PSEO: Optimizing Post-Hoc Stacking Ensemble Through Hyperparameter Tuning

Beicheng Xu[1], Wei Liu[1], Keyao Ding[1], Yupeng Lu[1], Bin Cui[1]*

[1]School of CS & Key Laboratory of High Confidence Software Technologies (MOE), Peking University, Beijing, China
{beichengxu, eularioal, maodeshi}@stu.pku.edu.cn, {xinkelyp, cui.bin}@pku.edu.cn

Abstract

The Combined Algorithm Selection and Hyperparameter Optimization (CASH) problem is fundamental in Automated Machine Learning (AutoML). Inspired by the success of ensemble learning, recent AutoML systems construct post-hoc ensembles for final predictions rather than relying on the best single model. However, while most CASH methods conduct extensive searches for the optimal single model, they typically employ fixed strategies during the ensemble phase that fail to adapt to specific task characteristics. To tackle this issue, we propose PSEO, a framework for post-hoc stacking ensemble optimization. First, we conduct base model selection through binary quadratic programming, with a trade-off between diversity and performance. Furthermore, we introduce two mechanisms to fully realize the potential of multi-layer stacking. Finally, PSEO builds a hyperparameter space and searches for the optimal post-hoc ensemble strategy within it. Empirical results on 80 public datasets show that PSEO achieves the best average test rank (2.96) among 16 methods, including post-hoc designs in recent AutoML systems and state-of-the-art ensemble learning methods.

Code — https://github.com/PKU-DAIR/mindware
Extended version — https://arxiv.org/pdf/2508.05144

1 Introduction

In recent years, machine learning has advanced in diverse application domains, such as computer vision (He et al. 2016; Jiang et al. 2024b), and recommendation systems (Wang et al. 2024; Chen et al. 2025). Despite these advancements, developing tailored solutions with strong performance remains a knowledge-intensive task, requiring careful selection of suitable ML algorithms and hyperparameter tuning. To lower barriers and streamline the deployment of machine learning applications, the AutoML community has introduced the Combined Algorithm Selection and Hyperparameter Optimization (CASH) problem (Thornton et al. 2013) and proposed several methods (Hutter, Kotthoff, and Vanschoren 2019; He, Zhao, and Chu 2021) to automate this optimization process. Furthermore, it's widely acknowledged ensembling promising models often outperforms single ones (Feurer et al. 2015; Luo et al. 2024; Zhu et al. 2025). And ensemble strategies can be frequently found in the top solutions of prediction competitions (Koren 2009; Hoch 2015). As a result, recent AutoML systems (e.g., Auto-sklearn (Feurer et al. 2022), AutoGluon (Erickson et al. 2020), and VolcanoML (Li et al. 2023)) build post-hoc ensembles using base models from the optimization process of the optimal individual model and show better empirical results than the best single models.

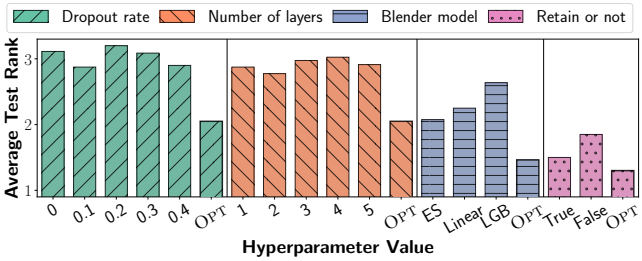


Figure 1: Average test rank of staking across each hyperparameter's values with other parameters fixed.

Despite the effectiveness of those post-hoc ensemble designs, they typically employ a fixed ensemble strategy or leave it to the user to specify, rather than performing comprehensive optimization like with a single optimal model. However, as shown in Figure 1 and detailed in Appendix G, when we apply stacking ensembles, fixing other hyperparameters and performing a grid search for each hyperparameter separately, the tuned value according to validation performance (OPT) achieve higher average test ranks than other fixed-value strategies, demonstrating the potential of optimizing ensembles. Additionally, existing AutoML systems seldom explore base model selection for ensembles; instead, all available models or the best subset are simply used. Therefore, how to search for the optimal post-hoc ensemble solution, including base model selection and strategies for ensemble base models, remains an unresolved problem.

The optimization of post-hoc ensemble is non-trivial. When we attempt to characterize an ensemble strategy using hyperparameters and search for an optimal ensemble, two challenges arise: **C1**. Since the input for post-hoc ensembles is a discrete pool of pre-trained base models, selecting models for integration is a combinatorial optimization problem,

*Bin Cui is the corresponding author

where the hyperparameter space can be exceedingly large. As a result, it is challenging to reduce the number of hyperparameters required to characterize the problem, enabling more efficient exploration. **C2**. It is challenging to fully realize the potential of the post-hoc ensemble and provide a flexible framework adaptable to various tasks, along with optional strategies for addressing specific issues.

In this paper, we propose PSEO, a new framework that optimizes post-hoc stacking ensemble through hyperparameter tuning. The contributions are summarized as follows: 1) To the best of our knowledge, PSEO is the first work to optimize post-hoc stacking ensemble through hyperparameter tuning. 2) To deal with **C1**, PSEO transforms the base model selection task into a binary quadratic programming problem and uses only two hyperparameters to control the optimization objective, achieving a trade-off between the diversity of selected base models and individual performance. 3) For **C2**, PSEO introduces the Dropout and Retain mechanisms to mitigate the problems of overfitting and feature degradation, respectively. 4) Instead of relying on fixed strategies, PSEO defines a hyperparameter space for post-hoc stacking ensembles and employs Bayesian optimization to search for the optimal strategy iteratively. Empirical results on 80 public datasets show that PSEO achieves the best average test rank (2.96) among 16 methods.

