DREAM: Debugging and Repairing AutoML Pipelines

XIAOYU ZHANG, Xi'an Jiaotong University, Xi'an, China
JUAN ZHAI and SHIQING MA, University of Massachusetts, Amherst, MA, USA
XIAOHONG GUAN and CHAO SHEN, Xi'an Jiaotong University, Xi'an, China

Deep Learning models have become an integrated component of modern software systems. In response to the challenge of model design, researchers proposed Automated Machine Learning (AutoML) systems, which automatically search for model architecture and hyperparameters for a given task. Like other software systems, existing AutoML systems have shortcomings in their design. We identify two common and severe shortcomings in AutoML, *performance issue* (i.e., searching for the desired model takes an unreasonably long time) and *ineffective search issue* (i.e., AutoML systems are not able to find an accurate enough model). After analyzing the workflow of AutoML, we observe that existing AutoML systems overlook potential opportunities in search space, search method, and search feedback, which results in performance and ineffective search issues. Based on our analysis, we design and implement DREAM, an automatic and general-purpose tool to alleviate and repair the shortcomings of AutoML pipelines and conduct effective model searches for diverse tasks. It monitors the process of AutoML to collect detailed feedback and automatically repairs shortcomings by expanding search space and leveraging a feedback-driven search strategy. Our evaluation results show that DREAM can be applied on two state-of-the-art AutoML pipelines and effectively and efficiently repair their shortcomings.

CCS Concepts: • **Software and its engineering** → **Software testing and debugging**; • **Computing methodologies** → *Neural networks*;

Additional Key Words and Phrases: Automated Machine Learning, Software Testing and Debugging, AutoML Systems, DL Model Testing and Repair

ACM Reference format:
Xiaoyu Zhang, Juan Zhai, Shiqing Ma, Xiaohong Guan, and Chao Shen. 2025. DREAM: Debugging and Repairing AutoML Pipelines. *ACM Trans. Softw. Eng. Methodol.* 34, 4, Article 95 (April 2025), 29 pages.
https://doi.org/10.1145/3702992

This work was supported partially by the National Key Research and Development Program of China (2023YFB3107400), the National Natural Science Foundation of China (62006181, 62132011, 62161160337, 62206217, U20A20177, U21B2018), and the Shaanxi Province Key Industry Innovation Program (2021ZDLGY01-02 and 2023-ZDLGY-38).
Authors' Contact Information: Xiaoyu Zhang, Xi'an Jiaotong University, Xi'an, China; e-mail: zxy0927@stu.xjtu.edu.cn; Juan Zhai, University of Massachusetts, Amherst, MA, USA; e-mail: juanzhai@umass.edu; Shiqing Ma, University of Massachusetts, Amherst, MA, USA; e-mail: shiqingma@umass.edu; Xiaohong Guan, Xi'an Jiaotong University, Xi'an, China; e-mail: xhguan@mail.xjtu.edu.cn; Chao Shen (corresponding author), Xi'an Jiaotong University, Xi'an, China; e-mail: chaoshen@mail.xjtu.edu.cn.
Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. Copyrights for components of this work owned by others than the author(s) must be honored. Abstracting with credit is permitted. To copy otherwise, or republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee. Request permissions from permissions@acm.org.
© 2025 Copyright held by the owner/author(s). Publication rights licensed to ACM.
ACM 1557-7392/2025/4-ART95
https://doi.org/10.1145/3702992

1 Introduction

In the Software 2.0 era, **Machine Learning (ML)** techniques that power the intelligent components of a software system are playing an increasingly significant role in software engineering. The development of **Deep Learning (DL)** brings software intelligence broader prospects with its recent advances, and DL models are increasingly becoming an integral part of software systems. The global AI software market is expected to increase from $138 billion in 2022 to $1,094 billion by 2032 [3]. The COVID-19 pandemic also accelerates the application of DL techniques in various industries. Microsoft Azure Health Bot service helps hospitals classify patients and answer questions about symptoms [9]. Google and Harvard Global Health Institute have released improved COVID-19 Public Forecasts based on AI techniques to provide a projection of COVID-19 cases [5]. As a growing trend, DL techniques are studied in a wide range of industries, involving domain experts to use DL to solve domain or even task-specific problems. Unfortunately, most domain experts have limited or no knowledge of DL techniques. This raises a great challenge for the software engineering community.

