AutoML
Final Project

Ran MEI Ling ZHAO Jiawei HE Bowen XIAO
Zeyi WEN

Abstract

We built a reproducible AutoML pipeline for LLM source attribution as a course project, focusing on clear staging, fair comparisons, and manageable compute. First, we standardize inputs by freezing a pretrained DeBERTa-v3 encoder and caching mean-pooled, mask-aware sentence embeddings, so all classifiers see identical features. On these fixed embeddings, we evaluate three representative models—LightGBM, SVM, and MLP—under a unified cross-validation protocol. We compare four hyperparameter optimization strategies, including Grid Search, Random Search, TPE via Optuna, and SMAC, using consistent, configurable search spaces and record per-trial parameters, CV logloss, and histories for repeatable analysis. To control runtime, the pipeline supports full-grid runs or subset evaluation via a trial budget, and provides a post-processing utility that extracts the first N trials to regenerate curves and submissions without rerunning training. Empirically, Random Search is a strong low-budget baseline for smoother landscapes like LightGBM and SVM, while Bayesian methods—especially SMAC—show late-stage gains on the non-convex MLP, surpassing static grids and memoryless sampling as the budget grows. Overall, decoupling feature extraction from tuning reduces compute and simplifies experimentation, and the delivered code offer a practical template for completing AutoML assignments that can standardize features, unify evaluation, log everything, and compare strategies across models and budgets.

1. Introduction

The generative capabilities of Large Language Models (LLMs) have precipitated a fundamental epistemic shift in the digital information ecosystem. As noted in the recent survey by Wu et al. (2025), the distinction between human-authored discourse and machine-generated content has blurred to the point of indistinguishability, creating an urgent need for robust "source attribution" mechanisms. The capacity to trace a text sequence back to its originating model—whether GPT-4, Llama-2, or others—is no longer merely an academic exercise but a critical requirement for enforcing intellectual property rights and maintaining academic integrity. The H2O.ai "Predict the LLM" competition encapsulates this challenge, framing model attribution as a fine-grained classification task where decision boundaries are defined by subtle statistical artifacts inherent to the decoding strategies of different architectures.

Current methodologies for machine-generated text detection predominantly operate under black-box constraints, relying heavily on the representational power of the feature extractor and the generalization capability of the downstream classifier. While feature extraction has seen standardization through Transformer-based encoders like DeBERTa (He et al., 2021), the optimization of the classifier remains a source of considerable variance. Specifically, the interplay between high-dimensional embedding spaces and classifier hyperparameters creates a non-convex optimization landscape that resists manual tuning.

Existing literature often applies hyperparameter optimization (HPO) strategies indiscriminately. This approach ignores the implication of the "No Free Lunch" theorem: that the efficacy of an optimizer is contingent upon the structure of the loss landscape. A strategy that efficiently navigates the discrete loss function of a Gradient Boosting Machine may prove inefficient for the smooth manifold of a Neural Network.

In this work, we present a systematic Automated Machine Learning (AutoML) framework to rigorously evaluate HPO strategies for LLM attribution. To ensure internal validity, we establish a unified infrastructure utilizing frozen **DeBERTa-v3-base** embeddings as a controlled variable. Upon this foundation, we construct a 3×4 experimental matrix to disentangle the relationship between model architecture and optimization strategy. We examine three distinct classification paradigms:

1. **Gradient Boosting Decision Trees (LightGBM):** Representing ensemble learning with leaf-wise growth (Ke

et al., 2017).
2. **Support Vector Machines (SVM):** Representing convex optimization via margin maximization (Cortes & Vapnik, 1995).
3. **Multi-Layer Perceptrons (MLP):** Representing non-convex optimization via backpropagation (Rumelhart et al., 1986).

Across these architectures, we investigate four tiers of optimization complexity, explicitly positioning Random Search as our primary scientific control:

- **Random Search (Baseline):** Leveraging the low effective dimensionality hypothesis proposed by Bergstra & Bengio (2012b), this serves as our stochastic baseline to benchmark the efficiency of more complex methods.
- **Grid Search:** Used as a deterministic benchmark to evaluate the coverage efficiency of stochastic sampling in lower-dimensional subspaces.
- **Tree-structured Parzen Estimator (TPE):** A Bayesian approach modeling $p(\theta|y)$ to maximize Expected Improvement, implemented via Optuna (Akiba et al., 2019).
- **Sequential Model-based Algorithm Configuration (SMAC):** A Bayesian optimization method utilizing Random Forests to model $p(y|\theta)$, specifically designed to handle conditional hyperparameters (Hutter et al., 2011).

Our primary contribution is a comprehensive benchmark demonstrating that the marginal utility of advanced HPO is highly heterogeneous across model architectures. We empirically verify that while Bayesian methods are critical for navigating the complex, non-convex parameter spaces of MLPs to unlock superior final performance, they incur a significant exploration cost before converging. Conversely, in the smoother landscapes of SVMs and LightGBM, we observe that the robust Random Search baseline remains exceptionally competitive, outperforming or matching sophisticated optimizers within a standard trial budget. Furthermore, we demonstrate that a properly optimized lightweight classifier on top of DeBERTa embeddings can achieve competitive attribution accuracy with a fraction of the computational cost required for end-to-end fine-tuning.

2. Related Work

Automated Machine Learning (AutoML) has been an active research area aimed at reducing human effort in model design. The field encompasses several interrelated components, including hyperparameter optimization (HPO), neural architecture search (NAS), and the development of practical pipelines that combine multiple strategies for performance improvement.

We will first review fundamental techniques for hyperparameter optimization, including both classical approaches and their recent developments. Next is about neural architecture search, which extends the concept of automated optimization to the design of model structures. Finally, we examine recent practical applications of AutoML, with a focus on large-scale LLM classification competitions, highlighting different approaches and lessons learned from top-performing solutions.

2.1. Hyperparameter Optimization

2.1.1. Definition

Hyperparameter optimization aims to automatically find the optimal configuration of hyperparameters for a given learning algorithm, improving model performance without extensive manual tuning. Classical approaches include grid search, random search, and Bayesian optimization. Grid search evaluates all combinations of parameters exhaustively, while random search samples configurations uniformly. Bayesian optimization models the performance landscape and selects promising configurations iteratively, often leading to better efficiency. Other methods, such as gradient-based optimization or evolutionary algorithms, have also been applied for continuous or structured hyperparameter spaces.

2.1.2. HPO Development Overview

Hyperparameter optimization has evolved from simple exhaustive searches to more sophisticated and efficient strategies. We highlight several representative approaches:

- **Random Search and Grid Search:** These classical methods provide a baseline for hyperparameter tuning. Grid search exhaustively evaluates all combinations of predefined hyperparameter values, ensuring that no configuration is missed. Random search(Bergstra et al., 2011), in contrast, samples configurations uniformly across the search space, often achieving comparable or better results than grid search (Bergstra & Bengio, 2012a)in high-dimensional spaces due to better coverage efficiency.
- **Bayesian Optimization:** Bayesian optimization(Snoek et al., 2012) models the unknown performance function with a surrogate, typically a Gaussian Process, and uses an acquisition function to select promising hyperparameter configurations iteratively. This approach balances exploration and exploitation, efficiently identifying high-performing configurations with fewer evaluations compared to exhaustive or random search.
- **Multi-fidelity Optimization (Successive Halving / Hyperband):** Evaluating all candidate configurations on the full dataset can be computationally expensive. Multi-fidelity methods reduce resource usage by evaluating candidates on subsets of data or for fewer training epochs(Ke et al., 2017). Successive Halving progressively allocates more resources to promising configurations, while Hyperband(Li et al., 2018) extends this idea by combining

multiple halving schedules, improving efficiency for large-scale hyperparameter search.

