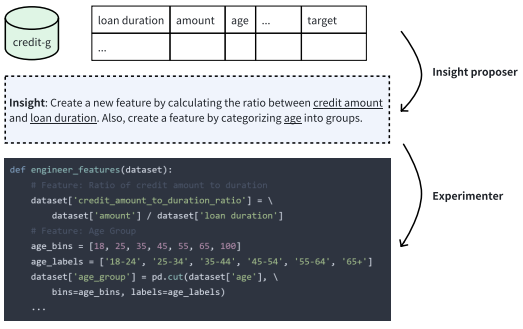

Figure 3: An example of a task-specific feature engineering insight generated by the insight proposer using the credit-g dataset, along with the corresponding code produced by the experimenter.

3.2 Pipeline Execution and Code Generation

We use an LLM agent, called the experimenter E, to conduct each trial by constructing experimental pipelines from natural language requirements. The agent follows two main steps. First, given an experiment configuration c, a set of insights provided by the search module (described in Section 3.3.2), the agent translates these insights into a detailed plan, consisting of task instructions $I^{\tau \in T}$ for each stage of the machine learning process. This step is referred to as E_{plan}.

Next, the agent writes and executes code σ^τ for each task τ based on the instructions I^τ, producing a complete set of code $\sigma^{\tau \in T}$ for the full pipeline and a final execution score s. The combined code outputs $\sigma^{\tau \in T}$ form the full solution σ_{sol} to the problem. This phase is referred to as $E_{\text{code \& execute}}$.

$$E_{\text{plan}}(p, d, c, M) \rightarrow I^{\tau \in T} \tag{3}$$

$$E_{\text{code \& execute}}(I^{\tau \in T}, D, M) \rightarrow (\sigma^{\tau \in T}, s) \tag{4}$$

3.3 Tree Search in Machine Learning Experiments

The search space is modeled as a hierarchical tree, where each node represents an experiment configuration. We use Monte Carlo Tree Search (MCTS) to explore and identify the best solution, balancing exploration and exploitation across different stages. The search process includes selection, expansion, simulation, and backpropagation, as outlined below:

3.3.1 Experiment Node

Each node x in the tree represents an insight λ and contains several attributes: the insight $\lambda(x)$ corresponding to the method or strategy for the pipeline stage, the depth $\delta(x)$ indicating the pipeline stage

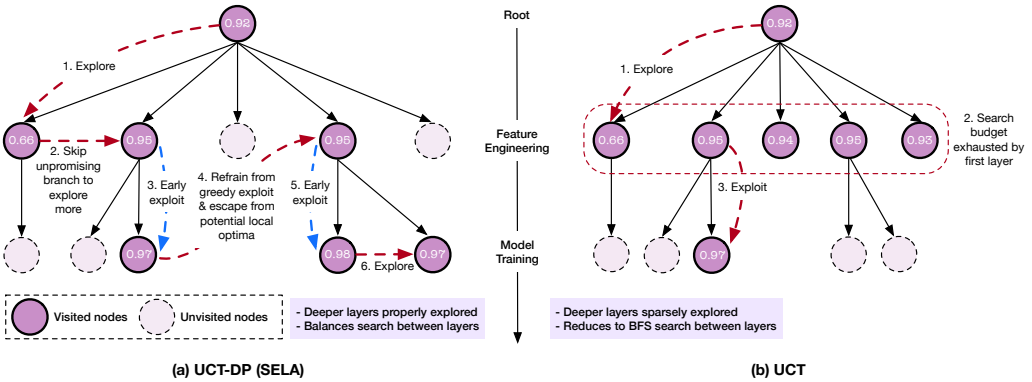

Figure 4: Mechanism of UCT-DP and search behavior of SELA. Our proposed UCT-DP algorithm offers key advantages over vanilla UCT, enabling more efficient exploration in machine learning scenarios where search budgets are constrained by high computational costs. With only a few rollouts (e.g., no more than 10), standard UCT selection degenerates into a breadth-first search (BFS), as unvisited nodes are given infinite priority. Consequently, deeper layers remain unexplored until every node in the current layer has been visited at least once, leading to sparse traversal of deeper nodes. In contrast, UCT-DP retains the core properties of standard UCT (red dotted lines) while crucially activating them earlier in the search process, mitigating this limitation through early exploitation (blue dotted lines). This allows SELA to assess deeper layers more effectively, achieving a well-balanced exploration across the entire search tree. For clarity, we present a simplified tree for illustration.

(e.g., preprocessing, feature engineering, model training), the value $v(x)$ representing the cumulative score from simulations for this node and its descendants, the number of visits $n_{\text{visits}}(x)$ showing the total number of simulations conducted, the simulation score $s(x)$ for the node's simulation, the solution code $\sigma_{\text{sol}}(x)$ produced after simulation, and the stage code $\sigma_{\text{stage}}(x)$ generated up to this node's stage. A path from the root to node x represents an experiment configuration $c(x) = \{\lambda(x_1), \lambda(x_2), \ldots, \lambda(x)\}$.

3.3.2 Tree Search for ML Experiments

Monte Carlo Tree Search (MCTS) is used to explore and identify optimal machine learning solutions. The search process involves selection, expansion, simulation, and backpropagation.

Selection At each iteration, we use a modified version of the UCT (Upper Confidence Bound for Trees) algorithm (Kocsis and Szepesvári, 2006), referred to as UCT-DP (depth-preferred), to select a node from the search tree. Unlike traditional MCTS, where simulations are often performed quickly due to a fixed action space and negligible action time, the context of machine learning tasks presents a different challenge. Processes such as model training introduce significant computational time, making efficient node exploration crucial. Since model selection can heavily influence the overall machine learning performance, we prioritize exploring nodes at greater depths early on.

This modification reduces the need to explore every unvisited node, allowing deeper nodes to be reached in fewer iterations—making the approach better suited for large-scale machine learning experiments. The modified selection algorithm is expressed as:

$$\text{UCT-DP}(x) = \frac{v(x)}{n(x)} + \alpha_{\text{explore}} \sqrt{\frac{\ln n_{\text{visits}}(x_{\text{parent}})}{n(x)}} \tag{5}$$

$$n(x) = \begin{cases} \alpha_{\text{unvisted}} & \text{if } n_{\text{visits}}(x) = 0 \\ n_{\text{visits}}(x) & \text{otherwise.} \end{cases} \tag{6}$$

Here, α_{unvisted} is a constant between 0 and 1 controlling the selection preference for unvisited nodes, balancing between full exploration and computational efficiency. This adjustment allows us to focus more on deeper parts of the tree that are likely to yield better solutions. We illustrate the mechanism of UCT-DP in comparison to standard UCT in Figure 4.

Expansion During the expansion phase, a set of child nodes X_{child} are instantiated from the selected node x for potential simulation. Note that a child node x_{child} from the node x at depth δ inherits the attributes of x and possesses $\lambda(x_{\text{child}}) \rightarrow \lambda^{\tau_{\delta+1}}$, an insight of stage $\tau_{\delta+1}$ from the search space.

Simulation Once expanded, a node x_{sample} is uniformly sampled from X_{child} for simulation. The path from root to the sampled node forms a set of insights $c(x_{\text{sample}}) = \{\lambda(x_1), \lambda(x_2), \ldots, \lambda(x_{\text{sample}})\} \subset \Lambda$, representing

the experiment configuration. This configuration is passed to the experimenter E for execution, following E_{plan} and $E_{\text{code \& execute}}$, yielding a simulation score s, as described in Section 3.3.1. The score serves as feedback for backpropagation. The simulation process is outlined in Algorithm 2.