In response to the challenge, researchers proposed the concept of **Automated Machine Learning (AutoML)**, which aims to design and build ML models without domain knowledge but only the provided task. In other words, it takes the user-provided tasks as inputs and automatically builds an ML model that suits the task. At present, many AI companies and institutions have built and publicly shared AutoML systems (e.g., AutoKeras [13], NNI [15], and Cloud AutoML [14]) to assist users with little or no DL knowledge to build high-quality models. The workflow of an AutoML system is as follows. First, AutoML engines preprocess the data and extract features. Then, they define the search space and design the search strategy to look for possible model architectures and hyperparameters. These generated models are then trained and evaluated by the engine. If the validation accuracy is high enough, the process terminates. Otherwise, it continues to search for the next possible model architecture and hyperparameter. This process (i.e., generating and training models, evaluating models) iteratively continues until a model meets the predefined condition, typically a threshold for validation accuracy. This AutoML design has been used in many fields, and the existing AutoML systems lower the barrier for novices, improve the productivity of data scientists, and promote the application of DL models in various fields. For example, the AutoML system powers healthcare and drastically reduces the processing time of medical diagnosis [7]. It also supports data-driven publishing and reduces the time overhead of content classification from years to months [8].

Like other software systems, AutoML pipelines have shortcomings that prevent them from efficiently searching for and building high-quality models for a given task. The performance issue (searching for a long time without finishing) and the ineffective search issue (searched architecture and hyperparameters are useless) are the most common shortcomings in AutoML, which lead to low effectiveness and poor performance [51, 108, 115]. To understand the root cause of these shortcomings, we manually debug and analyze AutoKeras [13] and Microsoft NNI [15], two of the most popular AutoML engines. We find that existing AutoML engines ignore several potential optimization opportunities, which leads to performance and ineffective search issues. Firstly, the search space of current pipelines misses some valuable optimization options in training, making it difficult to find optimal models in the search. That is, values of many searchable parameters are fixed in existing engines. We find that changing them can benefit the pipeline. Secondly, existing search strategies can still be improved. They often have a low probability of predicting the optimal architecture/hyperparameter change and require a large number of search trials to obtain a model with the desired performance. This directly leads to the waste of time and resources in search. Last but not least, existing AutoML engines leverage validation accuracy as the only feedback to guide the search, which is insufficient to evaluate the quality of previous searches. This also makes it impossible to identify potential problems in existing models. As a result, the existing AutoML pipeline requires hundreds of trials to search for a model even on simple datasets, and the obtained model can have far lower accuracy than expected.

In this article, we propose and implement DREAM to repair the performance and ineffective search issues in the AutoML pipeline. Specifically, DREAM supplements three mechanisms on the AutoML pipeline to repair these shortcomings, namely the search space expansion, the feedback-driven search, and the feedback monitor. DREAM first expands the search space and adds effective training options for the model search, such as weight initializers and learning rate scheduling strategy, whose positive effects on model performance have been demonstrated in existing research [49, 61, 74, 88]. DREAM also leverages a novel feedback-driven search strategy to effectively search for model architectures and hyperparameters. Unlike the simple feedback in the existing pipeline that ignores training traces and the model itself, the feedback monitor in DREAM records many types of feedback (e.g., model training loss, gradients) from the model training and evaluation. Our search will leverage such observations to determine optimal actions, i.e., how to modify the model architectures and hyperparameters (in the expanded search space). By doing so, DREAM effectively fixes performance and ineffective search issues. In summary, our contributions can be categorized as follows:

— We debug the existing AutoML pipelines and identify the root causes of two common and severe pipeline shortcomings (i.e., the performance issue and the ineffective search issue).
— We propose and design the three mechanisms, i.e., search space expansion, feedback-driven search, and feedback monitor, to repair AutoML pipeline shortcomings and conduct effective model searches.
— We develop a prototype DREAM based on the proposed ideas and evaluate it on six public datasets and two AutoML pipelines. The results show that, on average, searches repaired by DREAM achieve an accuracy of 82.99% on image datasets, improving the best AutoKeras baseline results within the same search budget by 50.84%. DREAM also improves the best NNI search results by 21.34%.
— Our implementation and data are publicly available at [16].