- **Meta-learning and Transfer-based HPO:** These methods leverage prior knowledge from related tasks to guide the hyperparameter search(Yogatama & Mann, 2014). By learning patterns of effective configurations from historical experiments, the search process can start closer to promising regions, reducing the number of evaluations needed and improving convergence speed. This is particularly useful in scenarios where repeated tuning is required across similar datasets or models.

Each of these methods represents a step in the evolution of HPO techniques, from simple exhaustive search to intelligent, resource-efficient, and knowledge-informed strategies, thus providing the foundation for modern AutoML pipelines.

2.2. Neural Architecture Search

2.2.1. Definition

Neural Architecture Search is a subfield of AutoML focused on automatically designing the structure of neural networks. Unlike hyperparameter optimization, which tunes fixed model parameters, NAS searches over possible network architectures to identify the most effective design for a given task.

2.2.2. NAS Method Categories

NAS methods are generally categorized into three main types:

- **Reinforcement Learning-based NAS:** These methods treat architecture design as a sequential decision-making problem(Zoph & Le, 2017). A controller, often implemented as an RNN, proposes candidate architectures. Each architecture is trained and evaluated, and the performance feedback is used to update the controller policy via reinforcement learning. This approach can discover high-performing architectures but often requires extensive computational resources.
- **Evolutionary Algorithm-based NAS:** Inspired by biological evolution, these methods maintain a population of architectures(Real et al., 2017). Mutation and crossover operations generate new candidates, which are evaluated on the task. High-performing architectures are selected to form the next generation. Evolutionary NAS is flexible and can explore diverse architectures, but convergence can be slow for large search spaces.
- **Gradient-based (Differentiable) NAS:** These methods relax the discrete architecture search space to a continuous space, allowing the use of gradient descent to optimize architecture parameters alongside network weights. Differentiable NAS, such as DARTS(Liu et al., 2019), significantly reduces computational cost compared to RL or evolutionary approaches and enables efficient search over complex architectures.

2.2.3. NAS Process Overview

The typical NAS process involves the following steps:

1. **Define Search Space:** Specify possible building blocks, operations (e.g., convolution, pooling), and connectivity rules.
2. **Search Strategy:** Choose an optimization method, such as reinforcement learning, evolutionary algorithms, or gradient-based optimization.
3. **Architecture Evaluation:** Train candidate architectures on the task and measure performance using a validation metric.
4. **Selection and Iteration:** Update the search strategy based on performance, iteratively refining the candidate architectures until convergence or resource limits are reached.

2.2.4. Advantages and Limitations

NAS has demonstrated the ability to discover architectures that outperform manually designed networks across multiple domains, including image recognition and natural language processing. Its main advantages are automation of design, potential discovery of novel architectures, and adaptability to specific tasks. Limitations include high computational cost, the need for careful definition of the search space, and potential overfitting to the validation dataset if not properly regularized.

2.3. Top Solutions In Kaggle Competition

H2o.ai Predict the LLM(H2O). This competition focused on predicting the next token generated by an LLM, given a fixed context and system prompt. The top teams approached the task not as traditional language modeling, but as a structured classification problem, modeling the next token from a closed vocabulary. Below we summarize the four top-ranked approaches. What we are specially interesting about is how we encode the dataset, so that we can have a better representation for the later classificaiton work.

2.3.1. Rank 1: Feature Engineering + Gradient Boosted Trees

The first-place solution treated the task as a tabular classification problem. Instead of modeling raw text, the authors extracted a wide set of contextual statistics from the prompt, conversation history, and token candidates. Examples included: token frequency features, positional information, token–context similarity, and simple language-model–based heuristics.

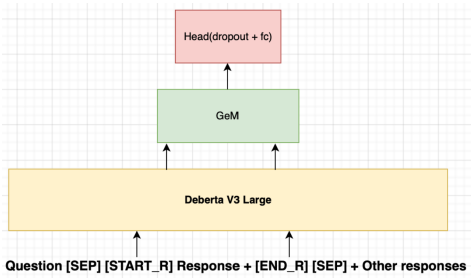

Figure 1. First place solution using feature engineering and gradient boosted trees

These features were fed into an ensemble of gradient boosting models (LightGBM and XGBoost). Each model produced a probability distribution over candidate tokens, and predictions were averaged across multiple seeds and folds. A calibrated softmax was applied to refine the final probability distribution. Their results show that a high-quality set of engineered features can outperform deep sequence models when the target space is fixed and relatively small.

2.3.2. Rank 2: RoBERTa-based Encoder Model

The second-place team used a pretrained RoBERTa encoder, converting the contextual text into dense vector representations. The model received concatenated segments of the system prompt, the user message, and the conversation history. A classification head was attached to predict among the set of allowed tokens.

The workflow included: (1) preprocessing the data into structured segments, (2) encoding with RoBERTa, (3) fine-tuning with cross-entropy on the candidate-token class set, and (4) aggregating outputs through multi-fold ensembling. Compared to gradient-boosting solutions, this model captured richer semantic interactions. However, it required careful input truncation and memory optimization due to long context sequences.

2.3.3. Rank 3: Electra Fine-tuning with Feature Augmentation

The third-place solution also relied on a pretrained transformer encoder, specifically Electra. The team combined textual embeddings with additional handcrafted features, such as token-level statistics and dialogue-structure indicators (turn index, role type, prompt length, etc.). These auxiliary features were appended to the transformer's pooled representation before classification.

Training involved multiple folds and heavy regularization. The authors reported that the augmented feature channels improved stability and helped the model differentiate between otherwise similar candidate tokens. Their final solution averaged several Electra checkpoints to smooth prediction variance.

2.3.4. Rank 4: DistilBERT with Lightweight Ensembling

The fourth-place team used a smaller transformer, DistilBERT, prioritizing speed and simplicity. The model was fine-tuned on the concatenated prompt and history using a standard classification head. Compared with the top entries, the approach did not use extensive feature engineering or auxiliary signals.

Performance was improved through a lightweight ensemble of independently fine-tuned DistilBERT models. The team reported that ensuring consistent preprocessing and careful token truncation had a greater effect than architectural complexity. Although more compact, the ensemble achieved strong accuracy due to stable training and consistent handling of edge cases.

2.3.5. Overall Analysis

The top solutions in the competition demonstrate different strategies for LLM authorship classification.

- **Decoder-based fine-tuning and ensemble methods:** Several solutions leveraged decoder-based LLMs with fine-tuning, combined with post-processing and ensemble strategies. These pipelines capture subtle generation characteristics of the text and often achieve the best performance.
- **Encoder-based feature extraction:** Other approaches employed pre-trained encoder models such as DeBERTa or BERT to extract textual embeddings, followed by a simple classification head. Despite the absence of complex ensemble or assignment techniques, these methods produced competitive results.

The above clues leads to the implication that for this classification task, while complex pipelines can improve accuracy, a well-configured encoder model alone is capable of capturing key textual features effectively. For similar classification tasks, encoder-based feature extraction provides a practical and efficient alternative. Also, the too much use of LLM can dilute the contribution of our AutoML model improvement on the hyper parameters. That is why after consideration, we decide to use a DeBERTa model for attribute extraction.

3. Methodology

Our pipeline is as follow: we first extract high-fidelity semantic embeddings with DeBERTa-v3 and cache them as static features, then systematically evaluate LightGBM, MLP, and SVM classifiers under multiple hyperparameter optimization strategies, including grid search and TPE and

SMAC. By decoupling feature extraction from model tuning, the design substantially reduces compute while enabling fair, repeatable comparisons of optimization efficiency and performance across models and budgets.

3.1. Data Preprocessing and Feature Extraction

To capture high-fidelity semantic representations of the input text, we forego traditional sparse vectorization methods (e.g., TF-IDF) in favor of a dense, contextualized embedding approach. We employ the **DeBERTa-v3-base** architecture (He et al., 2021) as our primary feature extractor. DeBERTa-v3 improves upon the original DeBERTa model (He et al., 2020) by replacing the standard Masked Language Modeling (MLM) objective with Replaced Token Detection (RTD), a sample-efficient pre-training task originally introduced in ELECTRA (Clark et al., 2020). By distinguishing between "real" and "replaced" input tokens generated by a generator network, the model learns more discriminative feature representations, which is critical for the subtle stylistic differences inherent in LLM source attribution.

