Cost-Bounded, Accuracy-Aware Hyperparameter Optimization Integrated with Rule-Based Book Recommendation for Developer Profiling on Stack Overflow

Fatma Altınsoy[1] and Muhammed Maruf Öztürk[2]
[1]Suleyman Demirel University, Department of Information Technologies, Isparta, Turkey
[2]Suleyman Demirel University, Department of Computer Engineering, Isparta, Turkey
E-mail: fatmaaltinsoy@sdu.edu.tr, muhammedozturk@sdu.edu.tr

Keywords: Hyperparameter optimization, AutoML, book recommendation, developer profiling, Stack Overflow, personalized learning

Received: August 28, 2025

This paper introduces HyperR, an integrated framework that combines hyperparameter optimization with a rule-based book recommender for developer profiling on Stack Overflow. The proposed optimization method employs a cost-bounded, accuracy-aware strategy with early stopping and adaptive search to balance computational efficiency and search effectiveness. The framework evaluates both model-free (Grid Search, Random Search, Nelder–Mead, DEoptim) and model-based (Bayesian Optimization, AutoFT, mlrMBO, TuRBO) optimizers across multiple classifiers on a dataset of 20,000 Stack Overflow questions, using stratified 70/30 train-test splits with 20 repeated runs per configuration. Experimental results show that the proposed method achieves the highest macro-averaged precision (0.671) and significantly outperforms classical baselines (adjusted $p < 0.008$), while remaining competitive with state-of-the-art model-based approaches (adjusted $p > 0.05$). In addition, it reduces peak memory usage by up to 25% compared with exhaustive search methods. The recommendation module achieves strong performance (Precision@3 = 0.82 and Recall@3 = 0.77), demonstrating the effectiveness of the proposed framework in delivering personalized learning resources.

Povzetek: Članek predstavi HyperR, okvir za profiliranje razvijalcev na Stack Overflow, ki združi stroškovno omejeno optimizacijo hiperparametrov z zgodnjim ustavljanjem in adaptivnim iskanjem ter pravilniški priporočilnik knjig za učinkovitejše personalizirano učenje.

1 Introduction

Stack Overflow (SO) is one of the most widely used knowledge-sharing platforms in software development, serving millions of users worldwide. Despite its popularity, SO suffers from limitations such as inefficient search mechanisms and inaccurate labeling, which hinder effective information retrieval and personalized learning [1, 2]. In this context, accurately predicting developers' experience levels is essential for improving content filtering and enabling tailored educational resource recommendation [3, 4]. Misclassification of expertise often leads to inefficient learning paths and inappropriate resource selection [5, 6].

Early approaches to developer profiling relied on rule-based techniques, which are limited in capturing complex semantic patterns in high-dimensional data [7]. Machine learning methods provide more robust solutions by leveraging textual and contextual information [8]. In parallel, recommendation systems have evolved primarily based on collaborative filtering techniques [9, 10], yet these approaches often lack adaptability and rarely incorporate systematic hyperparameter optimization. Meanwhile, prior research in HPO has demonstrated that suboptimal parameter configurations can significantly degrade model performance [11, 12], but these studies typically focus on predictive accuracy without integrating downstream recommendation tasks.

Therefore, a critical gap exists between hyperparameter optimization and personalized recommendation systems. Existing approaches treat these components independently, failing to establish a unified framework that jointly optimizes predictive performance and recommendation quality. Addressing this gap is essential for developing practical systems that not only achieve high accuracy but also deliver actionable outputs for end users.

To this end, this study proposes *HyperR*, an integrated framework that combines hyperparameter optimization with a rule-based book recommendation module for developer profiling on SO data. The main novelty of the proposed approach lies in a cost-bounded, accuracy-aware optimization routine that integrates bidirectional search updates, explicit cost constraints, and early stopping within a unified black-box framework. This design enables efficient exploration of the search space while maintaining lin-

ear computational complexity.

The main contributions of this study are summarized as follows:

- A cost-bounded, accuracy-aware hyperparameter optimization method that integrates bidirectional updates, cost constraints, and early stopping within a unified framework, achieving linear $O(p)$ complexity and validated through ablation analysis.
- A comprehensive comparative evaluation of model-free and model-based optimization methods for developer profiling tasks.
- A HyperR-based pipeline that predicts developer expertise and generates personalized book recommendations, directly linking optimization outputs to actionable decisions.
- An extensive evaluation across five classifiers using multiple performance metrics, including Accuracy, F1-score, Precision@K, Recall@K, runtime, and memory usage.

The remainder of this paper is structured as follows. Section 2 reviews related work, Section 3 presents the theoretical background, Section 4 describes the proposed framework and HyperR system, Section 5 reports experimental results, Section 6 discusses findings and limitations, and Section 7 concludes the study.

2 Literature review

SO plays a critical role in software development communities by facilitating knowledge sharing and improving development efficiency [13, 14]. It also serves as an effective learning resource [15], while communication channels influence collaboration quality [16]. Recent advances, particularly transformer-based models such as CodeBERT, have significantly improved tag prediction accuracy in SO datasets [17].

Despite these advances, SO still suffers from noisy labels, unanswered questions, and inefficient search mechanisms. Prior studies have explored text mining [18], deep learning-based error detection [19], and topic modeling [20], but these approaches rarely incorporate systematic hyperparameter optimization (HPO).

HPO is essential for improving machine learning performance [11]. Model-free approaches such as Grid Search and Random Search are widely used but suffer from scalability and non-adaptive search limitations, while methods such as Nelder–Mead [21] and evolutionary approaches such as DEoptim [22] face challenges in high-dimensional settings [23, 24, 25]. Model-based approaches, including Bayesian Optimization [12], TuRBO [26], mlrMBO [27], and AutoFT [28], improve efficiency through surrogate modeling, although they often introduce additional computational complexity.

Recent multi-fidelity strategies further accelerate HPO. Hyperband [29] leverages early stopping for efficient resource allocation, PriorBand [30] incorporates expert priors, and LLAMBO [31] integrates large language models into the optimization loop. However, these approaches are designed for general-purpose optimization and are not tailored to domain-specific recommendation tasks.

In parallel, recommendation systems have evolved from collaborative filtering to hybrid, context-aware, and deep learning-based approaches [32, 33]. Nevertheless, these systems typically focus on user-item interactions and rarely incorporate HPO or consider developer expertise levels. Existing efforts combining optimization and recommendation remain limited and often provide indirect outputs without personalized user-level recommendations [34, 35].

AutoML platforms such as Auto-sklearn [36], AutoGluon [37], FLAML [38], HyperOpt [39], and Optuna [40] have simplified model selection and tuning. Recent frameworks, including FairerML [41], DebiAI [42] and LAMDA-SSL [43] address domain-specific challenges. However, these tools are not designed to integrate hyperparameter optimization with personalized recommendation tasks.

Our previous work [44] introduced a web-based HPO platform for SO datasets but did not include recent model-based optimizers, statistical validation, or a recommendation module.

Table 1 summarizes the most relevant studies and highlights three key gaps: HPO has not been systematically applied to developer profiling on SO, recommendation systems rarely integrate HPO as a core component, and no prior work combines both model-free and model-based optimization with personalized recommendation.

To address these gaps, this study proposes HyperR, which integrates cost-aware hyperparameter optimization with a rule-based book recommendation module, providing a unified and scalable solution for developer-oriented learning systems.

3 Background

Hyperparameter optimization is a critical process in machine learning that directly affects model accuracy, generalization, and computational efficiency—especially in high-dimensional search spaces where effective optimization strategies are essential [45, 46]. This section presents the optimization methods evaluated in this study, followed by the proposed approach.

3.1 Model-free optimization methods

Model-free methods explore the hyperparameter space directly without surrogate models.

Grid Search discretizes each dimension of Λ into k_j values, yielding $|\Omega| = \prod_{j=1}^{d} k_j$ candidate configurations with

Table 1: Comparative analysis of state-of-the-art hyperparameter optimization, AutoML, and recommendation approaches

Study	Approach	Domain Dataset	HPO Method	Rec.	Cost	Early Stop	Eval. Metrics	Limitation
Bergstra & Bengio [11]	Random vs Grid Search	Synthetic, MNIST	Random Search	No	No	No	Accuracy, runtime	Non-adaptive search
Snoek et al. [12]	Bayesian Opt. (GP)	MNIST, CIFAR, LDA	GP + Expected Improvement	No	No	No	Validation error, runtime	High cost $O(t^3)$
Mantovani et al. [34]	Meta-learning tuning	OpenML (156 datasets)	Random Search + Nested CV	Yes	No	No	BAC, AUC	Limited to SVM; no rec.
Yang et al. [35]	AutoML via collab. filtering	OpenML + UCI	Matrix fact. + exp. design	Indirect	Yes	Yes	Error rate, regret, runtime	Model selection only
Feurer et al. [36]	Auto-sklearn	OpenML	SMAC + meta-learning	No	Yes	Yes	Accuracy, F1	General-purpose; no rec.
Eriksson et al. [26]	TuRBO	High-dim. benchmarks	Local GP + Thompson Sampling	No	Yes	Yes	Regret, convergence	Complex; no rec.
Mallik et al. [30]	PriorBand	HPOBench, LCBench	Hyperband + prior sampling	No	Yes	Yes	Regret, error rate, rank	Requires expert prior
Liu et al. [31]	LLAMBO	Bayesmark, HPOBench	BO + LLM warm-start	No	No	Yes	Regret, NRMSE, R^2	High cost; no rec.
He et al. [17]	PTM4Tag	Stack Overflow	—	Yes	No	No	P@k, R@k, F1@k	No HPO
This Study	HyperR	Stack Overflow (20K)	Multi-method + Proposed	Yes	Yes	Yes	Accuracy, F1, Precision, Recall, Specificity, P@3, R@3, runtime, memory	Rule-based recommendation; single domain

Rec. = Recommendation; Cost = Cost-aware; Eval. = Evaluation; rec. = recommendation; P@k = Precision@k; R@k = Recall@k

$O(k^d)$ complexity [47, 48]. Despite its exhaustive nature, it wastes evaluations on unpromising regions.

Random Search samples p configurations uniformly from Λ:

$$\{\lambda^{(1)}, \lambda^{(2)}, \ldots, \lambda^{(p)}\} \sim U(\Lambda) \tag{1}$$

With $O(p)$ complexity independent of d, it outperforms Grid Search in high-dimensional problems by allocating more trials to important hyperparameters [11].