Backpropagation After the simulation concludes, the performance score (e.g., based on the development set) is retrieved and backpropagated through the tree. The score is propagated from the simulated node up to the root, updating each parent node's value and visit count. This allows nodes representing more promising solutions to be prioritized in future rollouts. In addition, the solution code is also backpropagated up to the tree, and it can be processed and saved as stage code depending on the parent node during the update.

Backpropagation ensures that the algorithm learns which paths yield better results, guiding the search toward higher-performing nodes as more rollouts are conducted.

3.3.3 Experiment State Saving and Loading

To improve efficiency and reduce token usage, SELA caches code at the stage level for each configuration. This allows for reusing code when similar configurations are encountered, and ensures consistency by rerunning saved stage code, addressing LLM non-determinism. This approach minimizes resource consumption (Appendix H) while maintaining robust performance.

4 Experiments

4.1 Experimental Setup

Datasets We evaluate SELA alongside several baselines on 20 datasets, which include 13 classification tasks and 7 regression tasks from the AutoML Benchmark (AMLB) (Gijsbers et al., 2024) and Kaggle Competitions.

Table 5 provides detailed information on the datasets used. All datasets are split into training, validation, and test sets with a 6:2:2 ratio. Each framework utilizes the training and validation sets to train models and makes predictions on the test set labels.

Evaluation Metrics For the AMLB datasets, we use the default target column provided by OpenML. For Kaggle competition datasets, we rely on the target column specified in the competition description. Performance is measured using root mean squared error (RMSE) for regression tasks, F1 score for binary classification, and F1-weighted score for multi-class classification. To ensure comparability across datasets with varying metrics, we introduce a Normalized Score (NS), which maps RMSE into the range from 0 to 1.

$$\text{NS}(s_{\text{raw}}) = \begin{cases} \frac{1}{1+\log(1+s_{\text{raw}})} & \text{if the metric is RMSE.} \\ s_{\text{raw}} & \text{otherwise.} \end{cases} \tag{7}$$

Here, s_{raw} represents the raw score before normalization. To evaluate SELA against other frameworks, we employ three key metrics: average Normalized Score (NS), average rank, and average best rank. The average rank is calculated by considering all rankings of a method across datasets, while the average best rank focuses on the method's best performance in each dataset. We also want to quantify how other baselines perform relative to SELA. The "Rescaled NS" is defined as:

$$\text{Rescaled NS}(f) = \frac{\text{NS}_f}{\text{NS}_{\text{SELA}}} \tag{8}$$

where f represents the baseline method being compared to SELA.

Method and Baselines Setup We compare SELA with several baseline methods, including Data Interpreter (Hong et al., 2024a), AIDE (Jiang et al., 2025), AutoGluon (Erickson et al., 2020), and AutoSklearn (Feurer et al., 2015, 2020).

For our LLM-based approaches (SELA, Data Interpreter, and AIDE), we employ a consistent initial task prompt across all methods. This prompt encompasses the dataset name, target column, and evaluation metric. We choose `DeepSeek-V2.5` (DeepSeek-AI, 2024) as our foundation LLM due to its open-source nature, strong coding capabilities, and cost-effective token usage. To encourage output diversity, we set the temperature parameter to 0.5 for all LLM-based methods. AIDE conducts 10 iterations per execution, while SELA performs 10 rollouts.

For SELA, we employ Data Interpreter as the experimenter, leveraging its multi-step generation capability. We configured the hyperparameters of UCT-DP as follows: $\alpha_{\text{unvisited}}$ is set to 0.8 and α_{explore} is set to 1.4. These settings aim to balance exploration and exploitation in the method's search strategy. Each method, except for AutoGluon, is run three times for each dataset. AutoGluon, being deterministic, is run only once with its default settings. AutoSklearn is also run with default settings.

Method	Wins	Losses	Top 1	Avg. NS % ↑	Avg. Best NS % ↑	Avg. Rank ↓	Avg. Best Rank ↓
AutoGluon	7	13	4	53.2	53.2	**4.4**	4.4
AutoSklearn	5	15	5	46.1	47.5	7.6	6.1
AIDE	5	15	2	47.1	51.8	7.8	5.3
Data Interpreter	4	16	2	47.4	50.2	8.8	6.4
SELA	-	-	**7**	**53.3**	**54.7**	4.8	**2.7**

Table 2: Results of each AutoML framework on 20 tabular datasets. The “Wins" column indicates the number of datasets where the method outperforms SELA, while “Losses" shows the number of datasets where the method underperforms. The “Top 1" column represents the number of datasets where the method produces the best predictions across methods.

4.2 Results

As shown in Table 2, SELA achieves the highest average Normalized Score (NS) and average best rank among all frameworks. Notably, SELA excels in producing the highest number of top predictions, as indicated in the “Top 1" column across all datasets. Furthermore, the “Losses" column reveals that each competing method falls short against SELA, losing in 65-80% of the datasets.

Interestingly, AutoGluon exhibits a marginally higher average rank than SELA. This slight discrepancy may be attributed to the inherent randomness in LLMs and model training processes, which can influence the exploration of machine learning solutions. However, SELA’s higher average NS suggests that it performs strongly in the datasets where it excels, while its losses in other datasets are relatively minor. This means that even when SELA produces lower-ranked solutions, the performance gap is small, allowing it to fully compensate in the datasets where it performs well.

The two other agent-based methods exhibit relatively lower performance. The first method, Data Interpreter, struggles to enhance its score with multiple attempts due to its inability to refine its solution after completing a machine learning task. The second method, AIDE, does not have a stage-specific planning module, limiting its capacity to improve results after a series of greedy exploitation, which makes it prone to falling into local optima. These limitations likely account for their weaker performance.

Figure 5 further corroborates SELA’s effectiveness, revealing that its best solutions frequently occupy leading positions across various datasets. This visual representation exhibits the method’s consistent high performance and adaptability across different ML datasets. We also include a detailed results of each method in Appendix D.

Figure 5: Rescaled NS of AutoML frameworks relative to SELA on tabular datasets. Points to the left of the vertical line indicate poorer predictions compared to SELA. Notably, SELA mostly occupies a leading position across the datasets.

4.3 Ablation Study

For the rest of the study, we employ a subset of datasets to evaluate SELA under various settings. Our selection process involves choosing the first two datasets alphabetically for each machine learning task. Specifically, we use boston, colleges, credit-g, Click_prediction_small, GesturePhaseSegmentationProcessed, and mfeat-factors to conduct the ablation study.

	Avg. NS ↑	Avg. Rank ↓
Data Interpreter	56.4	9.2
SELA-RS	58.6	5.6
SELA-MCTS (UCT)	58.9	5.8
SELA-MCTS (UCT-DP)	**60.9**	**4.1**

Table 3: Performance across different search strategies on six datasets. SELA-MCTS with UCT-DP (depth-preferred) outperforms standard UCT and random sampling (RS), achieving the highest normalized score and best average rank. This highlights the effectiveness of guiding search deeper under limited rollouts.