3.1.1. Tokenization and Input Formatting

Raw text sequences are processed using the DeBERTa-v3 tokenizer, which is based on the **SentencePiece** algorithm (Kudo & Richardson, 2018). Unlike word-level tokenizers that struggle with out-of-vocabulary terms, SentencePiece treats the input as a raw byte stream, enabling robust subword segmentation. We implement a uniform sequence length of $L = 512$ tokens. Sequences exceeding this limit are truncated, while shorter sequences are padded with a special `[PAD]` token. Given the reliance of Transformer architectures on full-context attention, minimal cleaning is applied to the raw text (e.g., preserving punctuation and casing) to retain stylistic markers characteristic of specific LLMs. The implementation utilizes the Hugging Face Transformers library (Wolf et al., 2020).

3.1.2. Embedding Extraction and Pooling

For every input sequence x_i, we perform a forward pass through the pre-trained network to obtain the last hidden layer states $H \in \mathbb{R}^{L\times d}$, where $d = 768$ represents the hidden dimension size. While the special classification token (`[CLS]`) is traditionally used for sentence-level tasks in BERT-like models, recent empirical studies suggest that pooling over all token embeddings yields more robust representations for semantic classification (Reimers & Gurevych, 2019).

Consequently, we employ a **Mean Pooling** strategy that aggregates information across all valid tokens while strictly ignoring padding artifacts. Let $h_{i,j} \in \mathbb{R}^d$ be the vector representation of the j-th token in sequence i, and $m_{i,j} \in \{0, 1\}$ be the binary attention mask where 1 denotes a valid token and 0 denotes padding. The fixed-size sentence embedding vector v_i is computed as:

$$v_i = \frac{1}{\sum_{j=1}^{L} m_{i,j}} \sum_{j=1}^{L} h_{i,j} \cdot m_{i,j}$$

The resulting vectors $V = \{v_1, \ldots, v_N\}$ serve as the static input features for the subsequent hyperparameter optimization experiments. By freezing the DeBERTa parameters and caching these embeddings, we decouple feature extraction from classifier tuning, significantly reducing the computational overhead for the extensive search trials described in Section 4.

3.2. Classification Model

LightGBM (Light Gradient Boosting Machine) is an efficient implementation of the gradient boosting framework, optimized for large-scale and high-dimensional data. It was developed by Microsoft Research and widely used in industry and competitions. The core innovations include leaf-wise tree growth, histogram-based splitting, and direct support for categorical features (Ke et al., 2017).

In order to comprehensively evaluate different classification paradigms, we additionally include Support Vector Machines (SVM) and Multi-Layer Perceptrons (MLP) as comparative models. SVM represents a classical margin-based linear classifier with kernel extensions, while MLP illustrates the neural-network-based nonlinear modeling approach. These models complement LightGBM by offering diverse inductive biases and optimization strategies.

3.2.1. Principle Overview

LightGBM. LightGBM is based on the Gradient Boosting Decision Tree (GBDT) framework. Its main principles are:

- **Gradient Boosting Framework:** Iteratively build a series of weak learners (decision trees), where each new tree fits the residuals (negative gradients) of the previous model. The objective function is:

$$L^{(t)} = \sum_{i=1}^{n} l(y_i, \hat{y}_i^{(t-1)} + f_t(x_i)) + \Omega(f_t),$$

  where l is the loss function, Ω is the regularization term, and f_t is the tree at iteration t.
- **Leaf-wise Tree Growth:** LightGBM grows trees by splitting the leaf with the largest loss reduction, instead of level-wise splitting. This leaf-wise strategy reduces training loss faster, though it may lead to overfitting if not controlled by parameters like `num_leaves` or `max_depth`.
- **Histogram-based Splitting:** Continuous features are discretized into histogram bins, reducing memory consumption and accelerating split evaluation.

- **Categorical Feature Handling:** Supports direct processing of categorical features without one-hot encoding, by finding the optimal category grouping for splits.
- **Parallel and Incremental Training:** Supports feature- and data-parallel distributed training and allows incremental learning with early stopping.

Support Vector Machine (SVM). SVM seeks an optimal separating hyperplane that maximizes the margin between classes. For non-linearly separable data, kernel functions (e.g., RBF, polynomial) implicitly map inputs to a higher-dimensional feature space where linear separation becomes feasible. Its core formulation solves:

$$\min_{w,b,\{\xi_i\}_{i=1}^n} \quad \frac{1}{2}\|w\|^2 + C\sum_{i=1}^{n}\xi_i$$

$$\text{s.t.} \quad y_i\big(w^\top \phi(x_i) + b\big) \geq 1 - \xi_i,\ \xi_i \geq 0,\ i = 1, \ldots, n.$$

SVM is robust on high-dimensional sparse features and performs well with limited training data due to its margin-maximization principle.

Multi-Layer Perceptron (MLP). MLP is a feed-forward neural network composed of fully connected layers with nonlinear activation functions. Given input x, a two-layer MLP's prediction can be expressed as:

$$f(x) \;=\; W_2\,\sigma\big(W_1 x + b_1\big) + b_2,$$

where $\sigma(\cdot)$ is a non-linear activation such as ReLU. MLP learns complex nonlinear mappings through backpropagation and gradient-based optimization. Compared with tree models and margin-based methods, its flexibility arises from the universal approximation capability of neural networks.

3.2.2. Workflow

LightGBM Workflow.

1. **Data Preprocessing:** Construct histogram bins for continuous features, handle missing values, and encode categorical features if necessary.
2. **Iterative Tree Training:** For each boosting iteration:
   - Compute negative gradients (residuals) for current predictions.
   - Select the leaf with the largest gain and split (leaf-wise growth).
   - Update the model by adding the new tree.
3. **Regularization and Pruning:** Control tree complexity with `num_leaves`, `max_depth`, and learning rate to prevent overfitting.
4. **Model Output:** Combine all trained trees; final prediction is the weighted sum of tree outputs.

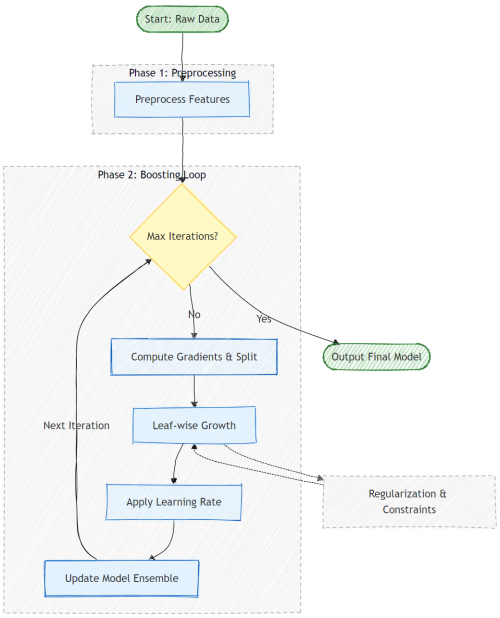

Figure 2. Structured flowchart illustrating the LightGBM training process

SVM Workflow.

1. **Feature Scaling:** Standardize embeddings to zero mean and unit variance.
2. **Kernel Selection:** Linear or RBF kernel depending on data structure.
3. **Optimization:** Solve convex quadratic programming to obtain support vectors.
4. **Prediction:** Decision function computed from support vector contributions.

MLP Workflow.

1. **Architecture Definition:** Choose number of layers, hidden units, activation functions.
2. **Forward and Backward Pass:** Compute output, evaluate loss, backpropagate gradients.
3. **Parameter Optimization:** Update weights via stochastic gradient descent or Adam.
4. **Regularization:** Apply dropout or weight decay to mitigate overfitting.