Nelder–Mead [21] is a derivative-free simplex-based method that updates $d+1$ points via reflection, expansion, contraction, or shrinkage:

$$\lambda_{t+1} = \lambda_t + \alpha(\lambda_t - \lambda_{t-1}) \tag{2}$$

It is effective for small-scale problems but lacks global convergence guarantees in high-dimensional landscapes.

DEoptim [22] is a population-based stochastic algorithm that updates candidate vectors via mutation and crossover:

$$v_i^{(t)} = \lambda_{r1}^{(t)} + F \cdot \left(\lambda_{r2}^{(t)} - \lambda_{r3}^{(t)}\right) \tag{3}$$

$$u_{i,j}^{(t)} = \begin{cases} v_{i,j}^{(t)} & \text{if } \mathrm{rand}_j \le CR \text{ or } j = j_{\mathrm{rand}}, \\ \lambda_{i,j}^{(t)} & \text{otherwise.} \end{cases} \tag{4}$$

where F is the mutation factor, CR is the crossover probability, and $r1, r2, r3$ are distinct random indices.

3.2 Model-based optimization methods

Model-based methods construct surrogate models to guide exploration, reducing the number of evaluations.

Bayesian Optimization approximates the objective via a surrogate $\hat{J}(\lambda)$, typically Gaussian Processes [12] or Tree-structured Parzen Estimators [49], selecting the next candidate by maximizing an acquisition function:

$$\lambda_{t+1} = \arg\max_{\lambda \in \Lambda} \alpha(\lambda \mid D_{1:t}) \tag{5}$$

It achieves high sample efficiency but incurs $O(t^3)$ complexity for Gaussian Processes.

AutoFT [28] formulates HPO as a bi-level problem where the outer loop updates λ and the inner loop learns a robust objective L_{meta}:

$$\lambda_{t+1} = \lambda_t - \eta \nabla_\lambda L_{\text{meta}}(f_{\lambda_t}) \tag{6}$$

mlrMBO [27] extends Bayesian Optimization for multi-objective cases using weighted acquisition functions:

$$\lambda_{t+1} = \arg\max_{\lambda \in \Lambda} \sum_{m=1}^{M} w_m \, \alpha_m(\lambda \mid D_{1:t}) \tag{7}$$

It supports mixed discrete–continuous spaces with efficient surrogate modeling.

TuRBO [26] restricts the search to a local trust region $T_t \subset \Lambda$ with dynamically adapting radius:

$$\lambda_{t+1} = \arg\max_{\lambda \in T_t} \alpha(\lambda \mid D_{1:t}) \tag{8}$$

$$R_{t+1} = \begin{cases} \min(\gamma_{\text{inc}} R_t, R_{\max}) & \text{if improvement observed,} \\ \max(\gamma_{\text{dec}} R_t, R_{\min}) & \text{otherwise.} \end{cases} \tag{9}$$

3.3 Proposed hyperparameter optimization method

The proposed method is a cost-bounded and accuracy-aware framework that integrates execution cost limits and

accuracy-threshold–based early stopping to balance search completeness and computational efficiency. Let p be the total number of iterations and Λ the hyperparameter search space. An accuracy threshold z terminates the optimization early if:

$$\mathrm{Acc}(\lambda_t) \geq z \tag{10}$$

Theorem 1 (Execution cost bound). The execution cost per tuning step satisfies $t \geq p/5$.

Proof. Let the total execution cost be $T(p) = \sum_{i=1}^{p} t_i$ and define the reduced stopping point as $\zeta = p/\alpha$ with $\alpha \geq 5$. For feasibility, we require $t_i \geq T(p)/p$. Substituting $\alpha = 5$ yields the bound $t \geq p/5$. Empirical analysis confirms that $\alpha < 5$ leads to under-exploration ($\leq 10\%$ accuracy gain), whereas $\alpha > 5$ increases computation without substantial improvement.

Theorem 2 (Independence of accuracy threshold from iteration count). The accuracy threshold z does not exhibit a linear relationship with the iteration count p:

$$z \neq p \cdot x + c \tag{11}$$

Proof. Empirically, reducing p by a factor d may increase accuracy z, i.e., $z(p-d) > z(p)$. Differentiating yields $\partial z/\partial p \neq x$, confirming that z depends on the optimization dynamics at each step rather than on p linearly.

The proposed method achieves $O(p)$ complexity through the accuracy threshold z, which terminates the process when satisfactory performance is reached. The overall cost is:

$$C(p) = \sum_{i=1}^{p} t_i \cdot I(\mathrm{Accuracy}_i < z) \tag{12}$$

where I is an indicator function. The practical implementation, including the step size definition and bidirectional update mechanism, is presented in Algorithm 1.

4 Method

4.1 Overview

The proposed framework consists of four main phases, as illustrated in Figure 1:

1. **Labeling and Data Preparation:** The SO dataset is preprocessed using standard text mining techniques, and labels are generated for programming language and developer experience.
2. **Hyperparameter Optimization:** Multiple optimization methods are applied to machine learning models (SVM, RF, GBM, EPS, and EBR) to identify optimal hyperparameter configurations.
3. **Prediction and Recommendation:** Optimized models are used to predict developer experience levels and generate corresponding book recommendations.
4. **Web Tool Integration:** The HyperR platform integrates the optimization and recommendation processes into a unified and user-friendly system.

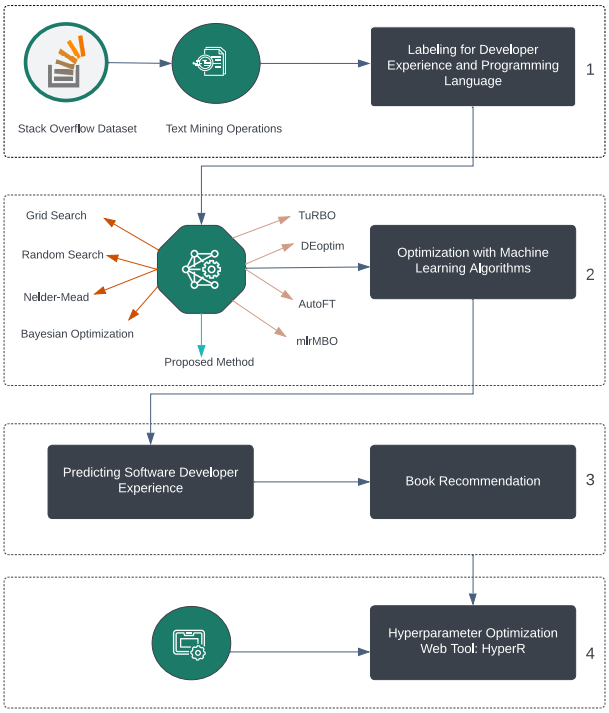


Figure 1: Overview of the proposed method

4.2 Research questions and evaluation criteria

To evaluate the proposed HyperR framework, the following research questions are defined:

- **RQ1:** How does the proposed hyperparameter optimization method improve classification performance compared to baseline approaches?
- **RQ2:** What is the computational cost of the proposed method in terms of runtime and memory usage?
- **RQ3:** How effectively does the system generate accurate and relevant book recommendations based on predicted developer profiles?
- **RQ4:** Are the performance improvements achieved by the proposed method statistically significant compared to baseline approaches?

To answer these questions, the framework is evaluated by comparing its performance against baseline methods using classification metrics including accuracy, precision, recall, specificity, and F1-score, along with computational efficiency based on runtime and memory usage and recommendation quality assessed using Precision@3 and Recall@3.

4.3 Dataset

This study employs the publicly available SO corpus, which provides a large-scale repository of question–answer data for research purposes [50]. The full dataset contains more

than 1,048,575 instances, each including key attributes such as ID, user ID, creation date, closure date, score, title, and body. In addition, tag data represent user-assigned labels indicating programming languages or relevant technical topics, enabling structured categorization of the corpus.

From this large dataset, a labeled subset of 20,000 instances was first constructed, where programming language and developer experience were assigned using text mining and rule-based labeling techniques. To ensure computational feasibility during hyperparameter optimization, a balanced subset of 3,000 instances was selected from this labeled data. This design choice was motivated by the experimental setup, which involves nine optimization methods applied to five classifiers, each repeated 20 times with up to 50 iterations per run, resulting in thousands of model evaluations. Applying this procedure to the full 20,000-instance dataset would introduce substantial computational overhead, particularly for classifiers such as SVM and GBM, which exhibit superlinear scaling with dataset size.

The 3,000-instance subset was constructed using stratified sampling to preserve the original class distribution across more than twenty programming languages and multiple developer experience levels. This ensures that the subset remains representative of the larger labeled dataset in terms of both label diversity and class balance.

To assess whether the results generalize beyond the selected subset, additional experiments were conducted on datasets of varying sizes (500, 1,000, and 3,000 instances). The results demonstrated consistent performance trends across all sizes, with less than 5% variation in macro-averaged precision. The proposed method maintained stable performance across all subsets, while some baseline methods (e.g., Nelder–Mead) exhibited greater sensitivity to dataset size. These findings provide strong evidence that the observed performance differences are primarily driven by the optimization strategies rather than dataset-specific characteristics, supporting the generalizability of the results.

4.4 Experimental protocol

To ensure reproducibility and transparency, all experiments were conducted under a standardized protocol.

The dataset, consisting of 3000 labeled instances, was divided into training and test sets using a stratified split, with 70% allocated for training and 30% for testing. Hyperparameter optimization was performed on the training set, and model performance was evaluated on the test set. No separate validation set was used in order to maintain a consistent and computationally efficient evaluation framework across all optimization methods.

A fixed random seed (`seed`=42) was used throughout all experiments to ensure consistency.

Hyperparameter optimization was carried out using multiple optimizers, including Grid Search, Random Search, Bayesian Optimization, Nelder–Mead, TuRBO, DEoptim, AutoFT, mlrMBO, and the proposed method. For each optimizer, the search space was defined according to the parameter ranges and step sizes provided in Table 3. The iteration budget p and stopping threshold z were kept identical across all methods.

All optimization methods were executed under the same computational budget to ensure a fair comparison. Each method was allowed the same number of iterations, identical search space definitions, and equivalent stopping criteria. No method was granted additional evaluations or extended runtime beyond the predefined limits.

Each experimental configuration was repeated 20 times, and the reported results correspond to the average performance across runs. Evaluation metrics include accuracy, precision, recall, specificity, and F1-score for classification tasks, as well as Precision@3 and Recall@3 for recommendation performance. Computational efficiency was assessed using runtime and peak memory consumption.