2 Background

2.1 Post-Hoc Ensemble

In post-hoc ensemble, we are given a fitted model pool $\mathbb{M} = \{m_1, m_2, ..., m_n\}$, which consists of all base models that are trained on $\mathcal{D}_{train}$ and validated during the search for the optimal individual model. The goal is to aggregate an optimal subset of models from $\mathbb{M}$ with an ensembler $f(\cdot;\theta)$ to minimize a loss function $\mathcal{L}$ in validation set $\mathcal{D}_{val}$, which is:

$$\min_{m_{i_1},...,m_{i_{n'}} \in \mathbb{M};\, \theta} \mathcal{L}(f(m_{i_1}, m_{i_2}, ..., m_{i_{n'}}; \theta), \mathcal{D}_{val}) \quad (1)$$

Intuitively, a post-hoc ensemble needs to address two key problems: (1) how to select an appropriate subset of base models from the candidate pool $\mathbb{M}$, and (2) how to determine the best ensembler $f(\cdot;\theta)$ to produce the final output.

2.2 Stacking

Among different ensemble strategies (e.g., bagging (Dietterich 2000), boosting (Freund, Schapire et al. 1996), ensemble selection (Caruana et al. 2004)), we adopt stacking (Van der Laan, Polley, and Hubbard 2007) as the ensembler because: 1) It has shown excellent experimental results, outperforming other methods (Purucker and Beel 2023a). 2) Stacking offers inherent flexibility. 3) It's a general framework, as commonly used methods can be considered special cases of one-layer stacking and serve as the blender model.

Recently, some AutoML systems like AutoGluon (Erickson et al. 2020), LightAutoML (Vakhrushev et al. 2022), and H2O (LeDell and Poirier 2020) adopt stacking for post-hoc ensemble, which trains a blender model to aggregate the predictions of a set of base models (Breiman 1996). Furthermore, in multi-layer stacking, predictions from each layer's stacker models become inputs for the next layer, forming a hierarchical structure. Table 1 shows the stacking strategies of the prominent open-source AutoML systems. We identify three common issues in their post-hoc stacking strategies.

System	Base Model	Layers	Blender Model
AutoGluon	ALL	Multiple	Ensemble Selection
LightAutoML	BEST	One/Two	Weighted Avaraging
H2O	ALL/BEST	One	Linear/LightGBM/...

Table 1: Stacking strategy of current AutoML systems.

Issue #1: Existing approaches lack an effective mechanism for selecting base models. Existing AutoML systems simply either include all models (ALL) or use the best-performing model from each algorithm class (BEST). However, both theory and experiments support that "many-could-be-better-than-all" in ensemble (Martinez-Munoz, Hernández-Lobato, and Suárez 2008; Zhou, Wu, and Tang 2002). On the other hand, it is generally accepted in the literature that diversity can help improve the performance of the ensemble (Bian and Chen 2021; Hansen and Salamon 1990; Ning et al. 2025), while selecting only the best subset lacks the assurance of base model diversity.

Issue #2: Existing works have not yet fully explored the potential of multi-layer stacking. There's a curious trend that while multi-layer stacking is feasible, existing systems restrict themselves to two layers. For example, AutoGluon's "best_quality" mode limits stacking to two layers. Similarly, LightAutoML reports using two-layer only for multi-class classification, which is the only case where consistent improvements are observed; one-layer stacking is used otherwise. H2O, meanwhile, only employs single-layer stacking. Moreover, there is also a risk of overfitting when aggregating base models, and existing systems lack effective solutions.

Issue #3: Existing approaches exhibit inflexible stacking strategies and lack optimization for specific tasks. Several parameters will determine the behavior and performance of stacking. For example, it involves the number of stacking layers and the choice of the blender model. However, only AutoGluon dynamically selects the number of stacking layers based on validation scores, while other systems fix it. As for the blender model of the final layer, AutoGluon and LightAutoML adopt fixed models, while H2O provides several choices for users to specify.

3 Method

In this section, we propose our method for post-hoc stacking ensemble optimization (PSEO). As shown in Figure 2, it first presents a **base model subset selection** algorithm with a trade-off between individual model performance and inter-model diversity. After that, PSEO builds a deep stacking ensemble with **Dropout** and **Retain** mechanisms to fully explore the potential of stacking. Finally, it defines a hyperparameter space for post-hoc ensemble, and uses a Bayesian optimizer to **search for the optimal ensemble** within it.

3.1 Base Model Subset Selection

In the scenario of post-hoc stacking, it is trivial to incorporate all available base models from the pool $\mathbb{M}$ for stacking. However, such a strategy lacks scalability. When provided with a large candidate model pool, stacking all models incurs significant overhead during both training and inference. In contrast, another approach selects only the best-performing models, which does not fully utilize the diversity of $\mathbb{M}$. Therefore, selecting the optimal base model subset for stacking is a meaningful problem worth investigating.

Nevertheless, the problem of selecting an optimal subset from a pool of candidate base models of size n is a variant of the constrained knapsack problem, which is NP-hard and impractical to solve exactly. To approximate a solution to this problem, we select a subset by focusing on two key factors: ensemble size and the trade-off between individual model performance and diversity among models.

We first use the pair-wise measures to represent the diversity of two given models. We utilize the definition in previous research (Poduval et al. 2024) that is supported by a theoretical framework and shows satisfactory empirical results. The diversity metric between two models (m_i, m_j) is:

$$D(m_i, m_j) = \mathbb{E}_{(x,y)\sim\mathcal{D}_{val}}(y - m_i(x)) \cdot (y - m_j(x)) \quad (2)$$

where for classification tasks, $m_i(x)$ is the predictive probability distribution and y is the one-hot label. For regression tasks, $m_i(x)$ is the predictive value and y is the label.