3.2.3. Advantage

We selected LightGBM as the classifier to optimize. Several factors motivated this choice.

First, LightGBM is highly efficient in handling large-scale and high-dimensional data, which is essential given the textual feature representations derived from encoder embeddings. Its leaf-wise tree growth and histogram-based splitting allow fast training without excessive memory us-

age, making it suitable for a quick iterative experimentation as we need to constantly change research settings.

Second, LightGBM can capture complex, non-linear relationships in the data without extensive feature engineering. Compared with other tree-based methods such as XGBoost or CatBoost, LightGBM generally provides faster training and lower memory footprint while maintaining competitive predictive performance. This combination of speed, efficiency, and accuracy makes it a practical choice.

Also, the flexibility of LightGBM allows straightforward integration into AutoML pipelines. This aligns with the overall goal of exploring automated model selection and hyperparameter tuning without spending excessive time on manual optimization.

SVM provides strong performance on high-dimensional sparse text features and benefits from convex optimization, ensuring global optima. However, SVM becomes computationally expensive on larger datasets and does not naturally scale as efficiently as tree-based boosting.

MLP offers powerful nonlinear modeling capacity but typically requires careful hyperparameter tuning, larger datasets, and longer training time to reach stable performance. Neural networks also demand more computational resources and regularization techniques to avoid overfitting.

Compared with these alternatives, LightGBM achieves an effective balance between speed, memory efficiency, and predictive accuracy. Its ability to capture complex feature interactions without extensive feature engineering and its compatibility with AutoML pipelines further reinforce its suitability for the current task.

3.3. HPO Algorithms

The optimization landscape for classifier hyperparameters is often complex, high-dimensional, and computationally expensive to traverse. To address these challenges, we move beyond baseline approaches to employ sophisticated Bayesian optimization strategies: the Tree-structured Parzen Estimator (TPE) and the Sequential Model-based Algorithm Configuration (SMAC).

3.3.1. Limitations of Baseline Methods

Classical methods like **Grid Search** and **Random Search** serve as essential baselines but suffer from significant limitations in complex high-dimensional spaces. Grid Search evaluates a fixed grid of points, leading to a computational cost that scales exponentially with the number of hyperparameters. **Random Search** samples uniformly across the space. While proven more efficient than Grid Search in many high-dimensional settings (Bergstra & Bengio, 2012b), it is an *uninformed* strategy: it does not leverage results from prior evaluations to guide subsequent sampling, leading to inefficient exploration when the evaluation budget is constrained.

Bayesian Optimization (BO) addresses these issues by modeling the unknown objective function $f(\mathbf{x})$ (the model's performance on hyperparameter configuration $\mathbf{x}$) to intelligently select the next configuration to evaluate.

3.3.2. Tree-structured Parzen Estimator (TPE)

The Tree-structured Parzen Estimator (TPE), implemented via Optuna (Akiba et al., 2019), is a sequential model-based optimization approach. Instead of using a Gaussian Process as a surrogate, TPE models the distributions directly using Parzen windows (or kernel density estimators).

TPE divides the historical hyperparameter configurations $\mathcal{D}$ into two groups based on a performance threshold y^*:

1. $\mathcal{L}$: The set of configurations leading to the top performances ($f(\mathbf{x}) < y^*$).
2. $\mathcal{G}$: The remaining set of configurations ($f(\mathbf{x}) \geq y^*$).

TPE then models two probability densities, $l(\mathbf{x}) = p(\mathbf{x}|y < y^*)$ and $g(\mathbf{x}) = p(\mathbf{x}|y \geq y^*)$. The threshold y^* is typically set as a quantile (e.g., the γ-quantile) of the observed function values.

The strategy proposes the next hyperparameter configuration $\mathbf{x}_{\text{next}}$ by maximizing the **Expected Improvement (EI)** acquisition function, which balances exploration and exploitation. Since $p(y|\mathbf{x})$ is difficult to compute directly, the Expected Improvement is calculated based on the two modeled densities $l(\mathbf{x})$ and $g(\mathbf{x})$.

The optimal $\mathbf{x}_{\text{next}}$ is approximated by maximizing the ratio $\frac{l(\mathbf{x})}{g(\mathbf{x})}$:

$$\mathbf{x}_{\text{next}} = \text{argmax}_{\mathbf{x}} \left(\text{EI}(\mathbf{x})\right) \propto \text{argmax}_{\mathbf{x}} \left(\frac{l(\mathbf{x})}{g(\mathbf{x})}\right)$$

Maximizing this ratio ensures that the algorithm prioritizes sampling configurations $\mathbf{x}$ that are highly probable under the good distribution $l(\mathbf{x})$ and less probable under the bad distribution $g(\mathbf{x})$.

Our implementation utilizes the **Optuna** framework (Akiba et al., 2019), which employs the TPE sampler by default. Optuna's key advantages include its dynamic search space definition, and its implementation of pruning (early stopping of unpromising trials), which enhances TPE's efficiency by quickly discarding poor hyperparameter configurations before they consume the full computational budget.

3.3.3. Sequential Model-based Algorithm Configuration (SMAC)

SMAC (Hutter et al., 2011) is an advanced Bayesian optimization framework that utilizes a **Random Forest** as its

surrogate model, $p(y|\mathbf{x})$. A key advantage of the Random Forest is its robust ability to handle heterogeneous and **conditional hyperparameters**, which often arise in complex configuration spaces.

The Random Forest models the conditional probability $p(y|\mathbf{x})$, providing both a mean prediction ($\mu(\mathbf{x})$) and a variance ($\sigma^2(\mathbf{x})$) for the objective function y at configuration $\mathbf{x}$. SMAC employs the Expected Improvement (EI) acquisition function to guide the search, calculating the expected gain from sampling $\mathbf{x}_{\text{next}}$ relative to the best observed performance y_{best}:

$$\text{EI}(\mathbf{x}) = \mathbb{E}[\max(y_{\text{best}} - f(\mathbf{x}), 0)]$$

The algorithm iteratively suggests the configuration $\mathbf{x}$ that maximizes $\text{EI}(\mathbf{x})$, evaluates it using cross-validation, and updates the Random Forest model with the new result.

A primary innovation in SMAC3 is its integration with **Hyperband** (Li et al., 2018), a multi-fidelity optimization technique. This integration enables SMAC to allocate computational resources more intelligently by evaluating configurations at different budget levels (e.g., fewer boosting rounds or epochs). Poor configurations are eliminated early through successive halving, dramatically reducing wasted computation while accelerating the discovery of high-performing solutions.

In this project, both TPE and SMAC are applied to optimize hyperparameters for LightGBM, MLP, and SVM models. The optimization history of each strategy is tracked and compared against Random Search results to empirically demonstrate their superior sample efficiency and convergence speed across various model types and computational budgets.

4. Experimental Setup

To ensure the fairness and systematic nature of the Hyperparameter Optimization (HPO) experiments, we defined rigorous search spaces for the three classifiers: LightGBM, Support Vector Machines (SVM), and Multi-Layer Perceptrons (MLP). These spaces cover the critical parameters that influence model performance. We utilized logarithmic sampling for multiplicative parameters (e.g., learning rates and regularization coefficients) to efficiently explore a wide range of magnitudes, and linear sampling for additive parameters. Table 1 summarizes the hyperparameter search ranges and categories used across our Grid Search, Random Search, TPE, and SMAC experiments.

5. Experimental Results and Analysis

Based on the comprehensive experimental data collected, we present a detailed analysis of the hyperparameter optimization performance for each classification model. This analysis is structured chronologically across three key evaluation milestones—10, 20, and 50 trials—to illustrate the temporal evolution of each optimization strategy. We specifically incorporate insights regarding the distinct convergence behaviors observed, particularly the late-stage breakthroughs in the MLP experiments.