Effectiveness of Search To evaluate the impact of search strategies, we compared MCTS with UCT and UCT-DP, random sampling of the insights from the insight pool, and the Data Interpreter (Table 3). Between the two MCTS variants, UCT-DP achieves the highest normalized score (60.9) and best average rank (4.1), clearly outperforming standard UCT (58.9, 5.8). This confirms that depth-

preferred search improves solution quality under limited rollouts by encouraging earlier exploration of deeper nodes. In contrast, standard UCT tends to over-explore shallow nodes, leading to less effective optimization. Moreover, even the random sampling variant of our method outperforms Data Interpreter, the base experimenter. This suggests that an appropriate search space and an experiment agenda is vital for improving a machine learning agent. Our insight proposer generates relevant and useful insights, facilitating such improvement, regardless of the selection method.

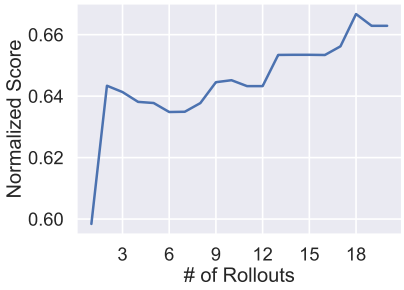

Figure 6: The average performance of SELA on six selected datasets with an increasing number of rollouts.

Number of Rollouts Figure 6 illustrates that the average performance of SELA improves as the number of permitted rollouts increases. The trend demonstrates the strong scalability of SELA, as it efficiently leverages additional opportunities to explore the search space, improving the normalized score by 4.7% after 10 rollouts and 6.4% after 20, compared to the initial rollout.

	DS V2.5	G4o	C3.5	Qwen2.5	DS-R1
Click_pred	23.2	27.6	15.6	18.0	**35.3**
Gesture	**67.9**	65.6	63.4	66.1	66.6
boston	40.1	40.9	**41.6**	40.5	41.4
colleges	87.8	88.0	87.3	87.5	**88.0**
credit-g	50.9	55.2	43.2	49.3	**57.7**
mfeat	95.7	**96.4**	96.1	94.6	95.8
Avg. NS ↑	60.9	62.3	57.9	59.3	**64.1**

Table 4: Performance of SELA across LLMs. DS V2.5, G4o, C3.5, Qwen2.5, and DS-R1 stand for DeepSeek V2.5, GPT-4o, Claude-3.5-Sonnet, Qwen2.5-72B-Instruct, and DeepSeek-R1 respectively. DeepSeek-R1 achieves the highest average normalized score (NS).

Robustness Across LLMs To evaluate the robustness and adaptability of our framework, we conduct experiments using a variety of LLMs. Specifically, we assess the performance of SELA with `GPT-4o` (OpenAI, 2024), `Claude-3.5-Sonnet` (Anthropic, 2024), `DeepSeek V2.5`, `Qwen2.5-72B-Instruct` (Yang et al., 2024), and `DeepSeek-R1` (Guo et al., 2025).

As illustrated in Table 4, SELA achieves strong and consistent performance across these different LLMs. Notably, `DeepSeek-R1` achieves the highest average normalized score (NS) of 64.1, while the smaller open-source model `Qwen2.5-72B-Instruct` also obtains a competitive score of 59.3. These results demonstrate that SELA is not only effective but also highly adaptable to a wide range of LLMs, including both proprietary and open-source models.

5 Conclusion

We introduced SELA, a novel framework that combines LLM-based agents with Monte Carlo Tree Search (MCTS) to automate machine learning workflows. Experimental results across 20 datasets demonstrate its effectiveness and advantages over traditional AutoML frameworks and existing LLM-based approaches.

Future work could extend SELA to other domains, such as software engineering, scientific discovery, game playing, and robotics as these sequential decision-making problems can potentially be represented as tree structures with scalar rewards. Improving search efficiency and scalability for larger solution spaces is another key direction, along with developing techniques for providing interpretable explanations to enhance transparency. SELA showcases the potential of combining traditional search algorithms with LLM flexibility in automated machine learning.

Limitations While focused on tabular datasets, SELA can be adapted to other data types, including time series, text, and images. Additionally, while SELA enhances machine learning workflows, it lacks inherent safeguards against harmful content generation, so we recommend implementing safety measures for responsible use.

References

Anthropic. 2024. Introducing Claude 3.5 Sonnet — anthropic.com. `https://www.anthropic.com/news/claude-3-5-sonnet`.

Graeme Best, Oliver M Cliff, Timothy Patten, Ramgopal R Mettu, and Robert Fitch. 2019. Dec-mcts: Decentralized planning for multi-robot active perception. *The International Journal of Robotics Research*, 38(2-3):316–337.

Tobias Block. 2019. 10kgnad: Ten thousand german news articles dataset for topic classification. https://tblock.github.io/10kGNAD/. Accessed: 2025-05-17.

Jun Shern Chan, Neil Chowdhury, Oliver Jaffe, James Aung, Dane Sherburn, Evan Mays, Giulio Starace, Kevin Liu, Leon Maksin, Tejal Patwardhan, Lilian Weng, and Aleksander Mądry. 2025. Mle-bench: Evaluating machine learning agents on machine learning engineering. *Preprint*, arXiv:2410.07095.

Yizhou Chi, Kevin Yang, and Dan Klein. 2024. Thoughtsculpt: Reasoning with intermediate revision and search.

Patrick Clary, Pedro Morais, Alan Fern, and Jonathan Hurst. 2018. Monte-carlo planning for agile legged locomotion. *Proceedings of the International Conference on Automated Planning and Scheduling*, 28(1):446–450.

DeepSeek-AI. 2024. Deepseek-v2: A strong, economical, and efficient mixture-of-experts language model. *Preprint*, arXiv:2405.04434.

Vikranth Dwaracherla, Seyed Mohammad Asghari, Botao Hao, and Benjamin Van Roy. 2024. Efficient exploration for llms.

Nick Erickson, Jonas Mueller, Alexander Shirkov, Hang Zhang, Pedro Larroy, Mu Li, and Alexander Smola. 2020. Autogluon-tabular: Robust and accurate automl for structured data. *Preprint*, arXiv:2003.06505.

Xidong Feng, Ziyu Wan, Muning Wen, Ying Wen, Weinan Zhang, and Jun Wang. 2023. Alphazero-like tree-search can guide large language model decoding and training.

Matthias Feurer, Katharina Eggensperger, Stefan Falkner, Marius Lindauer, and Frank Hutter. 2020. Auto-sklearn 2.0: Hands-free automl via meta-learning.

Matthias Feurer, Aaron Klein, Katharina Eggensperger, Jost Springenberg, Manuel Blum, and Frank Hutter. 2015. Efficient and robust automated machine learning. In *Advances in Neural Information Processing Systems 28 (2015)*, pages 2962–2970.

Pieter Gijsbers, Marcos L. P. Bueno, Stefan Coors, Erin LeDell, Sébastien Poirier, Janek Thomas, Bernd Bischl, and Joaquin Vanschoren. 2024. Amlb: an automl benchmark. *Journal of Machine Learning Research*, 25(101):1–65.

Daya Guo, Dejian Yang, Haowei Zhang, Junxiao Song, Ruoyu Zhang, Runxin Xu, Qihao Zhu, Shirong Ma, Peiyi Wang, Xiao Bi, Xiaokang Zhang, Xingkai Yu, Yu Wu, Z. F. Wu, Zhibin Gou, Zhihong Shao, Zhuoshu Li, Ziyi Gao, Aixin Liu, and 180 others. 2025. Deepseek-r1: Incentivizing reasoning capability in llms via reinforcement learning. *Preprint*, arXiv:2501.12948.