After that, we can build an error covariance matrix G, where $G_{ij} = D(m_i, m_j)$. The diagonal entries G_{ii} represent the mean square errors of model i on the validation data, while the off-diagonal entries G_{ij} quantify the error consistency between model i and j. Intuitively, a smaller diagonal entry indicates a better individual model, while a smaller off-diagonal entry suggests a larger error discrepancy between two models, implying greater diversity. Therefore, it's straightforward that a small value of $\sum_{ij} G_{ij}$ implies a good ensemble. Furthermore, to trade off between individual model performance and inter-model diversity, we introduce a weighting coefficient ω to G, where the weights of diversity and individual performance are ω and $1-\omega$:

$$\tilde{G} = (1-\omega) \cdot \mathrm{diag}(G) + \omega \cdot (G - \mathrm{diag}(G)) \quad (3)$$

Finally, given n candidate base models, selecting an optimal subset of size $n'(n' \leq n)$ can now be formulated as:

$$\min_z \; z^T \tilde{G} z \quad \text{s.t.} \quad \sum z_i = n', \quad z_i \in \{0, 1\} \quad (4)$$

This binary quadratic programming (BQP) problem is NP-hard. To enable efficient approximation, we adopt a semidefinite programming (SDP) relaxation (Zhang et al. 2006). Specifically, we lift the binary variable into a positive semidefinite matrix $Z = zz^\top$, and relax the rank-one and integrality constraints. The relaxed problem becomes:

$$\min_Z \mathrm{Tr}(\tilde{G}Z) \quad \text{s.t. } \mathrm{Tr}(Z) = n', \; Z \succeq 0, \; Z \succeq zz^T \quad (5)$$

Equation 5 can be solved by standard semi-definite programming solvers. After that, we select the indices corresponding to the n' largest values in $\mathrm{diag}(Z)$ and set their entries in z to 1, the rest to 0. When $z_i = 1$, the i-th model will be selected as one of the base models for ensemble.

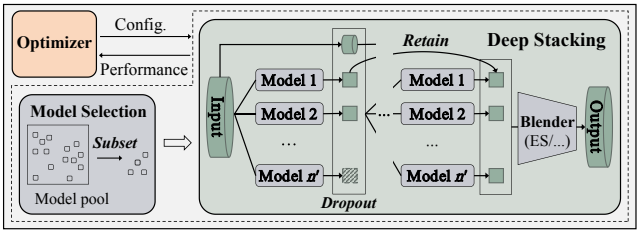


Figure 2: Overview of PSEO.

3.2 Deep Stacking Ensemble

After selecting the base model subset, we build a multi-layer stacking ensemble like the right part of Figure 2. Like AutoGluon, we reuse all base models as stacker models for each layer, which take inputs from the previous layer's predictive features and the original dataset. During training, we apply k-fold cross-validation to generate predictions for each stacker model, where each fold trains on part of the training set and predicts on the rest. This ensures stacker models always use out-of-fold (OOF) predictions and allow full training set utilization. During prediction, we average the outputs of k cross-validation models for each stacker to generate predictions on the validation or test set.

The above framework, referred to as deep stacking, can be viewed as a variant of a deep learning network, where each stacker model resembles a neuron. However, as the stacking depth increases, it becomes more complex, which can lead to two issues: **overfitting** and **feature degradation**. Overfitting refers to the phenomenon where stacking performs well on the training data but fails to generalize to test data. Feature degradation, on the other hand, describes the progressive decline in the quality of predictive features across layers. As a result, we introduce two mechanisms, Dropout and Retain, to mitigate the aforementioned issues, aiming to fully exploit the potential of deep stacking. The pseudo-code of the two mechanisms is provided in Appendix B.

Dropout. Consider a stacker model $S_{l,i}$ at layer l. If the predictive feature from the j-th stacker model $S_{l-1,j}$ of the previous layer is very close to the training set label, $S_{l-1,j}$ may become a dominating model in the training of $S_{l,i}$. It means there is a high probability that $S_{l,i}$ will overly rely on this predictive feature while neglecting other diverse features. For instance, a weighted model might assign a weight close to 1 to the j-th feature while setting others close to 0. Although it might effectively decrease the training loss, it may not generalize to test samples, leading to overfitting.

Therefore, motivated by the dropout in neural networks(Srivastava et al. 2014), we introduce the Dropout mechanisms in deep stacking. The key insight of Dropout is to encourage the ensemble to depend on diverse base models, preventing reliance on any dominating models that show strong training set performance. Specifically, receiving the predictive features from the previous layer, we first build a dropout mask for each predictive feature as:

$$m_i \sim \text{Bernoulli}(\frac{\mathcal{L}_{\text{min}}}{\mathcal{L}_i}\gamma_0)$$

where γ_0 is the only parameter representing the dropout rate,

$\mathcal{L}_{\min}$ represents the minimum loss with the training label among all predictive features from the previous layer, while $\mathcal{L}_i$ denotes the loss of the i-th predictive feature. When m_i is 1, we exclude the i-th predictive feature and train the current stacker model using the retained features. Additionally, due to the randomness introduced by the dropout process, we perform multiple repetitions to ensure robustness. For each stacker model, we apply dropout independently to each fold of cross-validation. For the blender model, we repeatedly apply dropout and training, and compute an average of multiple blender models' outputs for predicting. To sum up, by assigning higher dropout probabilities to features closer to the labels on the training set, we offer the opportunity to reduce the dominance of any single model in the ensemble.

Theorem 1. *Consider a set of uncorrelated base models predictions $\mathcal{Z} = \{z_1, z_2, ..., z_{n'}\}$ in a weighted ensemble $z_{ens} = \sum_i \beta_i z_i$. The i-th prediction is dropout with rate $d_i (d_i = \rho_i \gamma_0$ and $\rho_1 = 1 \geq \rho_i \geq 0, i \in \{1, 2, \ldots, n'\})$. If we perform N independent random samplings of $\mathcal{Z}$ and get the average weight of each prediction, as $N \to \infty$, the proportion of z_1's absolute average weight in the sum of all predictions' (i.e. $|\tilde{\beta}_1| / \sum_{i=1}^{n'} |\tilde{\beta}_i|$) decreases as γ_0 increases.*

We give a proof of Theorem 1 in Appendix A, which shows that with sufficient sampling, averaging results from multiple training with Dropout can reduce the weight proportion of the predictive feature with the highest dropout probability in the weighted ensemble. In our scenario, the contribution of the feature with the lowest training loss will be reduced. Although the feature with the lowest training loss may not always be the dominating model, they are aligned with high probability. Even if not, as shown in Section 4.3, Dropout may still reduce the weight of the dominating model, as it tends to have lower training loss and thus a higher dropout probability compared to others.