5.1. LightGBM Analysis

In the initial exploratory phase of the first 10 trials, the optimization landscape was clearly defined by the efficiency of Random Search. This method immediately identified a superior configuration with a Log Loss of 1.377, effectively outperforming both SMAC3, which achieved 1.383, and Grid Search, which lagged at 1.409. This early stage highlighted a fundamental characteristic of the competing methods: Random Search successfully capitalized on the statistical probability of hitting a "good enough" solution quickly within a well-behaved search space. In contrast, Grid Search struggled significantly due to the inherent coarseness of its pre-defined grid, resulting in the highest error rate among the three strategies. SMAC3 performed competitively but had not yet accumulated sufficient historical data to build a surrogate model capable of surpassing the random baseline.

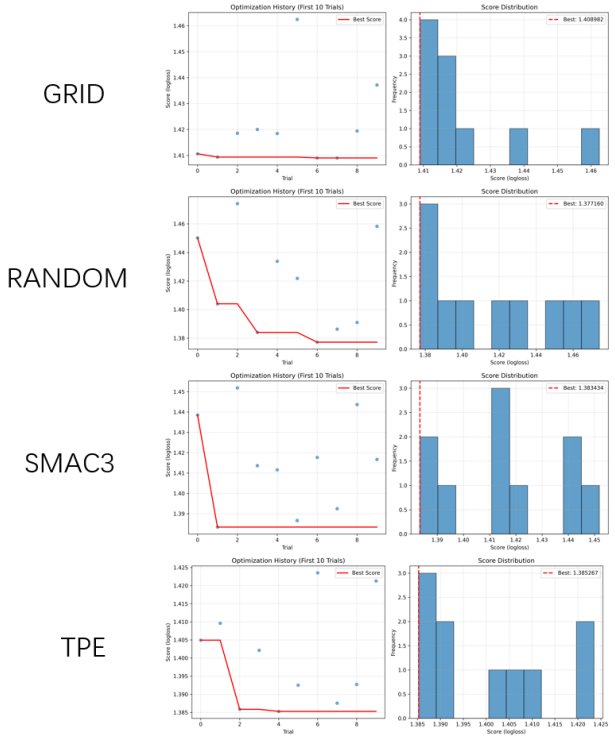

Figure 3. Performance comparison for LightGBM at 10 trials.

Table 1. Hyperparameter Search Spaces for LightGBM, SVM, and MLP Classifiers.

Model	Hyperparameter	Type	Search Range / Choices	Scale
LightGBM	`num_leaves`	Integer	$[20, 150]$	Linear
	`max_depth`	Integer	$[3, 12]$	Linear
	`learning_rate`	Float	$[0.01, 0.3]$	Logarithmic
	`n_estimators`	Integer	$[100, 1000]$	Linear
	`min_child_samples`	Integer	$[10, 100]$	Linear
	`subsample`	Float	$[0.6, 1.0]$	Linear
	`colsample_bytree`	Float	$[0.6, 1.0]$	Linear
	`reg_alpha` (L1)	Float	$[0.0, 10.0]$	Linear
	`reg_lambda` (L2)	Float	$[0.0, 10.0]$	Linear
SVM	`C` (Regularization)	Float	$[0.001, 100.0]$	Logarithmic
	`kernel`	Categorical	{rbf, linear, poly}	N/A
	`gamma` (Kernel Coeff.)	Float	$[0.0001, 10.0]$	Logarithmic
	`degree` (for 'poly')	Integer	$[2, 5]$	Linear
	`class_weight`	Categorical	{balanced, None}	N/A
MLP	`hidden_layer_sizes`	Categorical	{[128], [256], [512], [128, 64], [256, 128], [512, 256], [256, 128, 64], [512, 256, 128]}	N/A
	`learning_rate_init`	Float	$[0.0001, 0.01]$	Logarithmic
	`alpha` (L2 penalty)	Float	$[0.0001, 0.1]$	Logarithmic
	`batch_size`	Categorical	$\{32, 64, 128, 256\}$	N/A
	`activation`	Categorical	{relu, tanh}	N/A
	`max_iter`	Integer	$[200, 1000]$	Linear
	`early_stopping`	Categorical	{True, False}	N/A

Moving to the intermediate 20-trial mark, the performance dynamics shifted noticeably towards stagnation for the leading methods. Random Search failed to improve upon its initial discovery, remaining flat at a score of 1.377. This plateau is typical behavior for random sampling after an early "lucky" strike, as the probability of finding a marginally better solution decreases exponentially. Grid Search managed to refine its best score to 1.398, narrowing the gap slightly but still trailing significantly behind the other two methods. Similarly, SMAC3 showed minimal movement, improving only marginally to 1.383. This phase represented a critical plateau where none of the strategies could make a decisive breakthrough, suggesting that the "low-hanging fruit" of optimization had been harvested, and further improvements would require more sophisticated and targeted exploration.

By the final milestone of 50 trials, the systematic strength of Bayesian optimization became the defining factor. SMAC3 successfully converged, improving its score to 1.374, which virtually matched Random Search's best result of 1.373. This trajectory demonstrates a key trade-off: while Random Search is superior for immediate, low-budget results, SMAC3 possesses the capability to recover and refine its search to reach competitive performance given sufficient iterations. Grid Search, conversely, remained the least effective method, plateauing at 1.398 and failing to exploit the search space as effectively as the other two dynamic strategies.

5.2. MLP (Multi-Layer Perceptron) Analysis

The 10-trial phase for the MLP model presented a stark contrast in method efficacy compared to the tree-based model. Grid Search actually took the lead with a score of 1.327, followed closely by Random Search at 1.345. SMAC3, however, appeared to fail completely in this early stage, recording a very poor Log Loss of 1.442. The performance curves for Grid and Random Search appeared as "flat lines" almost immediately. This phenomenon is characteristic of the high-dimensional and complex parameter space of neural networks: Grid Search is limited by its fixed, discrete points and cannot improve once the best pre-defined combination is found, while Random Search, being memoryless, struggles to statistically hit a narrower, superior region after its initial lucky find.

At the 20-trial interval, the hierarchy remained largely unchanged, reinforcing the initial trends observed. Grid Search maintained its lead at 1.327, and Random Search improved slightly to 1.337. SMAC3 remained stuck at 1.442, unable

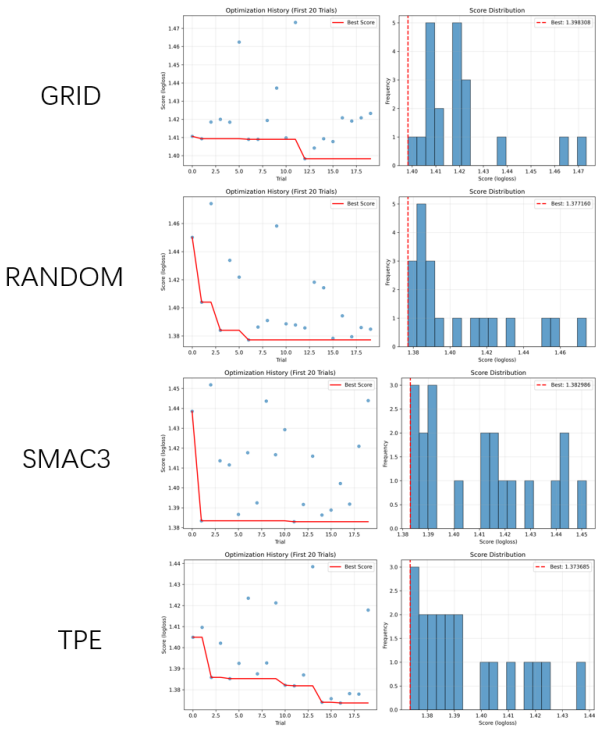

Figure 4. Performance comparison for LightGBM at 20 trials.