All experiments were implemented using R (version 4.2.1) and Python (version 3.13). The primary R packages include `caret`, `e1071`, `randomForest`, and `gbm`.

The experiments were conducted on a system equipped with an Intel Core i7-14700 CPU (2.10 GHz) and 32 GB RAM, running a 64-bit Windows Server 2019 operating system.

The dataset, preprocessing procedures, hyperparameter configurations, and full implementation are publicly available in the project repository, enabling exact reproduction of the reported results.

4.5 Text mining operations

To prepare the dataset for machine learning tasks, a systematic text mining pipeline was applied to the question titles and bodies from the SO corpus. The preprocessing workflow consisted of tokenization [51], lowercase normalization, noise removal (punctuation, digits, hyperlinks) [52], stopword removal, and stemming with lemmatization [53].

After preprocessing, feature extraction was performed using the Term Frequency–Inverse Document Frequency (TF–IDF) weighting scheme [54], where the weight of a term t in document d is computed as:

$$w_{t,d} = TF(t,d) \times IDF(t,D) \tag{13}$$

with $TF(t,d) = f_{t,d} / \sum_{t' \in d} f_{t',d}$ and $IDF(t,D) = |D|/(1 + |\{d \in D : t \in d\}|)$. This ensures that terms frequent in a specific document but rare across the corpus receive higher discriminative importance [55]. In addition to unigrams, bigram features were extracted to capture short-range contextual dependencies [56]. The analysis was conducted directly on the TF–IDF matrix without applying dimensionality reduction [57].

4.6 Labeling process

The dataset was gradually expanded to approximately 20,000 entries, with each configuration repeated at least 20 times for statistical reliability.

Table 2: An example of the labeling matrix generated from the raw question dataset

y_1	y_2	y_3	y_4	y_5	y_6	y_7	y_8	y_9	y_{10}	y_{11}	y_{12}	y_{13}	y_{14}	y_{15}	y_{16}	y_{17}	y_{18}	y_{19}	y_{20}	y_{21}	y_{22}	y_{23}
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	1	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	2	22
0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	1	0	0	1	1	0	1	41
0	0	0	0	1	1	0	0	0	0	0	0	0	0	0	0	0	0	1	1	0	1	51
0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	3	43
0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	21
0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0
0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	1	51

Each SO question $q_i = (\text{title}_i, \text{body}_i)$ was cleaned and transformed into a vector $q_i = (t_{i1}, \ldots, t_{im})$ with additional features (character count c_i, word count w_i), yielding an enriched representation $\tilde{x}_i = (x_i, c_i, w_i)$.

The label vector for each instance was defined as $y_i = (y_{i1}, \ldots, y_{i23})$. The first 21 dimensions encode programming language tags ($y_{ij} = 1$ if $q_i \in L_j$, 0 otherwise), covering 21 languages selected based on their consistent SO usage over 14 years [58]:

$$L = \left\{ \begin{array}{l} \text{JavaScript, SQL, Java, C\#, Python,} \\ \text{C++,C, PHP, Ruby, Swift,Objective-C,} \\ \text{VB.NET,Perl,Bash, CSS, Scala, HTML,} \\ \text{Lua,Haskell,Markdown, R} \end{array} \right\}$$

The 22nd label (y_{i22}) represents user experience level: 0 (unspecified), 1 (beginner), 2 (intermediate), or 3 (advanced). Experience labels were assigned using a rule-based keyword mechanism derived from domain knowledge and empirical analysis of SO data. Basic constructs (e.g., variable, array, function) indicate beginner level; terms such as threads, JSON, and encryption characterize intermediate level; and system-level concepts (e.g., REST, docker, kubernetes) are linked to advanced expertise. The dominant keyword group determines the final label. Complete keyword lists are available in the project repository.

The 23rd label (y_{i23}) corresponds to book recommendations, defined as:

$$y_{i23} = f(L_j, y_{i22}), \quad f : L \times \{0, 1, 2, 3\} \to \mathbb{N} \tag{14}$$

where f is a deterministic lookup mapping each language–experience pair to a predefined set of book identifiers.

To refine recommendations, Amazon ratings (r_{avg}) were incorporated [59]:

$$\hat{y}_{i23} = \arg\max_{b \in B} \left(f(L_j, y_{i22}) + \alpha \cdot r_{\text{avg}}(b) \right) \tag{15}$$

where B is the candidate book set and $\alpha = 0.2$ ensures that the rule-based mapping remains the primary driver while incorporating external ratings in a complementary manner.

The structured labeling process results in a multi-label encoding where each instance is represented as $(q_i, \tilde{x}_i, y_i)$. Table 2 illustrates this mapping.

4.7 The proposed algorithm

Algorithm 1 Heuristic hyperparameter optimization algorithm

Input: dataset, evaluation metric $\varphi(\cdot)$, bounds $[y, x]$, p, v, z
Output: maxAccuracy, λ^*

1: optimizationValue $\leftarrow \{y, y+v, y+2v, \ldots, x\} \subseteq \Lambda$
2: $\lambda(1) \leftarrow$ optimizationValue[1]
3: accuracy $\leftarrow \varphi(\lambda(1))$
4: accuracyOld $\leftarrow$ accuracy
5: maxAccuracy $\leftarrow$ accuracy
6: $\lambda^* \leftarrow \lambda(1)$
7: $\varepsilon \leftarrow 10^{-3}$
8: $i \leftarrow 1$
9: **while** (accuracy $< z$) **AND** ($i < p$) **do**
10: **if** (accuracy $\geq$ accuracyOld + ε) **then**
11: $\lambda(i+1) \leftarrow \lambda(i) + v$
12: **else**
13: $\lambda(i+1) \leftarrow \lambda(i) - v$
14: **end if**
15: **if** $\lambda(i+1) < y$ **then** $\lambda(i+1) \leftarrow y$
16: **if** $\lambda(i+1) > x$ **then** $\lambda(i+1) \leftarrow x$
17: accuracyOld $\leftarrow$ accuracy
18: accuracy $\leftarrow \varphi(\lambda(i+1))$
19: **if** accuracy $\geq$ maxAccuracy **then**
20: maxAccuracy $\leftarrow$ accuracy
21: $\lambda^* \leftarrow \lambda(i+1)$
22: **end if**
23: $i \leftarrow i + 1$
24: **end while**
25: **return** $(\lambda^*, \text{maxAccuracy})$

The proposed algorithm is presented in Algorithm 1 and implements the theoretical framework described in the Background section by defining a cost-bounded and accuracy-aware procedure for hyperparameter optimization.

In the initialization phase (Steps 1–8), the search interval $[y, x]$ is discretized using step size v, producing candidate configurations $\{y, y+v, y+2v, \ldots, x\} \subseteq \Lambda$. The first element λ_1 is evaluated, and its accuracy $\varphi(\lambda_1)$ initializes both the current and best accuracy values. An improvement

tolerance $\varepsilon = 10^{-3}$ is set to avoid sensitivity to noise.

The main loop (Step 9) continues while accuracy remains below threshold z and iteration count has not exceeded budget p. The core mechanism is the bidirectional update (Steps 10–14): if accuracy improves beyond ε, the search continues in the same direction ($+v$); otherwise, the direction reverses ($-v$). Boundary checks (Steps 15–16) ensure all candidates remain within $[y, x]$. The best configuration λ^* is retained throughout (Steps 19–22).

This bidirectional update distinguishes the proposed method from traditional line search (single direction) and coordinate descent (one parameter at a time). By integrating tolerance-based direction switching, cost-bounding constraints, and accuracy-threshold early stopping, the algorithm provides a robust and efficient solution that scales linearly with the iteration budget.

4.8 Web tool: HyperR

All experiments were executed through HyperR, a web-based platform that centralizes dataset selection, algorithm specification, optimizer choice, and resource monitoring, ensuring methodological consistency and reproducibility across runs.

The data flow, illustrated in Figure 2, consists of two interconnected pipelines. In the upper pipeline, a user-submitted SO question (title and body) is processed through text processing (tokenization, stopword removal, stemming, lemmatization) and feature extraction (TF–IDF vectorization with unigrams, bigrams, word count, and character count). The resulting features are passed to the model prediction module, which performs two tasks: predicting the programming language (y_1–y_{21}) and predicting the developer experience level (y_{22}). Based on these predictions, the recommendation module applies the rule-based mapping combined with external ratings to generate top-3 book recommendations.

The lower pipeline represents the hyperparameter optimization loop. Starting from a predefined search space (bounds, step size v, threshold z, budget p), the optimization process iteratively evaluates candidate configurations across classifiers (SVM, RF, GBM, EPS, EBR) and computes $\varphi(\lambda)$. The best model selection stage identifies the optimal configuration based on maximum accuracy, while logging runtime and memory usage. The selected model is then fed into the upper pipeline for inference.

The system is implemented on `ASP.NET Core MVC` with a layered architecture. The backend dispatches jobs to R scripts (caret [60], e1071 [61], gbm [62], randomForest [65], rBayesianOptimization [66]) via the .NET `System.Diagnostics.Process` class. Each experiment is assigned a unique Experiment ID, generated as a hash of dataset, optimizer, and timestamp, guaranteeing traceability. The R backend computes $\varphi(\lambda)$ for each candidate configuration and logs execution time and memory usage before returning results to the web tier.

The system interface and outputs are illustrated in Figure 3. Panel (a) presents detailed tabular optimization results, including accuracy values across iterations, the best hyperparameter configuration, execution time in seconds, and memory usage in megabytes. Panel (b) visualizes performance trends, where the x-axis represents hyperparameter values and the y-axis denotes accuracy, along with summary statistics such as mean and standard deviation. Across repeated runs, these are computed as:

$$\mathrm{MeanAcc} \;=\; \frac{1}{p}\sum_{i=1}^{p}\varphi(\lambda_i) \tag{16}$$

$$\mathrm{SD} \;=\; \sqrt{\frac{1}{p}\sum_{i=1}^{p}\big(\varphi(\lambda_i)-\mathrm{MeanAcc}\big)^2} \tag{17}$$

Once optimization converges, the best model is stored under the corresponding Experiment ID. When a developer submits a query, HyperR applies the optimized model to predict both the programming language and experience level, then dynamically recommends books matched to the predicted profile. The recommendation interface is depicted in Figure 4(a). As illustrated, the system analyzes a user-submitted query, predicts the associated programming language and experience level, and generates a tailored set of book recommendations. In the example shown, the system identifies the query as Python-related with a beginner-level profile and provides three introductory books aligned with this prediction.