Retain. When a stacker model $S_{l,i}$ generates low-quality predictive features on out-of-sample data—due to overfitting or poor training—and subsequent models fail to correct or compensate for these degraded features, errors can propagate across layers. This leads to a progressive decline in out-of-sample performance across layers, ultimately compromising the final output. We refer to this phenomenon as feature degradation in deep stacking.

To address this issue, we introduce the Retain mechanism. The insight of Retain to prevent the continued propagation of feature degradation after it occurs at a certain stacker model. Specifically, if Retain is enabled, it compares each stacker model's predictive loss on the validation set with that of its counterpart from the previous layer (i.e., the stacker model at the same position and with the same configuration). If the current model underperforms, its output is replaced with that generated by its predecessor; otherwise, its output is retained. In this way, we offer the potential for the predictive features to progressively converge toward better representations across layers on data outside the training set.

3.3 Post-hoc Stacking Ensemble Optimization

Overall, post-hoc stacking can be divided into two stages: selecting a subset of base models and constructing a specific stacking ensemble based on the subset. Building upon the preceding design, we notice that several hyperparameters will determine the behavior of post-hoc stacking: 1) ensemble size and 2) diversity weight determine the size of the selected base model subset and the trade-off between individual performance and diversity within the subset. During the stacking stage, 3) the number of stacking layers and 4) the choice of blender model define the stacking structure; while 5) "retain or not" and 6) dropout rate control the training mechanisms to overcome the issues of overfitting and feature degradation. Therefore, we define the hyperparameter space of post-hoc stacking optimization as the combination of the above six hyperparameters. In order to identify the optimal post-hoc stacking strategy over the hyperparameter space, we employ the state-of-the-art black-box optimization technique, Bayesian Optimization (BO) (Snoek, Larochelle, and Adams 2012; Hutter, Hoos, and Leyton-Brown 2011). First, PSEO will collect the predictions of all models in the candidate pool on the validation set. At each iteration, it uses BO to suggest a new ensemble configuration. It then constructs a corresponding post-hoc ensemble on the training set and evaluates its performance on the validation set. After exhausting the budget, the best ensemble configuration found in the observations is returned. Appendix C shows the pseudo-code of PSEO and further details on the disk storage and reuse strategies to reduce computation.

3.4 Discussions

Extension. As a post-hoc ensemble framework, PSEO operates independently of the tuning algorithm used in the single best model searching stage, and can be applied to any AutoML system that produces a candidate model pool. PSEO also provides a general ensemble framework, within which other ensemble methods (e.g., ensemble selection) can be integrated as options of blender models.

Overhead Analysis. Suppose T_f is the average cost of fitting a stacker model on one fold, T_{p_1} and T_{p_2} are the average cost of a stacker model to predict on one fold of the training set and the validation set, respectively. During each iteration, the computation cost of training and validating a stacking is $khn'(T_f + T_{p_1} + T_{p_2})$, where k is the fold number of cross-validation, h is the number of layers, and n' is the ensemble size. In practice, the training and validation of kn' stacker models in each layer can be executed in parallel, which can significantly reduces the computational cost.

Difference with Previous Methods. Methods such as Auto-WEKA (Thornton et al. 2013) and Autostacker (Chen et al. 2018) also conduct a searching of ensemble. We discuss the difference between PSEO and previous methods as follows: **D1**. The first difference lies in the search space. Auto-WEKA and Autostacker adopt an expanded search space covering both ensemble strategies and base model configurations. In contrast, PSEO adopts the post-hoc ensemble, where the search space covers base model selection and ensemble methods. **D2**. The main difference is in how the ensemble search space is explored. Auto-WEKA and Autostacker perform a joint search over both base models and their ensemble, leading to a combinatorial search space whose size grows exponentially with the maximum ensem-

Hyperparameter	Type	Range
Ensemble size	Int	[5, 50]
Diversity weight	Float	[0, 0.5]
Number of layers	Int	[1, 5]
Blender model	Categorical	{ES, Linear, LGB}
Dropout rate	Float	[0, 0.4]
Retain or not	Categorical	{True, False}

Table 2: Ensemble hyperparameter space of PSEO.

ble size. In contrast, PSEO represents base model subset selection using two key hyperparameters, allowing for only a linear increase in the number of candidate configurations.

4 Experiment

4.1 Experiment Setup

Baselines. We compare the proposed PSEO with 15 baselines. 1) The best model according to the validation metric (**Single-Best**); — *Three one-step ensemble learning methods*: 2) Ensemble optimization (**EO**) (Lévesque, Gagné, and Sabourin 2016) that maintains and updates an ensemble pool greedily; 3) **Autostacker** (Chen et al. 2018) that uses an evolutionary strategy to jointly search for the stacking structure and base model hyperparameters; 4) **Opt-DivBO** (Poduval et al. 2024) that maintains an ensemble selection and considers diversity while selecting the next model to train; — *Two post-hoc ensemble designs based on ensemble selection*: 5) Ensemble selection (**ES**) (Caruana et al. 2004); 6) **CMA-ES** (Purucker and Beel 2023b) that uses an evolutionary strategy to search for an optimal ensemble weight; — *Nine post-hoc ensemble designs based on stacking*: 7)-14) Based on the post-hoc stacking strategies adopted by existing AutoML systems, we consider three key hyperparameters. First, for base model selection, we adopt two commonly used strategies: (a) selecting all available models (ALL), and (b) selecting the best-performing model from each algorithm class (BEST). Second, we vary the number of stacking layers, considering both one-layer and two-layer configurations. Third, for the final blender model, we use two widely adopted approaches: ensemble selection (ES) and linear regression (Linear). By combining these choices, we obtain eight commonly used stacking strategies in existing AutoML systems (e.g., **ALL-Linear-L2** stacks all base models for two layers, and uses a linear blender for output); 15) We extracted AutoGluon's "best_quality" ensemble strategy (**AutoGluon-**), which stacks all candidate models with up to two layers. The final number of layers is determined based on validation scores. If the time budget is limited, time is evenly allocated across layers, with models trained sequentially based on their validation scores.