Siyuan Guo, Cheng Deng, Ying Wen, Hechang Chen, Yi Chang, and Jun Wang. 2024. Ds-agent: Automated data science by empowering large language models with case-based reasoning.

Noah Hollmann, Samuel Müller, and Frank Hutter. 2024. Large language models for automated data science: Introducing caafe for context-aware automated feature engineering.

Sirui Hong, Yizhang Lin, Bang Liu, Bangbang Liu, Binhao Wu, Danyang Li, Jiaqi Chen, Jiayi Zhang, Jinlin Wang, Li Zhang, Lingyao Zhang, Min Yang, Mingchen Zhuge, Taicheng Guo, Tuo Zhou, Wei Tao, Wenyi Wang, Xiangru Tang, Xiangtao Lu, and 6 others. 2024a. Data interpreter: An llm agent for data science. *Preprint*, arXiv:2402.18679.

Sirui Hong, Mingchen Zhuge, Jonathan Chen, Xiawu Zheng, Yuheng Cheng, Jinlin Wang, Ceyao Zhang, Zili Wang, Steven Ka Shing Yau, Zijuan Lin, Liyang Zhou, Chenyu Ran, Lingfeng Xiao, Chenglin Wu, and Jürgen Schmidhuber. 2024b. MetaGPT: Meta programming for a multi-agent collaborative framework. In *The Twelfth International Conference on Learning Representations*.

Qian Huang, Jian Vora, Percy Liang, and Jure Leskovec. 2023. Mlagentbench: Evaluating language agents on machine learning experimentation. *arXiv preprint arXiv:2310.03302*.

Wenyang Hui and Kewei Tu. 2024. Rot: Enhancing large language models with reflection on search trees.

Zhengyao Jiang, Dominik Schmidt, Dhruv Srikanth, Dixing Xu, Ian Kaplan, Deniss Jacenko, and Yuxiang Wu. 2025. Aide: Ai-driven exploration in the space of code. *Preprint*, arXiv:2502.13138.

Haifeng Jin, François Chollet, Qingquan Song, and Xia Hu. 2023. Autokeras: An automl library for deep learning. *Journal of machine Learning research*, 24(6):1–6.

Haifeng Jin, Qingquan Song, and Xia Hu. 2019. Auto-keras: An efficient neural architecture search system. In *Proceedings of the 25th ACM SIGKDD international conference on knowledge discovery & data mining*, pages 1946–1956.

Levente Kocsis and Csaba Szepesvári. 2006. Bandit based monte-carlo planning. In *Machine Learning: ECML 2006*, pages 282–293, Berlin, Heidelberg. Springer Berlin Heidelberg.

Jonathan Krause, Michael Stark, Jia Deng, and Li Fei-Fei. 2013. 3d object representations for fine-grained categorization. In *Proceedings of the IEEE International Conference on Computer Vision Workshops*, pages 554–561.

Akshay Krishnamurthy, Keegan Harris, Dylan J. Foster, Cyril Zhang, and Aleksandrs Slivkins. 2024. Can large language models explore in-context?

Erin LeDell and Sebastien Poirier. 2020. H2O AutoML: Scalable automatic machine learning. *7th ICML Workshop on Automated Machine Learning (AutoML)*.

Dawei Li, Zhen Tan, and Huan Liu. 2024. Exploring large language models for feature selection: A data-centric perspective. *Preprint*, arXiv:2408.12025.

Siyi Liu, Chen Gao, and Yong Li. 2024. Large language model agent for hyper-parameter optimization. *arXiv preprint arXiv:2402.01881*.

Daqin Luo, Chengjian Feng, Yuxuan Nong, and Yiqing Shen. 2024. Autom3l: An automated multimodal machine learning framework with large language models. *arXiv preprint arXiv:2408.00665*.

Randal S Olson and Jason H Moore. 2016. Tpot: A tree-based pipeline optimization tool for automating machine learning. In *Workshop on automatic machine learning*, pages 66–74. PMLR.

OpenAI. 2024. Hello GPT-4o. `https://openai.com/index/hello-gpt-4o/`.

Omkar M Parkhi, Andrea Vedaldi, Andrew Zisserman, and C V Jawahar. 2012. Cats and dogs. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 3498–3505.

Samuel Schmidgall, Yusheng Su, Ze Wang, Ximeng Sun, Jialian Wu, Xiaodong Yu, Jiang Liu, Zicheng Liu, and Emad Barsoum. 2025. Agent laboratory: Using llm agents as research assistants. *arXiv preprint arXiv:2501.04227*.

Marwin Segler, Mike Preuss, and Mark Waller. 2018. Planning chemical syntheses with deep neural networks and symbolic ai. *Nature*, 555:604–610.

David Silver, Aja Huang, Chris J. Maddison, Arthur Guez, L. Sifre, George van den Driessche, Julian Schrittwieser, Ioannis Antonoglou, Vedavyas Panneershelvam, Marc Lanctot, Sander Dieleman, Dominik Grewe, John Nham, Nal Kalchbrenner, Ilya Sutskever, Timothy P. Lillicrap, Madeleine Leach, Koray Kavukcuoglu, Thore Graepel, and Demis Hassabis. 2016. Mastering the game of go with deep neural networks and tree search. *Nature*.

David Silver, Julian Schrittwieser, Karen Simonyan, Ioannis Antonoglou, Aja Huang, Arthur Guez, Thomas Hubert, Lucas baker, Matthew Lai, Adrian Bolton, Yutian Chen, Timothy P. Lillicrap, Fan Hui, L. Sifre, George van den Driessche, Thore Graepel, and Demis Hassabis. 2017. Mastering the game of go without human knowledge. *Nature*.

Hao Tang, Keya Hu, Jin Peng Zhou, Sicheng Zhong, Wei-Long Zheng, Xujie Si, and Kevin Ellis. 2024a. Code repair with llms gives an exploration-exploitation tradeoff.

Zhiqiang Tang, Haoyang Fang, Su Zhou, Taojiannan Yang, Zihan Zhong, Tony Hu, Katrin Kirchhoff, and George Karypis. 2024b. Autogluon-multimodal (automm): Supercharging multimodal automl with foundation models. *arXiv preprint arXiv:2404.16233*.

Chris Thornton, Frank Hutter, Holger H Hoos, and Kevin Leyton-Brown. 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*, pages 847–855.

Ante Wang, Linfeng Song, Ye Tian, Baolin Peng, Dian Yu, Haitao Mi, Jinsong Su, and Dong Yu. 2024a. Litesearch: Efficacious tree search for llm. *Preprint*, arXiv:2407.00320.

Chi Wang, Qingyun Wu, Markus Weimer, and Erkang Zhu. 2021. Flaml: A fast and lightweight automl library. In *MLSys*.

Xingyao Wang, Boxuan Li, Yufan Song, Frank F. Xu, Xiangru Tang, Mingchen Zhuge, Jiayi Pan, Yueqi Song, Bowen Li, Jaskirat Singh, and 1 others. 2024b. Openhands: An open platform for ai software developers as generalist agents. *arXiv preprint arXiv:2407.16741*.