The code execution interface (Figure 4b) enables users to submit R scripts directly within the web-based environment and observe outputs in real time, with input and results displayed side by side for transparency and reproducibility. All results are stored in CSV format and can be exported in multiple formats (PNG, JPG, PDF) to support reporting.

4.9 Machine learning algorithms

Hyperparameter optimization is essential for selecting the best parameter configuration in machine learning models, as it significantly improves overall predictive performance. Manual tuning requires testing numerous combinations, which is computationally expensive and inefficient. To address this challenge, automated search techniques are employed. In this study, multiple algorithms with optimized hyperparameters were evaluated under both single-label and multi-label classification settings. In single-label tasks, only the developer's experience level (y_{22}) was predicted, while in multi-label tasks, programming language labels (y_1–y_{21}) were predicted simultaneously with y_{22}.

SVM are widely used for classification and rely on hyperparameters such as kernel type, cost, degree, and gamma, which directly affect model complexity and generalization [47]. Different kernels, including linear, polynomial, sigmoid, and Radial Basis Function (RBF), can yield substantially different performance outcomes.

RF is an ensemble learning method for classification and regression that combine multiple decision trees. Its key hy-

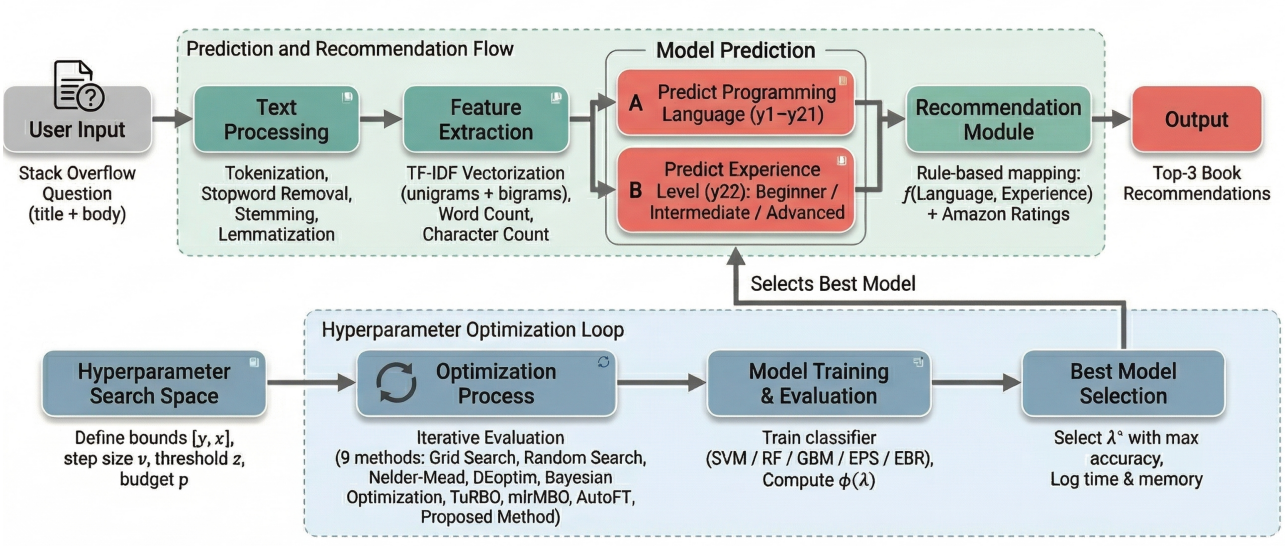


Figure 2: HyperR workflow and data flow architecture showing the prediction/recommendation pipeline (top) and the hyperparameter optimization loop (bottom)

Table 3: Hyperparameters employed in the experiments

Algorithm	Parameter	Description	Range	Step	Type	Target
SVM	kernel	Kernel type	–	–	Single	y_{22}
	degree	Polynomial exponent	1–10	1		
	gamma	Kernel coefficient	2^{-10}–2^{10}	0.1		
	cost	Error penalty	2^{-10}–2^{10}	1		
RF	mtry	Features per split	1–8	1	Single	y_{22}
	ntree	Number of trees	100–500	1		
GBM	shrinkage	Learning rate	0.001–1	0.1	Single	y_{22}
	n.trees	Number of trees	100–500	1		
	n.minobsinnode	Min. node samples	5–15	1		
EPS	subsample	Instance rate	0.1–1	0.1	Multi	y_1–y_{22}
	strategy	Cluster strategy	A–B	–		
	b	Strategy parameter	1–4	1		
	p	Pruned instances	1–4	1		
EBR	subsample	Instance rate	0.1–1	0.1	Multi	y_1–y_{22}
	attr.space	Feature rate	0.1–1	1		

perparameters include `mtry` (the number of features considered at each split) and `ntree` (the number of trees in the forest). Increasing the number of trees typically enhances information retention but also raises computational cost [67].

GBM build predictive models by iteratively combining weak learners to minimize residual errors. Key hyperparameters include `shrinkage` (learning rate), `n.trees` (number of boosting iterations), and `n.minobsinnode` (minimum observations per terminal node). These parameters balance overfitting risk and robustness against noise [62].

For multi-label classification, two algorithms were employed: EPS and EBR. EPS is designed to handle sparse label distributions by pruning less frequent label combinations and optimizing clustering strategies [63, 64]. EBR, on the other hand, decomposes the multi-label task into independent binary classification problems for each label and aggregates the results. Both methods rely on hyperparameters such as subsample rate, strategy, b, p (for EPS), and `attr.space` (for EBR), which were tuned to improve classification accuracy.

Each algorithm was selected based on its ability to handle complex datasets and improve prediction accuracy. Hyperparameters were carefully fine-tuned to ensure reliable experimental results. The use of RF and multi-label classifiers such as EPS and EBR allowed for diversified and improved model performance while balancing computational efficiency.

The hyperparameters used in the experiments, along with their descriptions, ranges, and step sizes, are summarized in Table 3. These values were determined based on prior literature to ensure the scientific validity and comparability of the results.

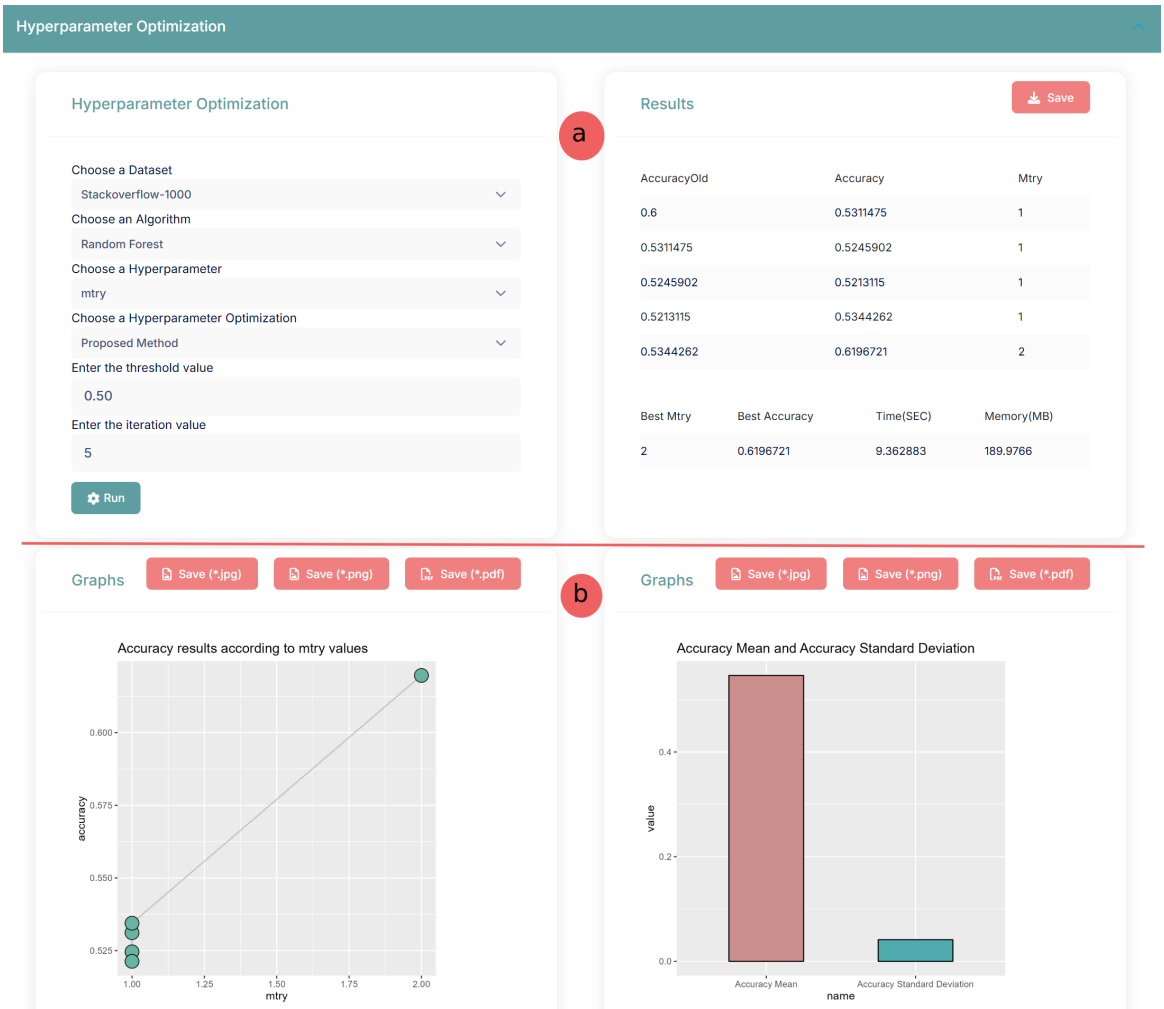


Figure 3: System interface and outputs

5 Results

5.1 RQ1: Classification performance comparison

Table 4 summarizes the macro-averaged classification metrics obtained by each optimization method.