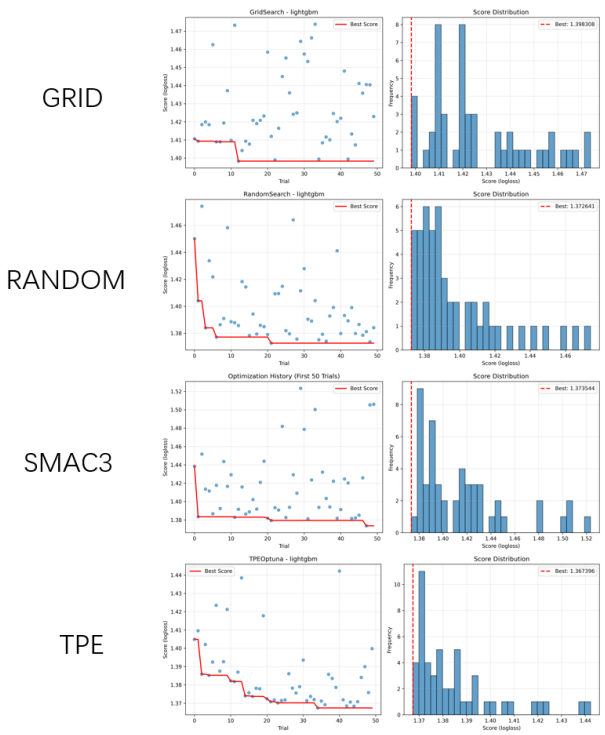

Figure 5. Performance comparison for LightGBM at 50 trials.

to find a better configuration. This persistence of poor performance for SMAC3 highlighted the ”exploration” cost inherent to Bayesian optimization. In such a vast and complex search space, the algorithm must spend a significant portion of its budget exploring diverse and potentially poor regions to build a reliable surrogate model of the loss landscape before it can begin to exploit promising areas.

The 50-trial milestone revealed the most dramatic turnaround of the entire experiment, directly explaining the ”late drop” observed in the SMAC3 curve. Unlike the flat lines of the other methods, SMAC3 leveraged its accumulated search history to identify a breakthrough configuration, dropping its score massively to 1.294. This late-stage improvement is the signature of Bayesian optimization shifting from exploration to exploitation. After building a sufficiently accurate probabilistic model of the hyperparameter space during the first 40+ trials, SMAC3 was finally able to pinpoint and fine-tune the global optimum that the blind Random Search and rigid Grid Search could not reach. This result proves that for complex models like MLPs, intelligent algorithms require a ”warm-up” period but ultimately unlock performance potential that simpler strategies miss.

5.3. SVM (Support Vector Machine) Analysis

In the first 10 trials for the SVM model, Random Search established a clear dominance with a score of 1.399, proving to be the most robust strategy for this specific problem. Grid Search followed at 1.410. SMAC3, similar to the MLP case, struggled severely in the beginning, posting a very high loss of 1.683. This indicates a misalignment between SMAC3’s initial exploration strategy and the SVM’s loss landscape, where it likely wasted early trials in poor regions of the parameter space while Random Search’s unbiased sampling provided immediate coverage of better areas.

As the budget increased to 20 trials, SMAC3 began a significant recovery. It improved its score drastically from 1.683 to 1.433, correcting its course as the surrogate model became more informed. However, it still trailed behind Random Search (1.399) and Grid Search (1.410), which remained stable. This phase illustrates the ”catch-up” dynamic of intelligent optimization: while it can recover from a bad start, the cost of that initial exploration can put it behind simpler methods in the medium term.

By the final 50-trial mark, the performance gap narrowed considerably. SMAC3 continued its improvement to reach 1.400, effectively catching up to Grid Search (1.397) and coming very close to Random Search. However, Random

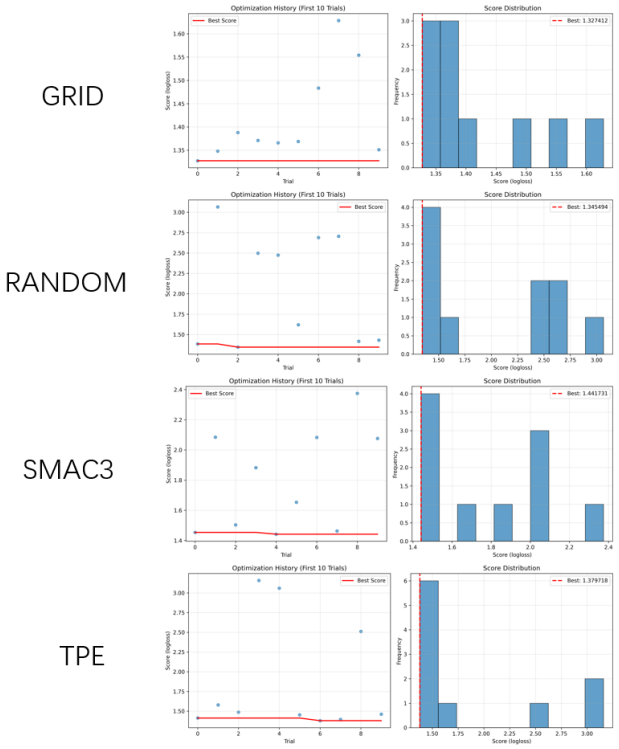

Figure 6. Performance comparison for MLP at 10 trials.

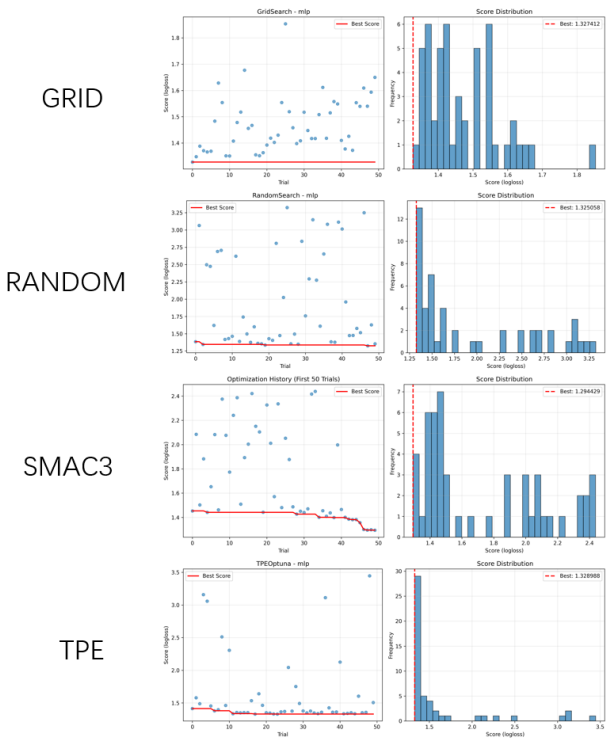

Figure 7. Performance comparison for MLP at 20 trials.

Search maintained its lead to the end, finishing with the best score of 1.392. This consistent leadership suggests that for this SVM problem, the hyperparameter space might be relatively smooth or compact, allowing unbiased random sampling to locate the optimal region more efficiently than the complex probabilistic modeling of SMAC3, which spent much of its budget recovering from a poor start.

6. Conclusion

In this comprehensive study, we conducted a rigorous evaluation of three distinct Hyperparameter Optimization (HPO) strategies—Grid Search, Random Search, and SMAC3 (Sequential Model-Based Algorithm Configuration)—across three diverse classification models: LightGBM, Multi-Layer Perceptron (MLP), and Support Vector Machine (SVM). By systematically comparing these methods across varying computational budgets (10, 20, and 50 trials), our experimental results reveal nuanced performance characteristics that challenge conventional assumptions about HPO algorithm superiority. The findings demonstrate that the effectiveness of an optimization strategy is not absolute but is highly dependent on the specific model architecture, the dimensionality of the search space, and the complexity of the hyperparameter landscape.