Feng Wu, Sarvapali D. Ramchurn, Wenchao Jiang, Jeol E. Fischer, Tom Rodden, and Nicholas R. Jennings. 2015. Agile planning for real-world disaster response. In *Proceedings of the 24th International Conference on Artificial Intelligence*, IJCAI'15, page 132–138. AAAI Press.

Han Xiao, Kashif Rasul, and Roland Vollgraf. 2017. Fashion-mnist: a novel image dataset for benchmarking machine learning algorithms. *CoRR*, abs/1708.07747.

An Yang, Baosong Yang, Beichen Zhang, Binyuan Hui, Bo Zheng, Bowen Yu, Chengyuan Li, Dayiheng Liu, Fei Huang, Haoran Wei, Huan Lin, Jian Yang, Jianhong Tu, Jianwei Zhang, Jianxin Yang, Jiaxi Yang, Jingren Zhou, Junyang Lin, Kai Dang, and 23 others. 2024. Qwen2.5 technical report. *Preprint*, arXiv:2412.15115.

Lei Zhang, Yuge Zhang, Kan Ren, Dongsheng Li, and Yuqing Yang. 2024. Mlcopilot: Unleashing the power of large language models in solving machine learning tasks. *Preprint*, arXiv:2304.14979.

Andy Zhou, Kai Yan, Michal Shlapentokh-Rothman, Haohan Wang, and Yu-Xiong Wang. 2024. Language agent tree search unifies reasoning acting and planning in language models.

A Datasets

Table 5 outlines the detailed information of the datasets used for evaluation.

Dataset name	# Features	# Rows	# Classes	Task Type	Metric	Source
boston	14	506	N/A	Regression	RMSE	OpenML (Dataset ID: 531)
colleges	48	7063	N/A	Regression	RMSE	OpenML (Dataset ID: 42727)
concrete-strength	9	4866	N/A	Regression	RMSE	Kaggle (playground-series-s3e9)
diamonds	10	53940	N/A	Regression	RMSE	OpenML (Dataset ID: 42225)
house-prices	81	1460	N/A	Regression	RMSE	Kaggle (house-prices-advanced-regression-techniques)
Moneyball	15	1232	N/A	Regression	RMSE	OpenML (Dataset ID: 41021)
SAT11-HAND-runtime-regression	118	4440	N/A	Regression	RMSE	OpenML (Dataset ID: 41980)
credit-g	21	1000	2	Classification	F1	OpenML (Dataset ID: 31)
Click_prediction_small	12	39948	2	Classification	F1	OpenML (Dataset ID: 42733)
icr	58	617	2	Classification	F1	Kaggle (icr-identify-age-related-conditions)
jasmine	145	2984	2	Classification	F1	OpenML (Dataset ID: 41143)
kc1	21	2109	2	Classification	F1	OpenML (Dataset ID: 1067)
kick	33	72983	2	Classification	F1	OpenML (Dataset ID: 41162)
smoker-status	23	143330	2	Classification	F1	Kaggle (playground-series-s3e24)
software-defects	22	91586	2	Classification	F1	Kaggle (playground-series-s3e23)
titanic	12	891	2	Classification	F1	Kaggle (titanic)
GesturePhaseSegmentationProcessed	33	9873	5	Multiclass	F1-weighted	OpenML (Dataset ID: 4538)
mfeat-factors	217	2000	10	Multiclass	F1-weighted	OpenML (Dataset ID: 12)
segment	20	2310	7	Multiclass	F1-weighted	OpenML (Dataset ID: 40984)
wine-quality-white	12	4898	7	Multiclass	F1-weighted	OpenML (Dataset ID: 40498)

Table 5: Summary of the machine learning datasets used in the experiments. OpenML datasets can be accessed using their respective dataset IDs. The Kaggle datasets are available at https://www.kaggle.com/competitions/{source}.

B Pseudo Code of SELA

Algorithm 1 Tree-Search Enhanced LLM Agents

Input: Problem description p, data information d, data D, LLM M, rollout number k.

$\Lambda \leftarrow \text{InsightProposer}(p, d, M)$
Initialize Tree using Λ
for $i = 1$ **to** k **do**
  node $x \leftarrow$ select(Tree)
  $X_{\text{child}} \leftarrow$ expand(Tree, x)
  Randomly sample a node x_{sample} from X_{child}
  Retreive experiment configuration $c(x_{\text{sample}})$
  $\sigma_{sol}, s \leftarrow \text{simulate}(c(x_{\text{sample}}), p, d, D, M)$
  attach the simulation result σ_{sol}, s to x_{sample} for final solution selection
  Backpropagate(Tree, s)
end for
$x_{\text{dev best}} \leftarrow \underset{x \in \text{Tree}}{\text{argmax}}(s(x))$

Output: $\sigma_{sol}(x_{\text{dev best}})$

Algorithm 2 Simulate

Input: Experiment configuration c, problem description p, data information d, data D, LLM M.

Draft plans $I^{\tau \in T} \leftarrow E_{\text{plan}}(p, d, c, M)$
Code and execute sequentially $\sigma^{\tau \in T}, s \leftarrow E_{\text{code \& execute}}(I^{\tau \in T}, D, M)$
$\sigma_{sol} \leftarrow \text{concatenate}(\sigma^{\tau \in T})$

Output: σ_{sol}, s

C Prompts

C.1 Task Prompt

All LLM-based methods start by receiving the same base requirement prompt at the beginning of the task. The prompt specifies the dataset's name, the target label column, the evaluation metric to be used, and the dataset's file path. Furthermore, the prompt include a path to a text file containing the dataset's metadata.

```
TASK_PROMPT = """
# User requirement
This is a {datasetname} dataset.
Your goal is to predict the target column `{target_col}`.
Perform data analysis, data preprocessing, feature engineering, and modeling to predict the target. Report {metric} on the eval data. Do not plot or make any visualizations.

# Data dir
train set (with labels): {train_path}
dev set (with labels): {dev_path}
test set (without labels): {test_path}
dataset description: {data_info_path}
(During EDA, you can use this file
to get additional information about the dataset)
"""
```

Since AIDE automatically splits the training data into a new train set and a validation set, we combine the original train and validation sets and provide them as input to AIDE. We set k_fold_validation to 1 in its configuration file to enforce a single train-val split for closer alignment with our setup. In both setups, the frameworks have access to the labels for both the train and validation sets.