Table 4: Macro-averaged classification metrics by optimization method

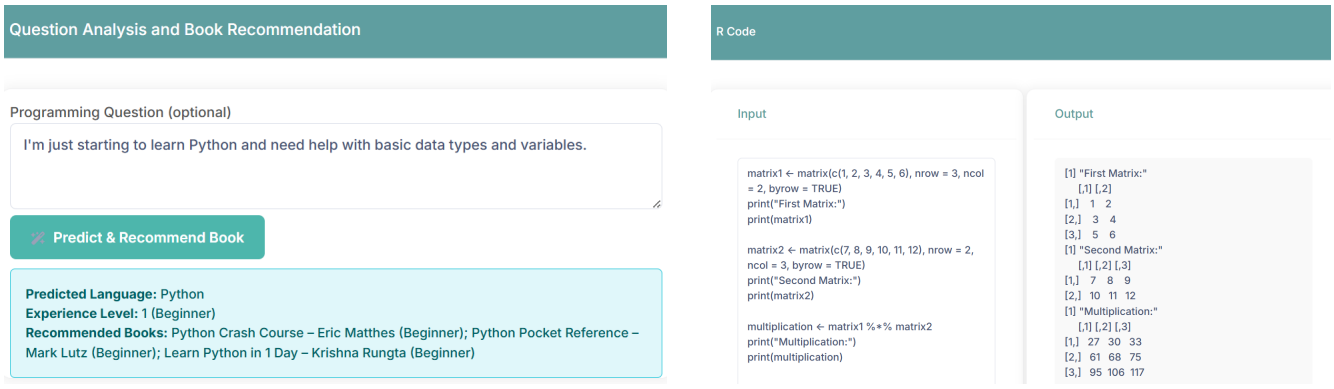
Method	F1	Precision	Recall	Specificity
Grid Search	0.486	0.618	0.472	0.880
Random Search	0.485	0.635	0.486	0.879
Nelder–Mead	0.404	0.525	0.406	0.839
DEoptim	0.508	0.660	0.490	0.885
Bayes Opt.	0.483	0.603	0.478	0.870
AutoFT	0.495	0.620	0.490	0.875
mlrMBO	0.507	0.640	0.492	0.883
TuRBO	0.515	0.645	0.500	0.882
Proposed Method	0.504	0.671	0.479	0.884

Among the classical model-free approaches, Grid Search and Random Search yield moderate results, with Random Search achieving slightly higher recall (0.486 vs. 0.472). Nelder–Mead clearly underperforms, recording the lowest F1-score (0.404) and specificity (0.839). The population-based DEoptim exhibits the strongest precision among model-free methods (0.660) with competitive F1 (0.508) and specificity (0.885).

Model-based methods provide more stable outcomes. TuRBO achieves the highest overall F1-score (0.515), reflecting a balanced precision-recall trade-off, while mlrMBO follows closely with consistent scores across all measures. AutoFT slightly surpasses Bayesian Optimization in recall (0.490 vs. 0.478).

The proposed method stands out with the highest precision (0.671), improving by approximately 13–18% over classical baselines. Although its recall (0.479) does not surpass TuRBO or mlrMBO, its F1-score (0.504) and specificity (0.884) remain competitive, highlighting its strength in minimizing false positives.

Convergence analyses based on iteration-wise accuracy trends (Figures 5 and 6) reveal clear differences between optimization strategies. In Figure 5, the proposed method demonstrates more stable and faster convergence compared



(a) Recommendation interface (b) Code execution interface

Figure 4: User interfaces of the HyperR system

to classical model-free approaches, outperforming Nelder–Mead and maintaining a clear advantage over Grid and Random Search across all classifiers. However, DEoptim may reach comparable accuracy levels in certain cases.

In Figure 6, advanced model-based methods such as TuRBO and mlrMBO exhibit strong convergence behavior. While the proposed method achieves comparable performance, TuRBO occasionally reaches similar accuracy levels in fewer iterations, reflecting its efficient local search capability.

Scalability analysis across dataset sizes (500, 1000, and 3000 instances) provides additional insights. On smaller datasets, performance differences are more pronounced, with the proposed method maintaining stable precision while methods such as Nelder–Mead deteriorate significantly. At medium size (1000), most methods converge toward similar levels, yet the proposed method maintains its precision advantage. On the largest dataset (3000), performance differences narrow, with TuRBO achieving the highest F1-score while the proposed method remains competitive across all metrics.

These results indicate that the proposed method provides a balanced optimization trajectory, combining stability, robustness under limited data, and competitive convergence. In line with state-of-the-art approaches (Table 1), advanced methods such as TuRBO and AutoFT achieve strong performance but at the cost of increased complexity. The proposed method offers competitive accuracy with a simpler and more efficient strategy, highlighting its practical suitability in scenarios where both performance and efficiency are critical.

5.2 RQ2: Computational efficiency analysis

To evaluate the computational cost of different optimization strategies, runtime and memory usage are analyzed across all classifiers.

Figure 7(a) presents the average runtime of the optimization methods, where the x-axis represents the classifiers (EBR, EPS, GBM, RF, and SVM) and the y-axis denotes execution time in seconds. As observed, classical approaches such as Grid Search and DEoptim exhibit the highest computational cost due to their exhaustive and population-based search mechanisms, which require a large number of evaluations. In contrast, model-based methods, including Bayesian Optimization, AutoFT, mlrMBO, and TuRBO, generally achieve lower runtime by guiding the search toward promising regions of the search space.

The proposed method demonstrates a balanced runtime profile across all classifiers. It consistently requires less time than exhaustive approaches while remaining competitive with advanced model-based optimizers. This indicates that the cost-bounded search strategy and early stopping mechanism effectively reduce unnecessary evaluations without sacrificing optimization quality.

Figure 7(b) illustrates the average memory consumption of the optimization methods, where memory usage is measured in megabytes (MB). Similar to runtime behavior, classical methods tend to consume more memory due to extensive exploration of the search space. Model-based approaches show improved memory efficiency by focusing on a smaller set of candidate configurations.

The proposed method maintains a stable and moderate memory footprint across all classifiers, avoiding the extreme values observed in some baseline methods (e.g., TuRBO and DEoptim). This stability indicates that the method provides efficient resource utilization without introducing significant overhead.

From a comparative perspective, model-based approaches such as TuRBO and mlrMBO achieve strong efficiency through surrogate modeling and adaptive sampling strategies. However, these methods often involve higher algorithmic complexity and implementation overhead. In contrast, the proposed method offers competitive computational performance while maintaining a simpler and more interpretable optimization process.

The results demonstrate that the proposed method achieves a favorable trade-off between computational cost and predictive performance, making it suitable for practical applications where both efficiency and scalability are criti-

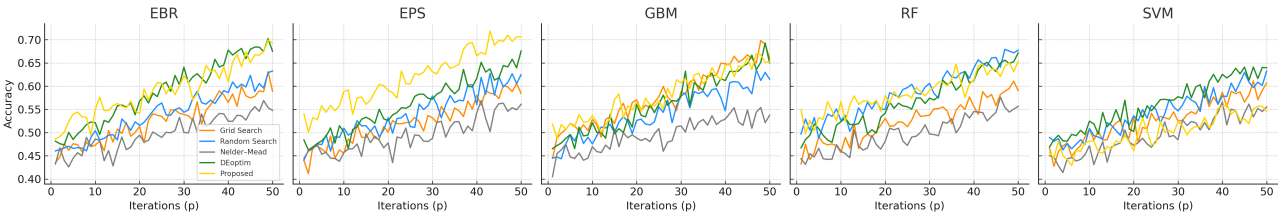


Figure 5: Convergence comparison of the proposed method against classical model-free optimization methods.

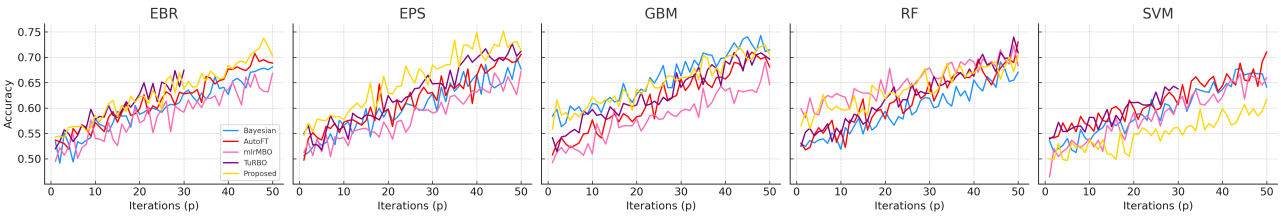


Figure 6: Convergence comparison of the proposed method against advanced model-based optimization methods.

cal.

5.3 RQ3: Recommendation performance evaluation

To evaluate the recommendation module, Precision@k and Recall@k metrics are used with $k = 3$. For each test instance i, the system generates a recommended list R_i of three books, while the ground-truth relevant set G_i is constructed based on the books associated with the corresponding programming language and experience level. Metrics are computed as:

$$\text{Precision@}k = \frac{1}{N}\sum_{i=1}^{N}\frac{|R_i \cap G_i|}{k}, \tag{18}$$

$$\text{Recall@}k = \frac{1}{N}\sum_{i=1}^{N}\frac{|R_i \cap G_i|}{|G_i|}. \tag{19}$$

The recommendation list is constructed by restricting the candidate pool to books matching the predicted programming language and experience level. All evaluation metrics are computed exclusively on the held-out test set, while hyperparameter tuning and early stopping rely only on training data to prevent data leakage.

Under this protocol, the system achieves a Precision@3 of 0.82 and a Recall@3 of 0.77, indicating that the recommended lists are both accurate and highly relevant to the user profile. Consistent with RQ1, optimization methods achieving higher classification accuracy tend to yield better recommendation performance. However, the proposed method does not uniformly outperform all alternatives; in certain cases, optimizers such as TuRBO or DEoptim may achieve comparable results, supporting the interpretation that the proposed method provides a balanced trade-off between predictive performance and computational efficiency.

To further assess sensitivity to variations in query complexity and expertise, a qualitative evaluation was conducted using representative queries across three experience levels and multiple programming languages (Table 5). The results indicate that the system consistently identifies the programming language and assigns appropriate experience levels, producing level-specific book recommendations. Beginner-level queries are associated with introductory resources, intermediate queries yield mid-level references, and advanced queries involving specialized concepts such as REST APIs, stored procedures, and containerization technologies are matched with advanced-level materials.