Our findings indicate that Random Search serves as an exceptionally strong and often underestimated baseline, particularly in low-budget scenarios and for models characterized by smoother or lower-dimensional loss landscapes. In the case of LightGBM and SVM, Random Search demonstrated a significant early advantage through fortunate discovery, achieving competitive or even superior performance within the initial 10 trials. For SVM specifically, Random Search maintained its performance lead throughout the entire 50-trial budget. This suggests that for certain problem spaces where the region of optimal hyperparameters is relatively broad, unbiased random sampling can be more efficient than complex probabilistic modeling, which may struggle with initial alignment or over-exploration in the early stages. The robustness of Random Search highlights its value as a first-line approach for rapid prototyping.

Grid Search, while systematic and deterministic, proved to be the least efficient method overall in our experiments. It consistently plateaued early across all three models, severely limited by the coarseness of the pre-defined grid. Its inherent inability to explore the vast space between grid points resulted in it being outperformed by both dynamic strategies (Random Search and SMAC3). This limitation highlights the inadequacy of Grid Search for fine-tuning high-dimensional models where the optimal configuration often

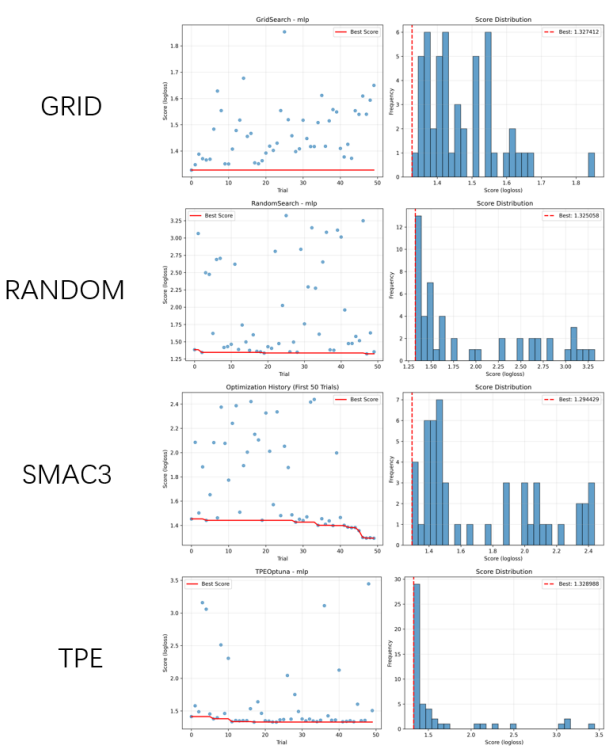

Figure 8. Performance comparison for MLP at 50 trials.

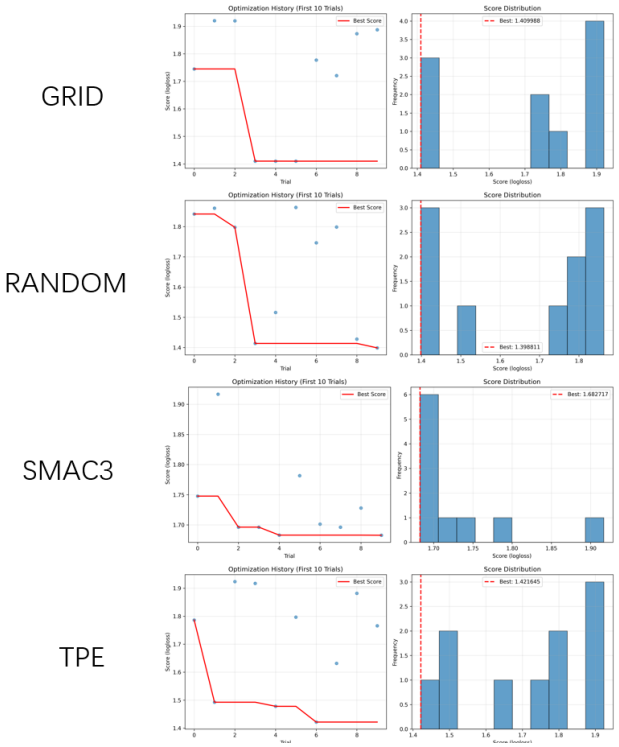

Figure 9. Performance comparison for SVM at 10 trials.

lies off-grid. While Grid Search provides a structured way to understand parameter sensitivity, its rigidity makes it computationally wasteful for finding global optima in complex landscapes.

The most significant and theoretically profound insight emerges from the performance of SMAC3 (Bayesian Optimization), particularly in the MLP experiments. The neural network's hyperparameter space is notoriously complex, non-convex, and high-dimensional. In this challenging environment, SMAC3 initially struggled, lagging behind other methods during the first 20 trials due to the necessary cost of exploration required to build its surrogate model. However, it demonstrated a dramatic "late-stage convergence" that validated its sophisticated approach. In the final phase (trials 20-50), SMAC3 leveraged its accumulated search history to identify a breakthrough configuration, achieving a Log Loss of 1.294, the best result recorded in the entire study. This behavior underscores the critical theoretical advantage of Bayesian optimization: its ability to intelligently shift from exploration to exploitation. Unlike Random Search, which stagnated after its early luck due to its memoryless nature, SMAC3 utilized historical data to navigate the complex landscape, unlocking performance potential that simpler strategies missed.

These results align with and extend the seminal findings of Bergstra and Bengio (2012)(**?**), confirming that while Random Search is efficient for simpler problems, intelligent methods like SMAC3 are essential for unlocking the full potential of complex architectures. The study suggests a clear, evidence-based heuristic for practitioners: for simple models, low-dimensional spaces, or extremely limited computational budgets, Random Search is a sufficient and efficient choice. However, for complex models like MLPs, or when the goal is to seek the absolute global optimum in a high-stakes environment, investing in the computational overhead and "warm-up" time of Bayesian optimization is not only justified but necessary.

Future work should explore several promising directions to build upon these findings. First, expanding the trial budget significantly beyond 50 iterations would be valuable to determine if SMAC3 can further widen the performance gap in LightGBM and SVM, potentially overcoming its slow start in those domains. Second, investigating multi-fidelity optimization techniques, such as Hyperband or BOHB, could offer a way to accelerate the "warm-up" phase of Bayesian methods by discarding poor configurations early. Finally, a deeper analysis of the specific hyperparameters that contributed most to the performance gains in SMAC3 could provide interpretable insights into the models' behavior,

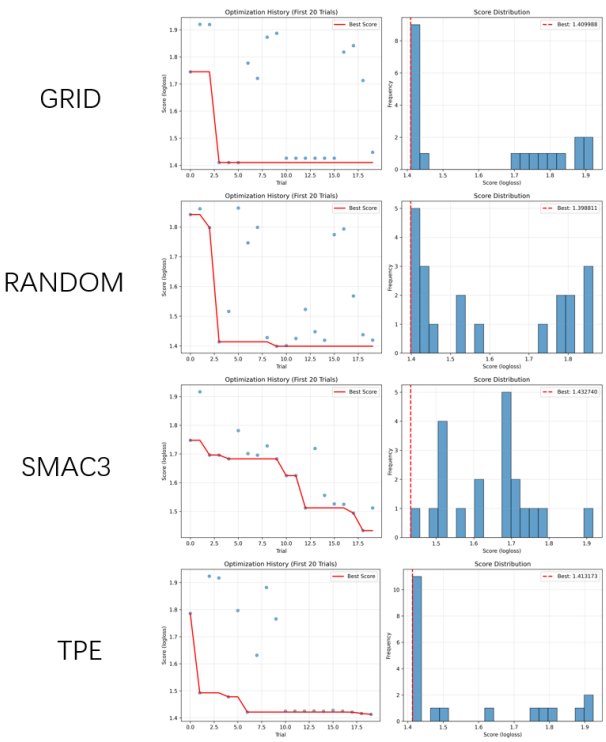

Figure 10. Performance comparison for SVM at 20 trials.