C.2 Instruction Prompt

The instruction prompt would direct the framework to save the final prediction file for evaluation.

```
DI_INSTRUCTION = """
## Attention
1. Please do not leak the target label in any form during training.
2. Test set does not have the target column.
3. When conducting data exploration or analysis, print out the results of your findings.
4. You should perform transformations on train, dev, and test sets at the same time (it's a good idea to define functions for this and avoid code repetition).
5. When scaling or transforming features, make sure the target column is not included.
6. You could utilize dev set to validate and improve model training. {special_instruction}

## Saving Dev and Test Predictions
1. Save the prediction results of BOTH the dev set and test set in `dev_predictions.csv` and `test_predictions.csv` respectively in the output directory.
- Both files should contain a single column named `target` with the predicted values.
2. Make sure the prediction results are in the same format as the target column in the training set.
- For instance, if the target column is categorical, the prediction results should be categorical as well.

## Output Performance
Print the train and dev set performance in the last step.

# Output dir
{output_dir}
"""
```

C.3 Insight Proposal Prompt

Insight Proposer uses this prompt to generate a search space of insights for different stages of the machine learning task.

```
DATASET_INSIGHT_PROMPT = """
# Dataset Description
{dataset}

# Dataset Metadata
{metadata}

# Dataset Head
{head}

# Instruction
Propose insights to help improve the performance of the model on this dataset.
The insights should be proposed based on the dataset description with different task types.
Each task type should have at least 5 insights.
Make sure each method is diverse enough and can be implemented separately.
Be specific about models' choices, ensemble and tuning techniques, and preprocessing & feature engineering techniques.

# Format
```json
[
    {{
        "task_type": "EDA",
        "insights": [
            "insight1",
            "insight2",
            "insight3",
            ...
            "insightN"
        ]
    }},
    {{
        "task_type": "Data Preprocessing",
        "insights": [
            "insight1",
            "insight2",
            "insight3",
            ...
            "insightN"
        ]
    }},
    {{
        "task_type": "Feature Engineering",
        "insights": [
            "insight1",
            "insight2",
            "insight3",
            ...
            "insightN"
        ]
    }},
    {{
        "task_type": "Model Training",
        "insights": [
            "insight1",
            "insight2",
            "insight3",
            ...
            "insightN"
        ]
    }}
]
```
"""
```

D Results

D.1 Main Results

Dataset	AutoGluon Avg.	AutoGluon Best	AutoSklearn Avg.	AutoSklearn Best	AIDE Avg.	AIDE Best	DI Avg.	DI Best	SELA Avg.	SELA Best
Click_prediction_small	7	7	2	1	7.3	4	11	10	7.7	6
GesturePhaseSegmentationProcessed	1	1	6.3	3	7.3	4	11	10	5.3	2
Moneyball	4	4	10	9	4	1	9	2	6	3
SAT11-HAND-runtime-regression	1	1	12	11	5.3	3	9	8	3.7	2
boston	5	5	12	11	3.7	2	9	8	4	1
colleges	1	1	12	11	6	2	8	7	4	3
concrete-strength	5	5	12	11	6.3	4	2	1	8.3	6
credit-g	4	4	10	9	10	5	5.3	1	3.7	2
diamonds	2	2	12	11	6	4	8.7	7	3	1
house-prices	1	1	12	11	6.7	5	7.3	3	4	2
icr	5	5	5.3	3	12	11	9	8	2.3	1
jasmine	7	7	6	4	8.7	5	11.3	9	2	1
kc1	10	10	2.7	1	8	5	11.3	9	5	2
kick	4	4	2	1	9.3	6	11	10	6.7	5
mfeat-factors	4	4	2	1	10	9	10.3	6	6.7	5
segment	3	3	6.3	5	11	10	9.7	7	2.3	1
smoker-status	7	7	4.7	3	11.3	9	7.7	2	4.3	1
software-defects	8	8	2	1	12	11	6	4	7.7	6
titanic	7	7	9.7	6	2.7	1	10.3	8	5.3	3
wine-quality-white	2	2	10	8	7.3	4	9	7	3.3	1
Overall Rank ↓	**4.4**	4.4	7.6	6.1	7.8	5.3	8.8	6.4	4.8	**2.7**

Table 6: Methods' ranking for each tabular dataset

E Tree Search vs. Reasoning-Based Models

With the emergence of advanced reasoning models like `DeepSeek-R1`, a pivotal question arises: how do deliberative capabilities compare to empirical, feedback-driven methods in complex decision-making tasks? To explore this, we evaluate three configurations that dissect the contributions of reasoning capabilities and structured search:

1. **Tree Search + Non-Reasoning Model**: SELA with `DeepSeek-V2.5`
2. **Reasoning Model**: standalone `DeepSeek-R1`
3. **Tree Search + Reasoning Model**: SELA with `DeepSeek-R1`

SELA with `DeepSeek-V2.5` outperforms the standalone reasoning-heavy `DeepSeek-R1` on 5 out of 6 datasets, demonstrating that empirical, feedback-driven experimentation can outperform purely deliberative reasoning in complex data analysis workflows. Notably, the best results are obtained when both approaches are combined.

	boston	**Click_pred**	**colleges**	**credit-g**	**Gesture**	**mfeat**	**Avg.**
SELA (V2.5)	40.1	23.2	87.8	50.9	**67.9**	95.7	60.9
DeepSeek-R1	39.9	29.8	87.6	48.5	64.7	93.9	60.7
SELA (R1)	**41.4**	**35.3**	**88.0**	**57.7**	66.6	**95.8**	**64.1**

Table 8: Performance comparison between tree search and reasoning-based modeling configurations across six datasets.

The results highlight a key distinction between reasoning-driven approaches like `DeepSeek-R1` and SELA's tree search framework. Long-form Chain of Thoughts (CoT) enables a model to carefully deliberate before coding, selecting solutions it deems optimal based on internal reasoning. While this can be beneficial, it remains a static decision that may not always align with real-world outcomes. In contrast, SELA emphasizes iterative experimentation—actively testing and refining solutions based on actual data feedback. Much like a human expert, while deep reasoning is valuable, it is through structured experimentation that one can verify assumptions and adapt strategies dynamically, leading to superior performance.

Notably, SELA and reasoning models like R1 are complementary rather than competing approaches. SELA's framework is model-agnostic and can seamlessly integrate reasoning-enhanced LLMs to further improve performance. We tested SELA with `DeepSeek-R1` and observed a significant performance boost, achieving an average score of 64.1. This demonstrates that incorporating stronger reasoning capabilities within SELA's experimental framework further enhances its effectiveness, pushing the performance boundary even higher.

F SELA's Adaptability to Additional Datasets and Benchmarks

To evaluate SELA's generalization capabilities, we conducted experiments on ten additional datasets encompassing various modalities, including text, image, and tabular data. These datasets were sourced from classical benchmarks, trending Huggingface datasets, and MLE-Bench (Chan et al., 2025) tasks.

The following table summarizes the normalized scores achieved by SELA and AIDE across the selected datasets:

Dataset	Modality (Metric)	AIDE	SELA
`sms_spam` (Huggingface)	Text (F1)	93.3	**97.5**
`banking77` (Huggingface)	Text (F1-weighted)	86.0	**88.4**
`gnad10` (Block, 2019)	Text (F1-weighted)	84.8	**85.6**
`random-acts-of-pizza` (MLE-Bench)	Text (AUC)	64.5	**65.9**
`fashion_mnist` (Xiao et al., 2017)	Image (F1-weighted)	0	**90.1**
`oxford-iiit-pet` (Parkhi et al., 2012)	Image (F1-weighted)	0	**88.9**
`stanford_cars` (Krause et al., 2013)	Image (F1-weighted)	0	**73.7**
`plant-pathology-2020-fgvc7` (MLE-Bench)	Image (AUC)	**98.1**	70.0
`tabular-playground-series-dec-2021` (MLE-Bench)	Tabular (Accuracy)	**95.8**	95.3
`nomad2018-predict-transparent-conductors` (MLE-Bench)	Tabular (RMSLE)	88.07	**94.2**

Table 9: Normalized scores of AIDE and SELA across diverse datasets.