Datasets and metrics. To compare PSEO with baselines, we use 80 real-world ML datasets from the OpenML repository (Vanschoren et al. 2014), 50 for classification tasks (CLS) and 30 for regression tasks (REG). The number of samples for CLS ranges from 1k to 110k, and that for REG ranges from 0.6k to 166k. Further dataset details can be found in Appendix D. We report the test error for all classification tasks to measure misclassification rates, which equals $1 - accuracy$. While for regression tasks, we use $mean\ squared\ error$ (MSE) to assess prediction error.

Construction of the base model pool. For all post-hoc ensemble methods, we construct the candidate base model pool $\mathbb{M}$ using the AutoML system VolcanoML (Li et al. 2023). It defines a CASH search space encompassing algorithmic choices and feature engineering hyperparameters, which contains 100 hyperparameters in total (see Appendix E for details). We run the Bayesian optimization method of VolcanoML over this space and collect all evaluated models to form $\mathbb{M}$. In each iteration, models are fitted on the training set and evaluated on the validation set. All the fitted models are saved to disk for ensemble utilization.

Implementation Details. Each dataset undergoes a split into three distinct sets: training (60%), validation (20%), and test (20%). Table 2 shows the specific hyperparameter space of PSEO. We implement PSEO based on OpenBox (Jiang et al. 2024a), a toolkit for black-box optimization. We used 5-fold cross-validation for stacking training and performed dropout training of the blender model with the same number of repetitions. More implementation details for other baselines and the training process are provided in Appendix F.

For all post-hoc ensemble approaches, 3600 seconds are dedicated to constructing the candidate base model pool $\mathbb{M}$, yielding an average of 437 base models per task. For the baseline using a fixed post-hoc ensemble strategy, we execute it to completion. For methods that perform additional optimization during the post-hoc ensemble phase (CMA-ES, AutoGluon-, and PSEO), we allocate an extra 3600 seconds for ensemble-level optimization. For the three ensemble learning methods, we impose a total budget of 7200 seconds, due to the adoption of a one-step optimization strategy rather than a two-step post-hoc ensemble.

4.2 Effectiveness of Base Model Subset Selection

We first conduct an experiment to validate the effectiveness of PSEO's base model subset selection algorithm. To investigate the impact of different ensemble size n' and diversity weight ω on the performance of stacking, we conduct a grid search over their feasible ranges. Specifically, n' is varied from 10 to 50 in increments of 10, while ω is varied from 0 to 0.5 with a step size of 0.1. For each combination of n' and ω, we generate a corresponding subset of base models and apply a one-layer stacking with Linear as the blender model. Then, the optimal combination (OPT) is selected based on validation performance and evaluated on the test set. We compare its test errors with those of 30 fixed hyperparameter combinations. Furthermore, we include comparisons with strategies commonly employed in AutoML systems, namely: ALL and BEST. Figure 3 presents the average test ranks of stacking ensembles constructed using base models selected by the different strategies. Each region represents a method, with color intensity and the number indicating the average rank (lighter is better). The 5×6 grid in the bottom-left corresponds to a grid search over n' and ω. The surrounding L-shaped region marks the optimal combinations selected on the validation set (OPT). The two blocks

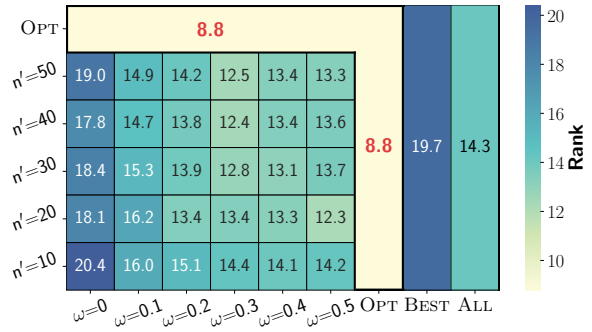
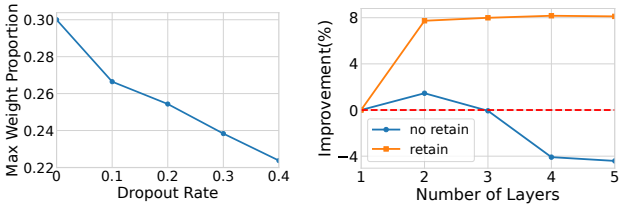


Figure 3: Average test ranks of stacking with different base model subset selection methods across 80 datasets.



(a) Proportion of max weight. (b) Improvement of stackers.

Figure 4: Effectiveness of Dropout and Retain mechanisms.

on the far right represent the BEST and ALL strategies.

We derive three observations from the results: 1) Ensembles based on low-diversity models show inferior performance. For instance, all the $\omega = 0$ strategies and BEST yield average ranks between 17.8 and 20.4. 2) PSEO's base model selection mostly outperforms ALL and BEST, with 18 and 29 wins out of 30 configurations, respectively. 3) Tuning n' and ω leads to significant improvements, with OPT achieving a rank of 8.8 vs. 12.3 for the second-best baseline. The superiority of OPT stems from two factors: 1) It balances individual model performance and inter-model diversity with diversity weights, as further demonstrated in Appendix H. 2) It optimizes the ensemble size and diversity weight to achieve a task-specific base model selection strategy.

4.3 Effectiveness of Dropout and Retain

This section evaluates the impact of Dropout and Retain mechanisms. Here, we select 30 base models with a diversity weight of 0.3 and use ES as the blender model.