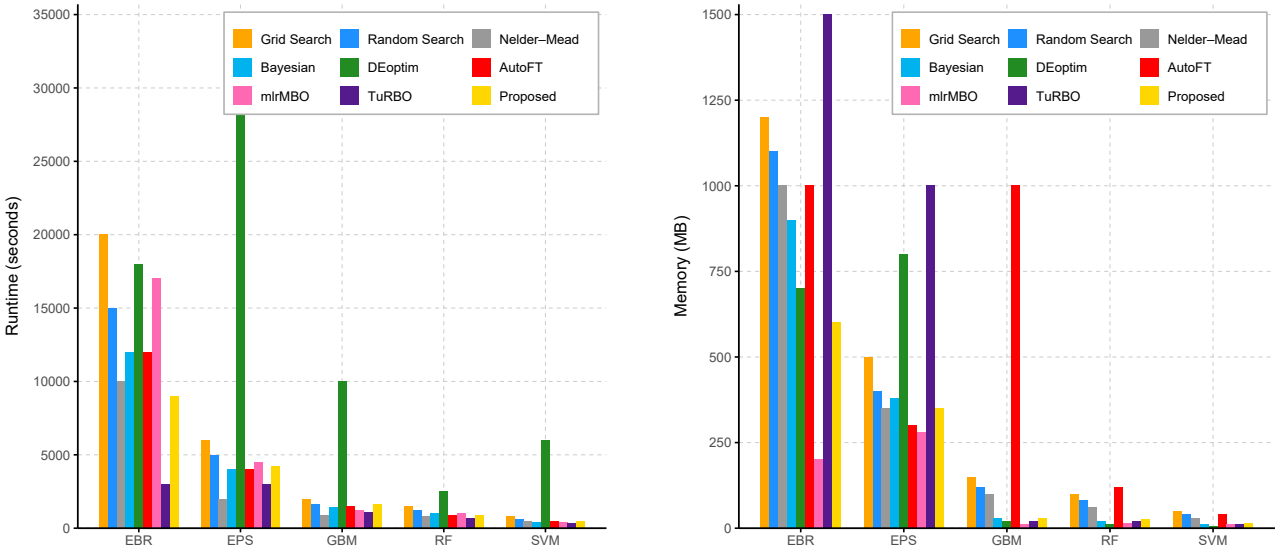
These findings confirm that the recommendation module is responsive to variations in query content and developer expertise. Nevertheless, its effectiveness depends on the coverage of the keyword-based labeling scheme, and performance may degrade for queries involving emerging or ambiguous terminology.

5.4 RQ4: Statistical significance and comparative analysis

To assess whether the observed performance differences are statistically significant, the Wilcoxon signed-rank test is applied to compare the proposed method with alternative optimization strategies in terms of F1-score.

Table 6 summarizes the pairwise Wilcoxon signed-rank test results, reporting the test statistic, original p-values, and Bonferroni-adjusted p-values ($k = 8$, adjusted values capped at 1.0).

The results indicate that the proposed method achieves statistically significant improvements over Grid Search, Random Search, Nelder–Mead, Bayesian Optimization,



(a) Average runtime across optimization methods

(b) Average memory usage across optimization methods

Figure 7: Computational efficiency comparison of optimization methods, where (a) shows average runtime in seconds and (b) shows average memory usage in MB across different classifiers.

and AutoFT (adjusted $p < 0.008$), confirming that the observed gains reflect consistent improvements rather than random variation.

In contrast, no statistically significant differences are observed with DEoptim, mlrMBO, and TuRBO (adjusted $p = 1.000$), suggesting that the proposed method remains competitive with state-of-the-art approaches without requiring the same level of computational complexity.

These findings support the interpretation that the proposed method offers a robust alternative to both classical and advanced strategies, achieving a balanced performance profile that combines competitive accuracy with improved computational efficiency. The observed significance remains consistent after Bonferroni correction across different classifiers and dataset sizes, indicating that the gains are stable and generalizable.

5.5 Ablation study

To further analyze the contribution of individual components of the proposed optimization method, an ablation study was conducted. Specifically, three key mechanisms were examined: the cost-bounding strategy, the accuracy threshold condition, and the bidirectional update rule. Each component was selectively removed to evaluate its impact on model performance.

The analysis was primarily performed on the Random Forest and GBM models, as these provide stable and interpretable environments for assessing optimization behavior. The results are summarized in Table 7.

The results indicate that the proposed optimization framework generally improves performance compared to the baseline configuration, with the most notable gains observed for the RF model. The substantial drop in baseline accuracy highlights the effectiveness of the overall optimization strategy.

For the RF model, removing individual components leads to noticeable performance variations, indicating a high sensitivity to adaptive optimization mechanisms. In contrast, the GBM model exhibits relatively stable behavior, with only minor performance changes when individual components are removed, suggesting that it benefits primarily from the overall optimization structure rather than specific mechanisms.

The SVM model shows a different pattern. Although removing individual components typically reduces performance, the baseline configuration achieves higher accuracy than the proposed method. This suggests that SVM is highly sensitive to specific hyperparameter settings and that the adaptive strategy may occasionally converge to suboptimal regions.

Similar to GBM, the EPS and EBR models demonstrate relatively stable behavior. The impact of removing individual components is limited, and in some cases, simplified configurations yield comparable or slightly improved performance. This indicates that, for ensemble-based models, the overall optimization framework contributes more to performance than any single component.

These findings suggest that the effectiveness of the proposed method is model-dependent. Greater improvements are observed in models that are highly sensitive to hyperparameter tuning, while more stable models benefit from the general optimization structure without relying heavily on individual components.

Table 5: Qualitative sensitivity analysis of the recommendation module across different query types and expertise levels

Query	Language	Experience	Recommended Books
Beginner-level queries			
What is a variable in python?	Python	1	Python Crash Course – E. Matthes; Python Pocket Reference – M. Lutz; Learn Python in 1 Day – K. Rungta
How to create an array in javascript?	JavaScript	1	Learn JavaScript Visually – I. Demirov; JavaScript: The Definitive Guide – D. Flanagan; Beginning JavaScript – P. Wilton
What is a function in java?	Java	1	Beginning Programming with Java For Dummies – B. Burd; Java: Programming Basics for Absolute Beginners – N. Clark; Head First Java – K. Sierra, B. Bates
Intermediate-level queries			
How do I use class in python?	Python	2	Fluent Python – L. Ramalho; Effective Python – B. Slatkin; Programming Python – M. Lutz
How to use threads in java?	Java	2	Effective Java (3rd Ed.) – J. Bloch; Java: The Complete Reference – H. Schildt; Thinking in Java – B. Eckel
How to handle JSON in javascript?	JavaScript	2	Eloquent JavaScript – M. Haverbeke; A Smarter Way to Learn JavaScript – M. Myers; JavaScript & jQuery – J. Duckett
Advanced-level queries			
How to implement REST API with services in python?	Python	3	Python Cookbook – D. Beazley, B.K. Jones; Advanced Python Programming – G. Lanaro et al.; Learn Python 3 The Hard Way – Z.A. Shaw
How to use stored procedures in SQL?	SQL	3	SQL Cookbook – A. Molinaro; SQL For Smarties – J. Celko; The Art of SQL – S. Faroult, P. Robson
How to deploy with docker and kubernetes in java?	Java	3	Well-grounded Java Developer – B. Evans et al.; Java Concurrency in Practice – B. Goetz et al.; Optimizing Java – B.J. Evans et al.

Table 6: Wilcoxon signed-rank test results comparing the proposed method with other optimizers (F1-score). Bonferroni-corrected p-values are reported for multiple comparisons ($k = 8$)

Method	Statistic	p-value	Adj. p-value
Grid Search	3.0	< 0.001	< 0.008
Random Search	0.0	< 0.001	< 0.008
Nelder–Mead	0.0	< 0.001	< 0.008
DEoptim	182.0	0.309	1.000
Bayes Opt.	0.0	< 0.001	< 0.008
AutoFT	36.0	< 0.001	< 0.008
mlrMBO	167.0	0.184	1.000
TuRBO	175.0	0.245	1.000

Table 7: Ablation study evaluating the impact of cost-bounding, accuracy threshold, and bidirectional update components

Model	Scenario	Accuracy	Δ Accuracy
RF	Full Model	0.6378	0.0000
RF	No Cost-Bounding	0.6389	+0.0011
RF	No Threshold	0.6411	+0.0033
RF	No Update	0.6433	+0.0056
RF	Baseline	0.5222	−0.1156
GBM	Full Model	0.6467	0.0000
GBM	No Cost-Bounding	0.6456	−0.0011
GBM	No Threshold	0.6467	0.0000
GBM	No Update	0.6433	−0.0033
GBM	Baseline	0.6133	−0.0333

6 Discussion

The experimental results provide a comprehensive comparison of classical and modern hyperparameter optimization strategies in the context of developer profiling and recommendation. Classical model-free approaches, including Grid Search, Random Search, and Nelder–Mead, exhibit limited scalability and unstable performance across classifiers, with their lack of guided exploration leading to inefficient search and increased computational cost. Among these, DEoptim performs relatively better due to its population-based mechanism, though its slower convergence results in higher runtime.

In contrast, model-based approaches such as mlrMBO, TuRBO, and AutoFT demonstrate more stable convergence behavior. TuRBO achieves the highest F1-score through effective local search, while mlrMBO provides consistent performance across multiple metrics. However, these methods rely on surrogate modeling, which introduces additional computational overhead.

Table 8 provides an analytical comparison of all evaluated methods in terms of performance, efficiency, and stability. Compared to classical methods, the proposed approach achieves higher precision and improved efficiency, and remains competitive with model-based optimizers while avoiding their computational overhead.

The effectiveness of the proposed method can be attributed to three key design mechanisms: (i) a cost-bounded search strategy that limits unnecessary exploration, (ii) an accuracy-threshold-based early stopping criterion that reduces computational overhead, and (iii) a tolerance-driven bidirectional update that enables flexible exploration and helps avoid suboptimal solutions. These mechanisms collectively explain the observed performance, where the proposed method achieves the highest macro-averaged precision (0.671) while maintaining competitive F1-score and specificity.