The results indicate that SELA exhibits robust performance across various data modalities. Notably, SELA successfully generated valid solutions for image-based datasets such as `fashion_mnist`, `oxford-iiit-pet`, and `stanford_cars`, where AIDE failed to produce valid outputs. This suggests that SELA is highly adaptable and capable of generalizing beyond tabular tasks.

G Comparison with General-Purpose Agentic Frameworks

We additionally compare SELA with general software engineering agents OpenHands (Wang et al., 2024b), MLAB (Huang et al., 2023), and Agent Laboratory (Schmidgall et al., 2025) across the ablation datasets. Each method is executed three times per dataset, and we report the average Normalized Score. If a method fails to produce a valid test set prediction, we exclude it from the average and note the success rate in parentheses.

Method	boston	Click_pred	colleges	credit-g	Gesture	mfeat
SELA	**40.1**	23.2	**87.8**	**50.9**	**67.9**	**95.7**
OpenHands	38.5 (66.7%)	22.3	87.6 (66.7%)	48.8 (33.3%)	64.4	94.7
MLAB	N/A (0%)	17.0 (33.3%)	0.4 (33.3%)	N/A (0%)	N/A (0%)	0.1 (33.3%)
Agent Laboratory (GPT-4o)	38.6	**27.2** (33.3%)	86.9 (66.7%)	32.9 (33.3%)	54.2	93.9 (66.7%)

Table 10: Normalized Scores across six datasets. Success rates are indicated in parentheses where applicable.

SELA achieves the highest average score and outperforms OpenHands on 5 of 6 datasets. OpenHands tends to generate minimal code, covering preprocessing, feature engineering, and modeling without deeper optimization. SELA's structured approach—dividing tasks into stages and enforcing diverse, sophisticated configurations—gives it a distinct advantage. MLAB frequently stalls in data inspection loops or produces weak models, while Agent Laboratory struggles with debugging loops and feature engineering failures, particularly with `DeepSeek-V2.5`. Even with `GPT-4o`, it has lower performance and occasional failures. Unlike general software engineering agents, which focus on task completion, machine learning agents must optimize performance based on data feedback. SELA excels by systematically exploring diverse strategies to maximize results.

H Cost-effectiveness Analysis

We conduct multiple trials of execution of each method to estimate the average running cost for the LLM-based baselines. As shown in Table 11, all methods incur relatively low costs to complete a single machine learning task. Among these, AIDE exhibits the lowest execution cost, due to the lack of stage-wise planning, resulting in fewer token generations compared to the other approaches. Additionally, SELA, which employs Data Interpreter as its base experimenter, is less costly than Data Interpreter itself. This efficiency is largely due to SELA's state-saving and loading mechanism, which reduces the generation of repeated tasks and code.

	Cost per ML task ($)
Data Interpreter (k=10)	0.07
AIDE (k=10)	0.01
SELA (k=10)	0.05

Table 11: Estimated costs of agent-based frameworks utilizing DeepSeekV2.5 on a single machine learning dataset over k iterations/rollouts.