In Figure 4(a), for each dropout rate γ_0 ranging from 0 to 0.4, we trained the ES five times, averaged the weights, and reported the average proportion of the largest base model weight across 80 tasks. We can observe that as γ_0 increases, the largest weight shows a decreasing trend. Appendix I further shows that as the dropout rate increases, the gap between the training and test errors gradually narrows, indicating a reduction in overfitting. In Figure 4(b), we compute the average test error of stacker models' predictive features at each layer, and report each layer's average improvement rate relative to the first layer across 80 tasks. It is observed that without the Retain mechanism, feature quality peaks at the second layer and subsequently deteriorates. In contrast, with the Retain mechanism, the predictive features show a more significant improvement, and no obvious degradation is observed as the number of layers increases. Appendix J provides further analysis of how predictive feature quality affects ensemble performance, concluding that in higher-level stacking, Retain significantly benefits the ensemble.

4.4 Comparison with Baselines

This section compares PSEO with state-of-the-art baselines on 80 real-world CASH problems. Table 3 shows the average test rank across different datasets, from which we can get five observations: 1) Ensemble indeed helps improve the performance of CASH results. The average rank of Single-Best is only 10.79, the lowest among all methods. 2) Among the ensemble learning methods, direct search for an ensemble (Autostacker, EO) is ineffective. This is because the huge search space, covering both ensemble and model hyperparameters, is challenging to optimize with Autostacker's evolutionary algorithm or EO's greedy update strategy. OptDivBO achieves a rank of 7.56, outperforming Autostacker (9.69) and EO (10.36), as it introduces ensemble selection to update the ensemble pool, thereby avoiding EO's instability caused by including poor models during optimization. 3) Among all post-hoc ensemble methods, stacking shows great potential. AutoGluon-, ALL-ES-L2, and ALL-Linear-L1 achieve ranks between 6.19 and 6.36, higher than ES (7.78) and CMA-ES (8.53). This is because ES relies on a fixed greedy strategy, which inherently limits its upper bound, and CMA-ES may lead to overfitting. 4) We observe that stacking all models (ALL-*-*) outperforms stacking the best ones (BEST-*-*), as it fully leverages the diversity in $\mathbb{M}$. However, stacking all models incurs substantial costs, with single-layer stacking taking over an hour on average and multi-layer stacking becoming even more unacceptable. 5) Among all methods, PSEO significantly outperforms the others. While the second-best baseline achieves a rank of 6.19, the rank of PSEO is 2.96. The specific loss per dataset per method is shown in Appendix N.

We attribute the superiority of PSEO to three key factors: 1) As demonstrated in Section 4.2, it provides a customized base model selection for different tasks. 2) Dropout and Retain mechanisms mitigate the issues of overfitting and feature degradation, enhancing the potential of deep stacking ensembles. We conduct ablation studies of them in Appendix L. 3) Through optimization, PSEO builds task-specific stacking structures. As shown in Appendix G, optimizing stacking hyperparameters yields significant benefits.

Normalised Improvement. To further investigate our results, we use boxplots of the normalized improvement to visualize the distribution of relative performance for all methods across 80 datasets in Figure 5. Specifically, for each dataset, 1 corresponds to the minimum test loss achieved among all methods on it, and 0 corresponds to the test loss of Single-Best. If Single-Best is the best, we penalize the relative performance of worse methods to -10. We can conclude that PSEO has better relative performance distributions than all the baselines (see the medians and whiskers).

	Single-Best	EO	Autostacker	OptDivBO	ES	CMA-ES	ALL-ES-L1	ALL-ES-L2	ALL-Linear-L1
CLS	9.44	9.12	9.46	7.10	7.36	8.14	7.44	5.86	<u>5.72</u>
REG	13.03	12.43	10.07	8.33	8.47	9.17	8.07	6.90	7.43
ALL	10.79	10.36	9.69	7.56	7.78	8.53	7.67	6.25	6.36

	ALL-Linear-L2	BEST-ES-L1	BEST-ES-L2	BEST-Linear-L1	BEST-Linear-L2	AutoGluon-	PSEO
CLS	7.36	7.50	7.28	8.26	6.74	6.18	**2.56**
REG	7.23	9.73	7.80	9.40	7.43	<u>6.20</u>	**3.63**
ALL	7.31	8.34	7.47	8.69	7.00	<u>6.19</u>	**2.96**

Table 3: Average test rank across 50 classification tasks (CLS), 30 regression tasks (REG), and all tasks (ALL) for 16 methods. A smaller rank is better. The best methods are shown in bold, while the second-best methods are underlined.

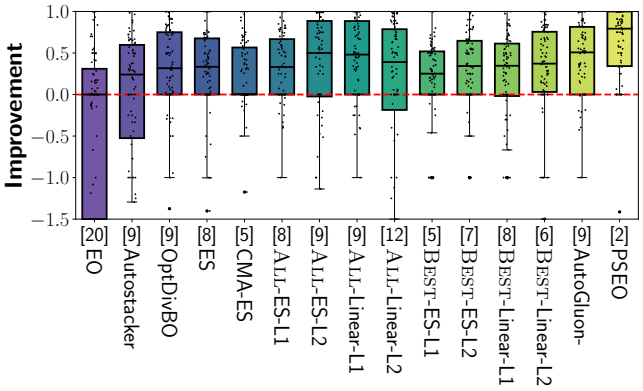


Figure 5: Normalised Improvement Boxplots: Higher is better. Each dot represents a dataset. The number in square brackets counts the outliers that are not shown in the plot.

	Single-Best	AutoGluon	PSEO
CLS	2.36	1.82	**1.38**
REG	2.63	2.03	**1.33**
ALL	2.46	1.90	**1.36**

Table 4: Average test rank in AutoGluon's search space.