Threat to Validity. DREAM is currently evaluated on four image datasets and two text datasets, including classification and regression tasks, which may be limited. This evaluation is a months-long effort (i.e., over 600 GPU days) on powerful hardware. Although our experiments show that the mechanisms in DREAM can efficiently and effectively conduct the model search on diverse tasks and data types, their effect may not hold on some datasets with unseen data distributions or tasks. In addition, although DREAM don't encounter an uncovered combination of conditions in experiments Section 5.2, it is still possible that the pre-built action distribution in DREAM can't cover not all possible conditions. To mitigate these threats, the search results (e.g., model architectures and hyperparameters), implementations including dependencies, and evaluation data are publicly available at [16] for reproduction. All the necessary information that can be used to reproduce our experiments and the code of the prototype DREAM will be released in our repository [16].

2 Background

Designing DL models can be challenging. In response, researchers propose AutoML to build DL models for given datasets without human intervention. The main goal of AutoML is to search for the most promising action $\mathcal{A}$ to improve the model M based on the collected feedback F, and it can

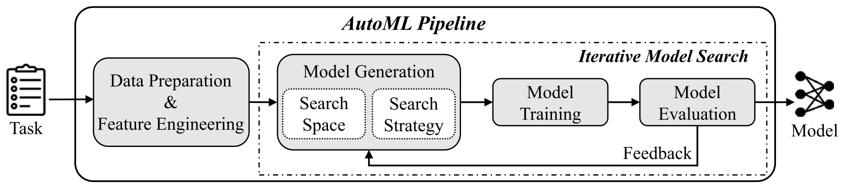


Fig. 1. The basic steps of the AutoML pipeline.

be expressed as follows:

$$\arg\max_{\mathcal{A}} \mathbb{P}(\mathcal{A}|M, F),$$

where the search action $\mathcal{A}$ indicates one operation that can be applied to modify the DNN architecture, training setting or hyperparameter, and F represents the feedback data of model training.

In AutoML, a *search* refers to one attempt to find models for the given task, and each *trial* in a search finds potential search actions and constructs a new model to train and evaluate [51]. AutoML engines evaluate every trained model with a *score*, which in most cases is the accuracy of the model. Moreover, we use *GPU hours* and *GPU days* to measure search efficiency, which are similar concepts to *person-month*. The AutoML pipeline contains a series of steps [51] (shown in Figure 1): data preparation and feature engineering, model generation, model training, and model evaluation. A trial consists of the latter three steps. The whole process is iterative until the maximum number of trials is reached or a model with the desired accuracy is found.

Data preparation and feature engineering prepare training data and extract features (if necessary) for searches and model building. The former collects data from users, cleans and normalizes it, and performs data augmentations if needed [52, 56, 91, 110, 111]. The latter conducts feature engineering, e.g., correlation coefficient, SVM-REF, and principal component analysis [47, 54, 60].

Model generation undertakes the main task of searching and generating models in the AutoML pipeline. There are two important components in an AutoML system design, *search space* and *search strategy*. The *search space* describes all possible model architectures, hyperparameters, and other configurable and searchable optimization options. AutoML engines then leverage different heuristics or statistical methods, namely *search strategies*, to search for models. There are two tasks in this step: **Neural Architecture Search (NAS)** and **Hyperparameter Optimization (HPO)**. NAS searches for suitable network architectures for given tasks, and HPO optimizes all configurable parameters [94] such as learning rates. Existing search strategies are typically gradient- or surrogate model-based methods [28, 32, 37, 39, 48, 57, 77, 86, 106, 112]. They keep all search history and conduct gradient descent or train surrogate models (e.g., Gaussian processes) to optimize towards the desired goal [59, 73, 83]. For example, the Bayesian search strategy in AutoKeras [57] builds a surrogate Gaussian process model. It fits the Gaussian process model with all previous training results to search for the next neural architecture. The Greedy method uses greedy strategies to start new trials from the current best result [13]. Hyperband [71] optimizes this method by adding dynamic resource allocations and early-stopping to these trials. The Grid search implemented by Microsoft NNI iteratively divides the current search spaces into a grid and performs an exhaustive search. **Tree-Structured Parzen Estimator (TPE)** [23] search method in NNI is an advanced Bayesian optimization method which models the probability distributions of good and bad hyperparameters. Note that AutoML pipelines do not search for duplicate models. If the search budget is infinite, any search strategy can obtain an optimal model after traversing the entire search space.