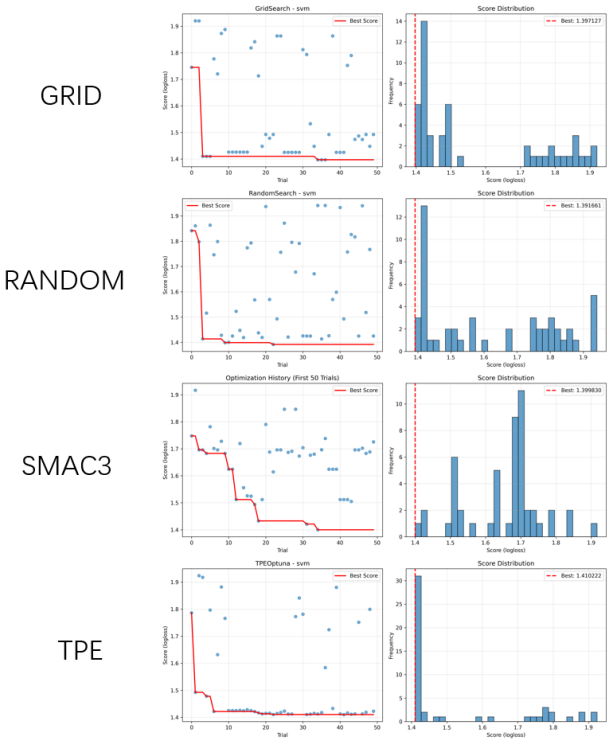

Figure 11. Performance comparison for SVM at 50 trials.

bridging the gap between black-box optimization and model understanding.

Accessibility

The source code of this project is publicly available on GitHub: https://github.com/Alune233/5003-Final-Project/

References

H2o.ai predict the llm (kaggle competition). https://www.kaggle.com/competitions/h2o-ai-predict-the-llm. Accessed: 2025-12-05.

Akiba, T., Sano, S., Yanase, T., Ohta, T., and Koyama, M. Optuna: A next-generation hyperparameter optimization framework. In *Proceedings of the 25th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, pp. 2623–2631, 2019.

Bergstra, J. and Bengio, Y. Random search for hyper-parameter optimization. *Journal of Machine Learning Research*, 13(10):281–305, 2012a.

Bergstra, J. and Bengio, Y. Random search for hyper-parameter optimization. *Journal of Machine Learning Research*, 13(2):281–305, 2012b.

Bergstra, J. S., Bardenet, R., Bengio, Y., and Kégl, B. Algorithms for hyper-parameter optimization. In *Advances in Neural Information Processing Systems 24 (NIPS 2011)*, pp. 2546–2554, Granada, Spain, 2011. Curran Associates, Inc.

Clark, K., Luong, M.-T., Le, Q. V., and Manning, C. D. Electra: Pre-training text encoders as discriminators rather than generators. In *International Conference on Learning Representations*, 2020.

Cortes, C. and Vapnik, V. Support-vector networks. *Machine Learning*, 20(3):273–297, 1995.

He, P., Liu, X., Gao, J., and Chen, W. Deberta: Decoding-enhanced bert with disentangled attention. In *International Conference on Learning Representations*, 2020.

He, P., Gao, J., and Chen, W. Debertav3: Improving deberta using electra-style pre-training with gradient-disentangled embedding sharing. *arXiv preprint arXiv:2111.09543*, 2021.

Hutter, F., Hoos, H. H., and Leyton-Brown, K. Sequential model-based optimization for general algorithm config-

uration. In *Learning and Intelligent Optimization: 5th International Conference, LION 5*, pp. 507–523. Springer, 2011.

Ke, G., Meng, Q., Finley, T., Wang, T., Chen, W., Ma, W., Ye, Q., and Liu, T.-Y. Lightgbm: A highly efficient gradient boosting decision tree. In *Advances in Neural Information Processing Systems*, volume 30, 2017.

Kudo, T. and Richardson, J. Sentencepiece: A simple and language independent subword tokenizer and detokenizer for neural text processing. In *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing: System Demonstrations*, pp. 66–71, 2018.

Li, L., Jamieson, K., DeSalvo, G., Rostamizadeh, A., and Talwalkar, A. Hyperband: A novel bandit-based approach to hyperparameter optimization. *Journal of Machine Learning Research*, 18(185):1–52, 2018.

Liu, H., Simonyan, K., and Yang, Y. Darts: Differentiable architecture search. In *International Conference on Learning Representations*, 2019.

Real, E., Moore, S., Selle, A., Saxena, S., Suematsu, Y. L., Tan, J., Le, Q. V., and Kurakin, A. Large-scale evolution of image classifiers. In *Proceedings of the 34th International Conference on Machine Learning*, 2017.

Reimers, N. and Gurevych, I. Sentence-bert: Sentence embeddings using siamese bert-networks. In *Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing and the 9th International Joint Conference on Natural Language Processing (EMNLP-IJCNLP)*, pp. 3982–3992, 2019.

Rumelhart, D. E., Hinton, G. E., and Williams, R. J. Learning representations by back-propagating errors. *Nature*, 323(6088):533–536, 1986.

Snoek, J., Larochelle, H., and Adams, R. P. Practical bayesian optimization of machine learning algorithms. In *Advances in Neural Information Processing Systems 25 (NIPS 2012)*, pp. 2960–2968, 2012.

Wolf, T., Debut, L., Sanh, V., Chaumond, J., Delangue, C., Moi, A., Cistac, P., Rault, T., Louf, R., Funtowicz, M., et al. Transformers: State-of-the-art natural language processing. In *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing: System Demonstrations*, pp. 38–45, 2020.

Wu, J., Yang, S., Zhan, R., Yuan, Y., Chao, L. S., and Wong, D. F. A survey on llm-generated text detection: Necessity, methods, and future directions. *Computational Linguistics*, 51(1):275–338, 2025.

Yogatama, D. and Mann, G. Efficient transfer learning method for automatic hyperparameter tuning. In *Proceedings of the 17th International Conference on Artificial Intelligence and Statistics (AISTATS)*, volume 33 of *Proceedings of Machine Learning Research*, pp. 1077–1085, 2014.

Zoph, B. and Le, Q. V. Neural architecture search with reinforcement learning. In *International Conference on Learning Representations*, 2017.

A. Contribution Statement

The project tasks were distributed using a method-centric approach. All members utilized a unified dataset (DeBERTa embeddings) and identical target models (LightGBM, SVM, MLP), while individually focusing on distinct Hyperparameter Optimization (HPO) strategies and corresponding report sections.

- **Student Jiawei He (Data Foundation & Baseline):** Led the data engineering pipeline by executing DeBERTa embedding extraction and defining the unified hyperparameter search space (JSON) for the team. Implemented the *Random Search* baseline experiments. Authored the *Introduction* and *Data Preprocessing* (Section 3.1).
- **Student Ling Zhao (Grid Search & Model Theory):** Implemented the *Grid Search* experiments, handling the necessary discretization of search spaces for complex models. Conducted the literature review and authored the *Related Work* and *Prediction Models* (Section 3.2), detailing the algorithmic paradigms of LightGBM, SVM, and MLP.
- **Student Ran Mei (Bayesian Optimization - TPE):** Implemented the *TPE (Tree-structured Parzen Estimator)* experiments using Optuna and generated parameter importance analyses. Responsible for the theoretical methodology, authoring the *HPO Algorithms* (Section 3.3) with mathematical formulations, and the *Experimental Setup* (Section 4).
- **Student Bowen Xiao (Advanced HPO - SMAC & Analysis):** Implemented the *SMAC (Sequential Model-based Algorithm Configuration)* experiments. Coordinated the data aggregation across all four methods to generate comparative convergence plots. Authored the *Experiments & Analysis* (Section 5) and *Conclusion*, providing the final synthesis of efficiency and performance.