4.5 Comparison with Autogluon

AutoGluon represents a state-of-the-art AutoML system with multi-layer stacking. The "best_quality" mode is not only AutoGluon's best-performing mode but also surpasses all other AutoML systems in predictive accuracy (Gijsbers et al. 2024). To align with its ensemble strategy, AutoGluon employs a more compact CASH search space compared to VolcanoML. Therefore, for a fairer comparison, we replicated its search space, which includes 108 zero-shot models with priorities. We use AutoGluon to train base models for up to 1 hour, then allocate the remaining time within the 2-hour limit to optimize the ensemble with PSEO. AutoGluon is given a 2-hour budget using "best_quality". The average test rank in 80 datasets is shown in Table 4, where we also include Single-Best as a baseline. We can conclude that both AutoGluon and PSEO outperform Single-Best, and PSEO is the best, with an average rank of 1.36. Additionally, we calculate the improvements in test loss achieved by PSEO and AutoGluon over Single-Best across all datasets, which are 10.4% and 4.0% on average, respectively. To evaluate the statistical significance of PSEO, we conduct a Wilcoxon signed-rank test for the improvements of the two methods, where the value $p = 0.002 \leq 0.05$, indicating that PSEO is statistically significantly better than AutoGluon. To sum up, the consistent success of PSEO across candidate pools derived from two systems (VolcanoML and AutoGluon) highlights its robustness and general applicability.

5 Related Work

One-step Ensemble learning. Some methods directly search for an ensemble of models in a single step. Auto-WEKA (Thornton et al. 2013) adopts a search space covering both ensemble and model hyperparameters, and solves it with Bayesian optimization. Ensemble Optimization (Lévesque, Gagné, and Sabourin 2016) maintains a fixed-size ensemble pool and optimizes each model in turn. Autostacker (Chen et al. 2018) and Neural ensemble search (Zaidi et al. 2021) focus on generating a complex ensemble via evolutionary search. Another type of work (Shen et al. 2022; Poduval et al. 2024) maintains and updates an ensemble selection with consideration of model diversity.

Ensemble Selection. Many AutoML systems, like Auto-Sklearn (Feurer et al. 2015), Auto-Pytorch (Zimmer, Lindauer, and Hutter 2021), and MLJAR (Płońska and Płoński 2021) utilize ensemble selection (Okun 2009). It iteratively adds models from the pool with replacement to maximize the ensemble performance. CMA-ES (Purucker and Beel 2023b) and Q(D)O-ES (Purucker et al. 2023) improve upon ES through choosing models with evolutionary algorithms.

6 Conclusion

In this paper, we propose PSEO, an efficient optimization framework to tune the post-hoc stacking ensemble. In PSEO, we proposed three components: a base model subset selection algorithm with a trade-off between individual model performance and inter-model diversity, a deep stacking ensemble with Dropout and Retain mechanisms, and finally a Bayesian optimizer to search for the optimal ensemble strategy. We evaluated PSEO on 80 public datasets and demonstrated its superiority over competitive baselines.

Acknowledgements

This work is supported by National Natural Science Foundation of China (U23B2048, U22B2037).

References

Bian, Y.; and Chen, H. 2021. When does diversity help generalization in classification ensembles? *IEEE Transactions on Cybernetics*, 52(9): 9059–9075.

Breiman, L. 1996. Stacked regressions. *Machine learning*, 24: 49–64.

Caruana, R.; Niculescu-Mizil, A.; Crew, G.; and Ksikes, A. 2004. Ensemble selection from libraries of models. In *Proceedings of the twenty-first international conference on Machine learning*, 18.

Chen, B.; Wu, H.; Mo, W.; Chattopadhyay, I.; and Lipson, H. 2018. Autostacker: A compositional evolutionary learning system. In *Proceedings of the genetic and evolutionary computation conference*, 402–409.

Chen, L.; Gao, C.; Lei, J.; Du, X.; Shi, X.; Luo, H.; Jin, D.; Li, Y.; and Wang, M. 2025. Privacy-preserving recommendation with coarse-grained spatiotemporal contexts. *Science China Information Sciences*, 68(4): 140104.

Dietterich, T. G. 2000. Ensemble methods in machine learning. In *International workshop on multiple classifier systems*, 1–15. Springer.

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:2003.06505.

Feurer, M.; Eggensperger, K.; Falkner, S.; Lindauer, M.; and Hutter, F. 2022. Auto-sklearn 2.0: Hands-free automl via meta-learning. *Journal of Machine Learning Research*, 23(261): 1–61.

Feurer, M.; Klein, A.; Eggensperger, K.; Springenberg, J.; Blum, M.; and Hutter, F. 2015. Efficient and robust automated machine learning. *Advances in neural information processing systems*, 28.

Freund, Y.; Schapire, R. E.; et al. 1996. Experiments with a new boosting algorithm. In *icml*, volume 96, 148–156. Citeseer.

Gijsbers, P.; Bueno, M. L.; Coors, S.; LeDell, E.; Poirier, S.; Thomas, J.; Bischl, B.; and Vanschoren, J. 2024. Amlb: an automl benchmark. *Journal of Machine Learning Research*, 25(101): 1–65.

Hansen, L. K.; and Salamon, P. 1990. Neural network ensembles. *IEEE transactions on pattern analysis and machine intelligence*, 12(10): 993–1001.

He, K.; Zhang, X.; Ren, S.; and Sun, J. 2016. Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, 770–778.

He, X.; Zhao, K.; and Chu, X. 2021. AutoML: A survey of the state-of-the-art. *Knowledge-based systems*, 212: 106622.

Hoch, T. 2015. An Ensemble Learning Approach for the Kaggle Taxi Travel Time Prediction Challenge. In *DC@ PKDD/ECML*.

Hutter, F.; Hoos, H. H.; and Leyton-Brown, K. 2011. Sequential model-based optimization for general algorithm configuration. In *International Conference on Learning and Intelligent Optimization*, 507–523. Springer.

Hutter, F.; Kotthoff, L.; and Vanschoren, J. 2019. *Automated machine learning: methods, systems, challenges*. Springer Nature.

Jiang, H.; Shen, Y.; Li, Y.; Xu, B.; Du, S.; Zhang, W.; Zhang, C.; and Cui, B. 2024a. Openbox: A Python toolkit for generalized black-box optimization. *Journal of Machine Learning Research*, 25(120): 1–11.