From a methodological perspective, the novelty lies in integrating bidirectional updates, cost constraints, and early stopping within a unified black-box optimization framework. This design enables an explicit balance between search efficiency and performance without relying on com-

Table 8: Analytical comparison of hyperparameter optimization methods based on experimental observations across performance and efficiency criteria

Method	Precision	F1	Efficiency	Stability	Key Insight
Grid Search	Medium	Medium	Very Low	Low	High computational cost with limited improvement despite exhaustive search.
Random Search	Medium	Medium	Medium	Medium	Provides balanced performance but lacks convergence consistency.
Nelder–Mead	Low	Low	High	Low	Unstable performance and weak convergence across classifiers.
DEoptim	High	Medium–High	Low	Medium	Strong exploration improves precision but increases runtime significantly.
Bayesian Optimization	Medium	Medium	Medium	High	Stable convergence but does not consistently achieve top performance.
AutoFT	Medium–High	Medium–High	Medium	High	Maintains stable optimization across models with moderate cost.
mlrMBO	High	High	Medium	High	Consistently strong performance across all metrics.
TuRBO	High	Highest	Medium–High	High	Best F1-score due to effective local search and fast convergence.
Proposed Method	Highest	High	High	High	Achieves highest precision with efficient search by limiting unnecessary evaluations and early stopping.

putationally expensive surrogate models.

The recommendation module further demonstrates practical value, achieving Precision@3 of 0.82 and Recall@3 of 0.77, indicating that improved classification performance directly enhances recommendation quality.

From a practical perspective, deployment in real-world environments requires consideration of security and privacy issues, as the framework processes user-generated content and predicts developer expertise levels. Future implementations should incorporate privacy-preserving mechanisms, secure data handling procedures, and transparent usage policies to ensure responsible application.

6.1 Limitations and future work

Despite the promising results, several limitations should be acknowledged.

First, the recommendation module relies on a rule-based mapping between programming language, experience level, and predefined book lists. While this ensures interpretability and reproducibility, it limits personalization by not capturing individual user preferences or adapting to dynamic user behavior. In dynamic communities such as SO, where user expertise evolves over time and new technologies emerge frequently, such static mappings may require periodic updates to remain effective. Consequently, users with similar predicted profiles receive identical recommendations. Incorporating adaptive approaches such as neural collaborative filtering or transformer-based ranking models could improve personalization, albeit at the cost of increased computational complexity.

Second, the experimental evaluation is conducted on a single domain (Stack Overflow). Although the dataset is rich and diverse, the generalizability of the framework to other domains remains to be validated through cross-domain experiments.

Third, scalability to very large datasets remains a consideration. The current analysis is based on subset experiments (500, 1,000, and 3,000 instances) drawn from a corpus of over one million posts. While results demonstrate stable performance with limited variation, further evaluation on full-scale datasets and more heterogeneous corpora is necessary. Such scenarios may require distributed computation or incremental learning strategies.

Finally, the labeling process relies on keyword-based heuristics, which may not fully capture nuanced developer expertise levels. In addition, the proposed method does not consistently outperform all state-of-the-art optimizers across every metric, particularly recall and F1-score, although it demonstrates strong performance in precision and computational efficiency.

Future work will focus on (i) integrating adaptive recommendation models, (ii) extending the framework to cross-domain settings, (iii) evaluating scalability on large-scale datasets, and (iv) improving labeling strategies using data-driven approaches.

Acknowledgement

Availability of data and material

The dataset used in this manuscript is publicly available from Kaggle: `https://www.kaggle.com/datasets/stackoverflow/stacksample`.

Code availability

The codes for the experiments are available at: `https://github.com/fatmaaltinsoy/a-heuristic-technique-for-hyperpameter-tuning.git`.

Web interface

The developed web interface, HyperMLopt, can be accessed at: `https://hypermlopt.sdu.edu.tr`.

Declarations

The datasets generated and/or analysed during the current study are available in the Kaggle repository: `https://www.kaggle.com/datasets/stackoverflow/stacksample`.

References

[1] Ndukwe I, Licorish S, and MacDonell S (2023) Perceptions on the utility of community question and answer websites like Stack Overflow to software developers, *IEEE Transactions on Software Engineering*, IEEE, vol. 49, pp. 2413–2425. https://doi.org/10.1109/TSE.2022.3220236

[2] Meldrum S, Licorish S, Owen C, and Savarimuthu B (2020) Understanding Stack Overflow code quality: A recommendation of caution, *Science of Computer Programming*, Elsevier, vol. 199, p. 102516. https://doi.org/10.1016/j.scico.2020.102516

[3] Siegmund J, Kästner C, Liebig J, Apel S, and Hanenberg S (2013) Measuring and modeling programming experience, *Empirical Software Engineering*, Springer, vol. 19, no. 5, pp. 1299–1334. https://doi.org/10.1007/s10664-013-9286-4

[4] Wu Y, Wang S, Bezemer C, and Inoue K (2018) How do developers utilize source code from Stack Overflow?, *Empirical Software Engineering*, Springer, vol. 24, pp. 637–673. https://doi.org/10.1007/s10664-018-9634-5

[5] Sela A, Hadar L, Morgan S, and Maimaran M (2019) Variety-seeking and perceived expertise, *Journal of Consumer Psychology*, Wiley. https://doi.org/10.1002/jcpy.1110

[6] Li C, Ishak I, Ibrahim H, Zolkepli M, Sidi F, and Li C (2023) Deep learning-based recommendation system: Systematic review and classification, *IEEE Access*, IEEE, vol. 11, pp. 113790–113835. https://doi.org/10.1109/ACCESS.2023.3323353

[7] Wei H, Jia H, Li Y, and Xu Y (2020) Verify and measure the quality of rule-based machine learning, *Knowledge-Based Systems*, Elsevier, vol. 205, p. 106300. https://doi.org/10.1016/j.knosys.2020.106300

[8] Wang M, Fu W, He X, Hao S, and Wu X (2020) A survey on large-scale machine learning, *IEEE Transactions on Knowledge and Data Engineering*, IEEE, vol. 34, pp. 2574–2594. https://doi.org/10.1109/TKDE.2020.3015777

[9] Bischl B, Binder M, Lang M, et al. (2023) Hyperparameter optimization: Foundations, algorithms, best practices, and open challenges, *Wiley Interdisciplinary Reviews: Data Mining and Knowledge Discovery*, Wiley, vol. 13, no. 2, p. e1484. https://doi.org/10.1002/widm.1484

[10] Majidi F, Openja M, Khomh F, and Li H (2022) An empirical study on the usage of automated machine learning tools, *Proceedings of the 38th IEEE International Conference on Software Maintenance and Evolution (ICSME)*, IEEE, pp. 190–200. https://doi.org/10.1109/ICSME55016.2022.00014

[11] Bergstra J and Bengio Y (2012) Random search for hyper-parameter optimization, *Journal of Machine Learning Research*, MIT Press, vol. 13, no. 2. [Online]. Available: https://www.jmlr.org/papers/v13/bergstra12a.html

[12] Snoek J, Larochelle H, and Adams RP (2012) Practical Bayesian optimization of machine learning algorithms, *Advances in Neural Information Processing Systems (NeurIPS)*, vol. 25, pp. 2951–2959

[13] Vasilescu B, Filkov V, and Serebrenik A (2013) Stack Overflow and GitHub: Associations between software development and crowdsourced knowledge, *Proceedings of the 2013 International Conference on Social Computing*, IEEE, pp. 188–195. https://doi.org/10.1109/SocialCom.2013.35

[14] Chen X, Xu F, Huang Y, Zhou X, and Zheng Z (2024) An empirical study of code reuse between GitHub and Stack Overflow during software development, *Journal of Systems and Software*, Elsevier, p. 111964. https://doi.org/10.1016/j.jss.2024.111964

[15] Nasehi SM, Sillito J, Maurer F, and Burns C (2012) What makes a good code example? A study of programming Q&A in Stack Overflow, *Proceedings of the 28th IEEE International Conference on Software Maintenance (ICSM)*, IEEE, pp. 25–34. https://doi.org/10.1109/ICSM.2012.6405249

[16] Storey M-A, Zagalsky A, Figueira Filho F, Singer L, and German DM (2016) How social and communication channels shape and challenge a participatory culture in software development, *IEEE Transactions on Software Engineering*, IEEE, vol. 43, no. 2, pp. 185–204. https://doi.org/10.1109/TSE.2016.2584053

[17] He J, Xu B, Yang Z, Han D, Yang C, and Lo D (2022) PTM4Tag: Sharpening tag recommendation of Stack Overflow posts with pretrained models, *Proceedings of the 30th IEEE/ACM International Conference on Program Comprehension*, ACM/IEEE, pp. 1–11. https://doi.org/10.1145/3524610.3527897

[18] Joorabchi A, English M, and Mahdi AE (2016) Text mining Stack Overflow: An insight into challenges and subject-related difficulties faced by computer science learners, *Journal of Enterprise Information Management*, Emerald, vol. 29, no. 2, pp. 255–275. https://doi.org/10.1108/JEIM-11-2014-0109

[19] Harrag F and Khamliche M (2020) Mining Stack Overflow: A recommender systems-based model, *Preprints*. https://doi.org/10.20944/preprints202008.0265.v2

[20] Silva CC, Galster M, and Gilson F (2021) Topic modeling in software engineering research, *Empirical Software Engineering*, Springer, vol. 26, no. 6, p. 120. https://doi.org/10.1007/s10664-021-10026-0

[21] Nelder JA and Mead R (1965) A simplex method for function minimization, *The Computer Journal*, Oxford University Press, vol. 7, no. 4, pp. 308–313. https://doi.org/10.1093/comjnl/7.4.308

[22] Storn R and Price K (1997) Differential evolution: A simple and efficient heuristic for global optimization over continuous spaces, *Journal of Global Optimization*, Springer, vol. 11, no. 4, pp. 341–359. https://doi.org/10.1023/A:1008202821328

[23] Liashchynskyi P and Liashchynskyi P (2019) Grid search, random search, genetic algorithm: A big comparison for NAS, *arXiv preprint* arXiv:1912.06059. https://doi.org/10.48550/arXiv.1912.06059

[24] Bergstra J, Yamins D, and Cox D (2013) Making a science of model search: Hyperparameter optimization in hundreds of dimensions for vision architectures, *Proceedings of the International Conference on Machine Learning*, PMLR, pp. 115–123. [Online]. Available: https://proceedings.mlr.press/v28/bergstra13.html

[25] Lagarias JC, Reeds JA, Wright MH, and Wright PE (1998) Convergence properties of the Nelder–Mead simplex method in low dimensions, *SIAM Journal on Optimization*, SIAM, vol. 9, no. 1, pp. 112–147. https://doi.org/10.1137/S1052623496303470