I Case Study

I.1 Overview of SELA's search process

```
Number of simulations: 10
[Node 0]
Plans:
1. Perform exploratory data analysis on the train and dev datasets
2. Preprocess the train, dev, and test datasets
3. Perform feature engineering on the train, dev, and test datasets
4. Train multiple models and evaluate their performance
5. Train a weighted ensemble model using the best performing models
6. Evaluate the ensemble model on the dev set and save predictions
7. Generate predictions for the test set and save them
Simulated: True
Score: avg score: 0.6150206840685731, simulated score: {'train_score': 1.0, 'dev_score':
     0.6855841857240594, 'test_score': 0.6814818772150697, 'score': 0.6855841857240594}, Visits: 10

    [Node 0-0]
    Plans:
    3. Perform feature engineering on the train, dev, and test datasets by creating new features that
         calculate the magnitude of the vectorial velocities and accelerations to capture the overall
         movement intensity.
    Simulated: True
    Score: avg score: 0.6507249985568175, simulated score: {'train_score': 0.982920964830782, 'dev_score':
         0.6420233166755841, 'test_score': 0.647550336228104, 'score': 0.6420233166755841}, Visits: 2

        [Node 0-0-0]
        Plans:
        4. Train a Random Forest classifier to leverage its ability to handle high-dimensional data and
             capture non-linear relationships, and evaluate its performance
        Simulated: False
        Score: avg score: 0, simulated score: {}, Visits: 0

        [Node 0-0-1]
        Plans:
        4. Train multiple models, including a Support Vector Machine (SVM) with a radial basis function
             (RBF) kernel, and evaluate their performance.
        Simulated: False
        Score: avg score: 0, simulated score: {}, Visits: 0

        [Node 0-0-2]
        Plans:
        4. Implement a Neural Network with multiple layers to capture the hierarchical patterns in the data
             and evaluate its performance
        Simulated: True
        Score: avg score: 0.6594266804380511, simulated score: {'train_score': 1.0, 'dev_score':
             0.6594266804380511, 'test_score': 0.6702614538699305, 'score': 0.6594266804380511}, Visits: 1

        [Node 0-0-3]
        Plans:
        4. Train multiple models, apply an ensemble method like Gradient Boosting to combine them, and
             evaluate their performance
        Simulated: False
        Score: avg score: 0, simulated score: {}, Visits: 0

        [Node 0-0-4]
        Plans:
        4. Train multiple models, perform hyperparameter tuning using Grid Search or Random Search, and
             evaluate their performance
        Simulated: False
        Score: avg score: 0, simulated score: {}, Visits: 0

    [Node 0-1]
    Plans:
    3. Perform feature engineering on the train, dev, and test datasets by generating time-based features,
         such as the difference between consecutive frames, to capture the rate of change in movements.
    Simulated: True
    Score: avg score: 0.6464940718972336, simulated score: {'train_score': 1.0, 'dev_score':
         0.5985614604756948, 'test_score': 0.5857379626419719, 'score': 0.5985614604756948}, Visits: 2

        [Node 0-1-0]
        Plans:
        4. Train a Random Forest classifier to leverage its ability to handle high-dimensional data and
             capture non-linear relationships
        Simulated: False
        Score: avg score: 0, simulated score: {}, Visits: 0

        [Node 0-1-1]
        Plans:
        4. Train multiple models, including a Support Vector Machine (SVM) with a radial basis function
             (RBF) kernel, and evaluate their performance to model the complex decision boundaries between
             different gesture phases.
        Simulated: True
        Score: avg score: 0.6944266833187726, simulated score: {'train_score': 1.0, 'dev_score':
             0.6944266833187726, 'test_score': 0.6928451194338062, 'score': 0.6944266833187726}, Visits: 1
```

```
    [Node 0-1-2]
    Plans:
    4. Implement a Neural Network with multiple layers to capture the hierarchical patterns in the data and evaluate its performance
    Simulated: False
    Score: avg score: 0, simulated score: {}, Visits: 0

    [Node 0-1-3]
    Plans:
    4. Train multiple models, apply an ensemble method like Gradient Boosting to combine them, and evaluate their performance
    Simulated: False
    Score: avg score: 0, simulated score: {}, Visits: 0

    [Node 0-1-4]
    Plans:
    4. Train multiple models and perform hyperparameter tuning using techniques like Grid Search or Random Search to optimize and evaluate their performance.
    Simulated: False
    Score: avg score: 0, simulated score: {}, Visits: 0

[Node 0-2]
Plans:
3. Perform feature engineering on the train, dev, and test datasets by creating features that represent the spatial relationships between different body parts, such as the distance between the hands and the head.
Simulated: True
Score: avg score: 0.6296836159165489, simulated score: {'train_score': 0.7619969104124632, 'dev_score': 0.5997286931710517, 'test_score': 0.604077566134264, 'score': 0.5997286931710517}, Visits: 3

    [Node 0-2-0]
    Plans:
    4. Train a Random Forest classifier to leverage its ability to handle high-dimensional data and capture non-linear relationships, and evaluate its performance
    Simulated: False
    Score: avg score: 0, simulated score: {}, Visits: 0

    [Node 0-2-1]
    Plans:
    4. Train multiple models, including a Support Vector Machine (SVM) with a radial basis function (RBF) kernel, and evaluate their performance to model the complex decision boundaries between different gesture phases.
    Simulated: True
    Score: avg score: 0.6446610772892973, simulated score: {'train_score': 0.9952809245924918, 'dev_score': 0.6372459669415207, 'test_score': 0.6423549137767338, 'score': 0.6372459669415207}, Visits: 2

        [Node 0-2-1-0]
        Plans:
        5. Train a weighted ensemble model using the best performing models from task 4
        Simulated: False
        Score: avg score: 0, simulated score: {}, Visits: 0

        [Node 0-2-1-1]
        Plans:
        5. Using the models that performed best in task 4, train a weighted ensemble model to improve overall performance.
        Simulated: False
        Score: avg score: 0, simulated score: {}, Visits: 0

        [Node 0-2-1-2]
        Plans:
        5. Develop a weighted ensemble model by integrating the top-performing models from task 4, ensuring to evaluate and adjust the weights for optimal performance.
        Simulated: True
        Score: avg score: 0.6520761876370741, simulated score: {'train_score': 1.0, 'dev_score': 0.6520761876370741, 'test_score': 0.6563435152603494, 'score': 0.6520761876370741}, Visits: 1

        [Node 0-2-1-3]
        Plans:
        5. Train a weighted ensemble model by combining the predictions of the top-performing models from task 4 to improve overall performance.
        Simulated: False
        Score: avg score: 0, simulated score: {}, Visits: 0

        [Node 0-2-1-4]
        Plans:
        5. Develop a weighted ensemble model by combining the top-performing models from task 4, ensuring to optimize the weights for improved performance.
        Simulated: False
        Score: avg score: 0, simulated score: {}, Visits: 0

    [Node 0-2-2]
    Plans:
```

```
        4. Implement a Neural Network with multiple layers to capture the hierarchical patterns in the data
             and evaluate its performance
        Simulated: False
        Score: avg score: 0, simulated score: {}, Visits: 0

        [Node 0-2-3]
        Plans:
        4. Train multiple models, apply an ensemble method like Gradient Boosting to combine them, and
             evaluate their performance
        Simulated: False
        Score: avg score: 0, simulated score: {}, Visits: 0

        [Node 0-2-4]
        Plans:
        4. Perform hyperparameter tuning using Grid Search or Random Search to train multiple models and
             evaluate their performance
        Simulated: False
        Score: avg score: 0, simulated score: {}, Visits: 0

    [Node 0-3]
    Plans:
    3. Apply feature selection techniques such as Recursive Feature Elimination (RFE) or SelectKBest to
         identify and retain the most important features in the train, dev, and test datasets.
    Simulated: True
    Score: avg score: 0.49056683315196203, simulated score: {'train_score': 0.9988177730410426,
         'dev_score': 0.51620611302976, 'test_score': 0.525989891002361, 'score': 0.51620611302976},
         Visits: 2

        [Node 0-3-0]
        Plans:
        4. Train a Random Forest classifier to leverage its ability to handle high-dimensional data and
             capture non-linear relationships, and evaluate its performance.
        Simulated: False
        Score: avg score: 0, simulated score: {}, Visits: 0

        [Node 0-3-1]
        Plans:
        4. Train multiple models, including a Support Vector Machine (SVM) with a radial basis function
             (RBF) kernel, and evaluate their performance to model the complex decision boundaries between
             different gesture phases.
        Simulated: True
        Score: avg score: 0.4649275532741641, simulated score: {'train_score': 0.7299159411193588,
             'dev_score': 0.4649275532741641, 'test_score': 0.4631598897487413, 'score':
             0.4649275532741641}, Visits: 1

        [Node 0-3-2]
        Plans:
        4. Implement and train a Neural Network with multiple layers to capture hierarchical patterns in
             the data and evaluate its performance
        Simulated: False
        Score: avg score: 0, simulated score: {}, Visits: 0

        [Node 0-3-3]
        Plans:
        4. Train multiple models, apply an ensemble method like Gradient Boosting to combine them, and
             evaluate their performance
        Simulated: False
        Score: avg score: 0, simulated score: {}, Visits: 0

        [Node 0-3-4]
        Plans:
        4. Train multiple models, perform hyperparameter tuning using techniques like Grid Search or Random
             Search, and evaluate their performance
        Simulated: False
        Score: avg score: 0, simulated score: {}, Visits: 0

    [Node 0-4]
    Plans:
    3. Create interaction features by combining existing features, such as the product of velocity and
         acceleration, to capture complex relationships in the train, dev, and test datasets
    Simulated: False
    Score: avg score: 0, simulated score: {}, Visits: 0

Generated 29 unique codes.
Best node: 0-1-1, score: {'train_score': 1.0, 'dev_score': 0.6944266833187726, 'test_score':
     0.6928451194338062, 'score': 0.6944266833187726}
Dev best node: 0-1-1, score: {'train_score': 1.0, 'dev_score': 0.6944266833187726, 'test_score':
     0.6928451194338062, 'score': 0.6944266833187726}
```

In this case study, we demonstrate how SELA conducts a search cycle using MCTS:

Pre-search Step: Initialization

SELA begins by defining high-level stages, such as exploratory data analysis, data preprocessing, feature engineering, and model training, which structure the overall machine learning workflow. During the search, SELA populates these stages with specific insights, which act as experimental configurations for

simulation.

Step 1 & 2: Selection and Expansion

SELA leverages MCTS to explore specific stages like feature engineering and model training. For example, in one iteration, SELA selects Node 0-1. This node corresponds to a stage insight that generates time-based features, expanding into five child nodes representing various model specifications and training strategies, such as Random Forests, Support Vector Machines, Neural Networks, Gradient Boosting, or Grid Search.

Step 3: Simulation

Next, SELA samples one of the expanded child nodes for simulation. For instance, when Node 0-1-1 is chosen, SELA runs a complete experiment where time-based feature engineering (Node 0-1) is followed by training a Support Vector Machine (SVM) with a kernel specified by Node 0-1-1. The simulation yields an evaluation score.

Step 4: Backpropagation

After the simulation, the resulting performance score is propagated back through the tree. For example, after simulating Node 0-1-1, MCTS updates the numeric feedback for its parent nodes, such as Node 0-1 and Node 0. The search cycle repeats from Steps 1 to 4 until a stopping condition is reached.

Post-search Step: Best Node Selection

In the final phase, SELA selects the node representing the best-performing solution. In this example, Node 0-1-1, using an SVM with an RBF kernel, achieved the highest score in the current dataset by combining effective feature engineering with advanced model training. SELA then presents the code associated with this node as the optimal solution.