Jiang, R.; Zheng, G.-C.; Li, T.; Yang, T.-R.; Wang, J.-D.; and Li, X. 2024b. A Survey of Multimodal Controllable Diffusion Models. *Journal of Computer Science and Technology*, 39(3): 509–541.

Koren, Y. 2009. The bellkor solution to the netflix grand prize. *Netflix prize documentation*, 81(2009): 1–10.

LeDell, E.; and Poirier, S. 2020. H2o automl: Scalable automatic machine learning. In *Proceedings of the AutoML Workshop at ICML*, volume 2020, 24.

Lévesque, J.-C.; Gagné, C.; and Sabourin, R. 2016. Bayesian hyperparameter optimization for ensemble learning. In *Proceedings of the Thirty-Second Conference on Uncertainty in Artificial Intelligence*, 437–446.

Li, Y.; Shen, Y.; Zhang, W.; Zhang, C.; and Cui, B. 2023. VolcanoML: speeding up end-to-end AutoML via scalable search space decomposition. *The VLDB Journal*, 32(2): 389–413.

Luo, F.-M.; Xu, T.; Lai, H.; Chen, X.-H.; Zhang, W.; and Yu, Y. 2024. A survey on model-based reinforcement learning. *Science China Information Sciences*, 67(2): 121101.

Martinez-Munoz, G.; Hernández-Lobato, D.; and Suárez, A. 2008. An analysis of ensemble pruning techniques based on ordered aggregation. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 31(2): 245–259.

Ning, Q.; Guo, C.; Liu, K.; Tang, J.; Zhang, S.; Zeng, W.; and Zhao, X. 2025. Dual Sequence Modeling for Knowledge Tracing. *Data Science and Engineering*, 1–12.

Okun, O. 2009. *Applications of supervised and unsupervised ensemble methods*, volume 245. Springer Science & Business Media.

Płońska, A.; and Płoński, P. 2021. MLJAR: State-of-the-art automated machine learning framework for tabular data. *Version 0.10*, 3.

Poduval, P.; Patnala, S. K.; Oberoi, G.; Srivasatava, N.; and Asthana, S. 2024. Cash via optimal diversity for ensemble learning. In *Proceedings of the 30th ACM SIGKDD Conference on Knowledge Discovery and Data Mining*, 2411–2419.

Purucker, L.; and Beel, J. 2023a. Assembled-OpenML: Creating Efficient Benchmarks for Ensembles in AutoML with OpenML. arXiv:2307.00285.

Purucker, L. O.; and Beel, J. 2023b. CMA-ES for post hoc ensembling in AutoML: a great success and salvageable failure. In *International Conference on Automated Machine Learning*, 1–1. PMLR.

Purucker, L. O.; Schneider, L.; Anastacio, M.; Beel, J.; Bischl, B.; and Hoos, H. 2023. Q (D) O-ES: Population-based quality (diversity) optimisation for post hoc ensemble selection in AutoML. In *International Conference on Automated Machine Learning*, 10–1. PMLR.

Shen, Y.; Lu, Y.; Li, Y.; Tu, Y.; Zhang, W.; and Cui, B. 2022. Divbo: diversity-aware cash for ensemble learning. *Advances in Neural Information Processing Systems*, 35: 2958–2971.

Snoek, J.; Larochelle, H.; and Adams, R. P. 2012. Practical bayesian optimization of machine learning algorithms. In *Advances in neural information processing systems*, 2951–2959.

Srivastava, N.; Hinton, G.; Krizhevsky, A.; Sutskever, I.; and Salakhutdinov, R. 2014. Dropout: a simple way to prevent neural networks from overfitting. *The journal of machine learning research*, 15(1): 1929–1958.

Thornton, C.; Hutter, F.; Hoos, H. H.; and Leyton-Brown, K. 2013. Auto-WEKA: Combined selection and hyperparameter optimization of classification algorithms. In *Proceedings of the 19th ACM SIGKDD international conference on Knowledge discovery and data mining*, 847–855.

Vakhrushev, A.; Ryzhkov, A.; Savchenko, M.; Simakov, D.; Damdinov, R.; and Tuzhilin, A. 2022. LightAutoML: AutoML Solution for a Large Financial Services Ecosystem. arXiv:2109.01528.

Van der Laan, M. J.; Polley, E. C.; and Hubbard, A. E. 2007. Super learner. *Statistical applications in genetics and molecular biology*, 6(1).

Vanschoren, J.; Van Rijn, J. N.; Bischl, B.; and Torgo, L. 2014. OpenML: networked science in machine learning. *ACM SIGKDD Explorations Newsletter*, 15(2): 49–60.

Wang, G.; Li, X.; Guo, Z.-Y.; Yin, D.-W.; and Ma, S. 2024. SMEC: Scene Mining for E-Commerce. *Journal of Computer Science and Technology*, 39(1): 192–210.

Zaidi, S.; Zela, A.; Elsken, T.; Holmes, C. C.; Hutter, F.; and Teh, Y. 2021. Neural ensemble search for uncertainty estimation and dataset shift. *Advances in Neural Information Processing Systems*, 34: 7898–7911.

Zhang, Y.; Burer, S.; Nick Street, W.; Bennett, K. P.; and Parrado-Hernández, E. 2006. Ensemble Pruning Via Semi-definite Programming. *Journal of machine learning research*, 7(7).

Zhou, Z.-H.; Wu, J.; and Tang, W. 2002. Ensembling neural networks: many could be better than all. *Artificial intelligence*, 137(1-2): 239–263.

Zhu, J.; Zhao, X.; Sun, Y.; Song, S.; and Yuan, X. 2025. Relational data cleaning meets artificial intelligence: A survey. *Data Science & Engineering*, 10(2).

Zimmer, L.; Lindauer, M.; and Hutter, F. 2021. Auto-pytorch: Multi-fidelity metalearning for efficient and robust autodl. *IEEE transactions on pattern analysis and machine intelligence*, 43(9): 3079–3090.