[26] Eriksson D, Pearce M, Gardner JR, Turner RD, and Poloczek M (2019) Scalable global optimization via local Bayesian optimization, *Advances in Neural Information Processing Systems (NeurIPS)*, vol. 32, pp. 5497-5508

[27] Bischl B, Richter J, Bossek J, Horn D, Thomas J, and Lang M (2017) mlrMBO: A modular framework for model-based optimization of expensive black-box functions, arXiv preprint, arXiv:1703.03373. https://doi.org/10.48550/arXiv.1703.03373

[28] Choi C, Lee Y, Chen A, Zhou A, Raghunathan A, and Finn C (2024) AutoFT: Learning an objective for robust fine-tuning, *arXiv preprint*, arXiv:2401.10220. https://doi.org/10.48550/arXiv.2401.10220

[29] Li L, Jamieson K, DeSalvo G, Rostamizadeh A, and Talwalkar A (2017) Hyperband: A novel bandit-based approach to hyperparameter optimization, *Journal of Machine Learning Research*, MIT Press, vol. 18, no. 1, pp. 6765–6816. https://doi.org/10.48550/arXiv.1603.06560

[30] Mallik N, Bergman E, Hvarfner C, Stoll D, Janowski M, Lindauer M, and Hutter F (2023) PriorBand: Practical hyperparameter optimization in the age of deep learning, *Advances in Neural Information Processing Systems (NeurIPS)*, vol. 36, pp. 7377–7391

[31] Liu T, Astorga N, Seedat N, and van der Schaar M (2024) Large language models to enhance Bayesian optimization, *arXiv preprint*, arXiv:2402.03921. https://doi.org/10.48550/arXiv.2402.03921

[32] Montaner M, López B, and De La Rosa JL (2003) A taxonomy of recommender agents on the internet, *Artificial Intelligence Review*, Springer, vol. 19, pp. 285–330. https://doi.org/10.1023/A:1022850703159

[33] He X, Liao L, Zhang H, Nie L, Hu X, and Chua TS (2017) Neural collaborative filtering, *Proceedings of the 26th International Conference on World Wide Web (WWW)*, ACM, pp. 173–182. https://doi.org/10.1145/3038912.3052569

[34] Mantovani RG, Rossi AL, Alcobaça E, Vanschoren J, and de Carvalho AC (2019) A meta-learning recommender system for hyperparameter tuning: Predicting when tuning improves SVM classifiers, *Information Sciences*, Elsevier, vol. 501, pp. 193–221. https://doi.org/10.1016/j.ins.2019.06.005

[35] Yang C, Akimoto Y, Kim DW, and Udell M (2019) OBOE: Collaborative filtering for AutoML model selection, *Proceedings of the 25th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, ACM, New York, USA, pp. 1173–1183. https://doi.org/10.1145/3292500.3330909

[36] Feurer M, Klein A, Eggensperger K, Springenberg J, Blum M, and Hutter F (2019) Autosklearn: Efficient and robust automated machine learning, in *Automated Machine Learning*, Springer, Cham, pp. 113–134. https://doi.org/10.1007/978-3-030-05318-5_6

[37] Erickson N, Mueller J, Shirkov A, Zhang H, Larroy P, Li M, and Smola A (2020) AutoGluon-Tabular: Robust and accurate AutoML for structured data, *arXiv preprint*, arXiv:2003.06505. https://doi.org/10.48550/arXiv.2003.06505

[38] Wang C, Wu Q, Weimer M, and Zhu E (2021) FLAML: A fast and lightweight AutoML library,

arXiv preprint, arXiv:1911.04706. https://doi.org/10.48550/arXiv.1911.04706

[39] Komer B, Bergstra J, and Eliasmith C (2014) Hyperopt-sklearn: Automatic hyperparameter configuration for Scikit-learn, *Proceedings of the 13th Python in Science Conference*, pp. 32–37. https://doi.org/10.25080/Majora-14bd3278-006

[40] Akiba T, Sano S, Yanase T, Ohta T, and Koyama M (2019) Optuna: A next-generation hyperparameter optimization framework, *Proceedings of the 25th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, ACM, pp. 2623–2631. https://doi.org/10.1145/3292500.3330701

[41] Yuan B, Gui S, Zhang Q, Wang Z, Wen J, Mao B, Liu J, and Yao X (2024) FairerML: An extensible platform for analysing, visualising, and mitigating biases in machine learning, *IEEE Computational Intelligence Magazine*, IEEE, vol. 19, no. 2, pp. 129–141. https://doi.org/10.1109/MCI.2024.3364430

[42] Mansion T, Braud R, Amrani A, Chaouche S, Adjed F, and Cantat L (2024) Debiai: Open-source toolkit for data analysis, visualisation and evaluation in machine learning, *Proceedings of the 20th International Conference on Autonomic and Autonomous Systems*, IARIA, Athens, Greece

[43] Jia Y, Wang H, and Chen D (2023) LAMDA-SSL: Label augmentation and manifold-based data alignment for semi-supervised learning, *Pattern Recognition*, Elsevier, vol. 144, p. 109690. https://doi.org/10.1016/j.patcog.2023.109690

[44] Altınsoy F and Öztürk MM (2026) Hyperparameter optimization web tool: HyperOpt, *Sigma Journal of Engineering and Natural Sciences*, vol. 44, no. 2, pp. 1219–1239. https://doi.org/10.14744/sigma.2026.2033

[45] Zhu Y, Li W, and Li T (2023) A hybrid artificial immune optimization for high-dimensional feature selection, *Knowledge-Based Systems*, Elsevier, vol. 260, p. 110111. https://doi.org/10.1016/j.knosys.2022.110111

[46] Pan H, Chen S, and Xiong H (2023) A high-dimensional feature selection method based on modified gray wolf optimization, *Applied Soft Computing*, Elsevier, vol. 135, p. 110031. https://doi.org/10.1016/j.asoc.2023.110031

[47] Hsu CW, Chang CC, and Lin CJ (2003) A practical guide to support vector classification, Technical Report, Department of Computer Science and Information Engineering, National Taiwan University, Taipei, pp. 1–12. Available: https://www.csie.ntu.edu.tw/~cjlin/papers/guide/guide.pdf

[48] Montañés E, Quevedo JR, and del Coz JJ (2014) Dependent binary relevance models for multi-label classification, *Pattern Recognition*, Elsevier, vol. 47, no. 3, pp. 1494–1508. https://doi.org/10.1016/j.patcog.2013.09.029

[49] Bergstra J, Bardenet R, Bengio Y, and Kégl B (2011) Algorithms for hyper-parameter optimization, *Advances in Neural Information Processing Systems (NeurIPS)*, vol. 24, pp. 2546–2554

[50] Kaggle (2023) Kaggle dataset: Stack Overflow - Stacksample. Available: https://www.kaggle.com/datasets/stackoverflow/stacksample, accessed: 2023-10-17

[51] Manning CD, Raghavan P, and Schütze H (2008) Introduction to information retrieval, Cambridge University Press. https://doi.org/10.1017/CBO9780511809071

[52] Aggarwal CC and Zhai C (2012) Mining text data, Springer. https://doi.org/10.1007/978-1-4614-3223-4

[53] Porter MF (1980) An algorithm for suffix stripping, *Program*, Emerald, vol. 14, no. 3, pp. 130–137. https://doi.org/10.1108/eb046814

[54] Salton G and Buckley C (1988) Term-weighting approaches in automatic text retrieval, *Information Processing & Management*, Elsevier, vol. 24, no. 5, pp. 513–523. https://doi.org/10.1016/0306-4573(88)90021-0

[55] Jurafsky D and Martin JH (2025) Speech and language processing: An introduction to natural language processing, computational linguistics, and speech recognition (3rd ed., draft), Available: https://web.stanford.edu/~jurafsky/slp3/

[56] Mikolov T, Sutskever I, Chen K, Corrado GS, and Dean J (2013) Distributed representations of words and phrases and their compositionality, *Advances in Neural Information Processing Systems (NeurIPS)*, vol. 26. https://doi.org/10.48550/arXiv.1310.4546

[57] Deerwester S, Dumais ST, Furnas GW, Landauer TK, and Harshman R (1990) Indexing by latent semantic analysis, *Journal of the American Society for Information Science*, Wiley, vol. 41, no. 6, pp. 391–407. https://doi.org/10.1002/(SICI)1097-4571(199009)41:6<391::AID-ASI1>3.0.CO;2-9

[58] Peruma A, Simmons S, AlOmar EA, Newman CD, Mkaouer MW, and Ouni A (2022) How do I refactor this? An empirical study on refactoring trends and topics in Stack Overflow, *Empirical Software Engineering*, Springer, vol. 27, no. 1, p. 11. https://doi.org/10.1007/s10664-021-10045-x

[59] West E (2022) Buy now: How Amazon branded convenience and normalized monopoly, MIT Press

[60] Kuhn M (2008) Building predictive models in R using the caret package, *Journal of Statistical Software*, vol. 28, no. 5, pp. 1–26. https://doi.org/10.18637/jss.v028.i05

[61] Dimitriadou E, Hornik K, Leisch F, Meyer D, and Weingessel A (2009) E1071: Misc functions of the Department of Statistics (E1071), TU Wien

[62] Ridgeway G (2007) Generalized boosted models: A guide to the gbm package. Available: https://cran.r-project.org/web/packages/gbm/gbm.pdf, accessed: 2021

[63] de Carvalho AC and Freitas AA (2009) A tutorial on multi-label classification techniques, in *Foundations of Computational Intelligence Volume 5: Function Approximation and Classification*, Springer, vol. 5, pp. 177–195. https://doi.org/10.1007/978-3-642-01536-6_8

[64] Read J, Pfahringer B, and Holmes G (2008) Multi-label classification using ensembles of pruned sets, *Proceedings of the Eighth IEEE International Conference on Data Mining (ICDM)*, IEEE, pp. 995–1000. https://doi.org/10.1109/ICDM.2008.74

[65] Liaw A and Wiener M (2002) Classification and regression by randomForest, *R News*, vol. 2, pp. 18–22

[66] Yan Y (2016) rBayesOptimization: Bayes optimization of hyperparameters, CRAN: Contributed Packages

[67] Probst P, Wright MN, and Boulesteix AL (2019) Hyperparameters and tuning strategies for random forest, *Wiley Interdisciplinary Reviews: Data Mining and Knowledge Discovery*, Wiley, vol. 9, no. 3, p. e1301. https://doi.org/10.1002/widm.1301