Model Training and model evaluation start after completing the model generation. The pipeline first trains the generated model. Subsequently, the model evaluation component evaluates the

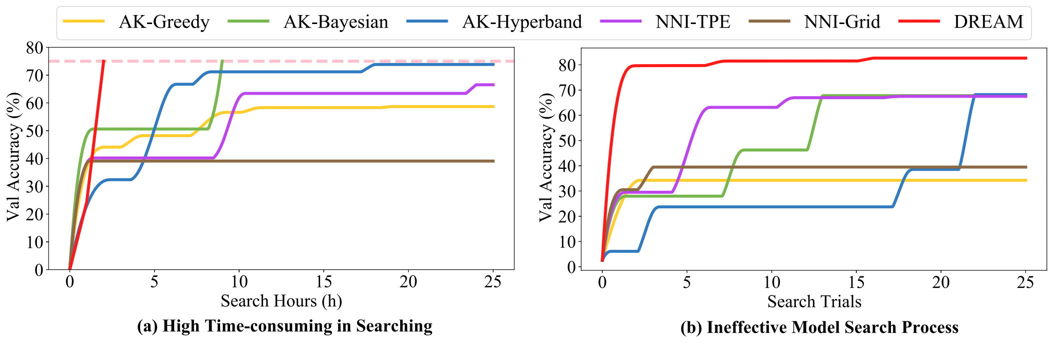


Fig. 2. Motivation cases of AutoML shortcomings.

models and the corresponding training results and provides feedback for future trials. In most implementations, the evaluation criterion is model accuracy. Modern AutoML engines [13, 38, 62, 63] also leverage early-stopping mechanisms to improve search efficiency.

3 Motivation

Existing AutoML pipelines have shortcomings, preventing them from efficiently searching for high-quality models. There are two general and server shortcomings in existing AutoML engines: *performance issue* and *ineffective search issue.* The former, *performance issue* indicates that an AutoML engine can take too much time to reach the desired goal, which leads to high carbon footprints, waste of energy, and inefficient use of resources. Due to the high cost of modern AI model training, this problem is more severe in AutoML than in other systems. The latter, *ineffective search issue* refers to the problem that AutoML engines are not able to find optimal model architectures or hyperparameters to improve the current search result, which is a typical example of functional errors. Even after numerous search trials, the AutoML pipeline still cannot obtain a model with good performance on the given task. Unlike traditional logic-based programs where testing methods can identify functional errors by formally describing the logic of the given program, the results of AutoML have no ground truth, and determining if the result is optimal or not is challenging. Thus, repairing AutoML systems is inherently challenging. In this article, we focus on two types of AutoML shortcomings.

3.1 Motivating Examples

Here, we use AutoKeras (AK) [13] and Microsoft NNI (NNI) [15], two of the most popular AutoML engines with over 9k and 13K stars on GitHub, to illustrate how common and severe the shortcomings of AutoML pipelines are. Specifically, we use two pipelines to search on the CIFAR-100 dataset with five search strategies they provide, i.e., Greedy, Bayesian, Hyperband, TPE, and Grid search.

Case I: Performance Issue. In this experiment, we set the goal of validation accuracy to 75%. Each strategy starts from the same seed and runs for no less than 24 hours if it has not achieved the goal. We use the default values of AutoML pipelines for all other configurable parameters. The result in Figure 2(a) illustrates how the performance issue severely impacts the searches of AutoML pipelines. For the AutoKeras pipeline, except for the Bayesian search strategy, which uses 9.5 hours and meets the goal, the other two methods (i.e., Greedy and Hyperband) cannot reach the target in searches. After 24 hours and 242 trials, Hyperband obtains 71.2% accuracy; and Greedy reaches 70.8% after searching for over 72 hours. The TPE and Grid search methods implemented by NNI also fail to achieve the target score, ending with scores of 66.52% and 39.07% respectively in the

24-hour search. As a comparison, the search repaired by DREAM achieves the target accuracy within 95 minutes, which is one-sixth the time used by the Bayesian strategy. Compared with the best of AutoKeras, DREAM saves 7.92 GPU hours and 2.80 kWh electricity and emits 1.21 KG less CO_2 and carbon footprints [67].

Case II: Ineffective Search Issue. The ineffective search issue causes the AutoML pipeline to continuously try useless hyperparameters and architectures in search trials, failing to improve search results. To illustrate the impact of this issue on existing pipelines, we compare different search strategies with the same search budget (i.e., 25 trials). Figure 2(b) shows results of using different search strategies. Hyperband improves the accuracy in 4/25 trials and achieves 68.21% accuracy. Each effective search trial boosts the accuracy by 16.41% on average. Bayesian, Greedy, and TPE have three fruitful trials and respectively achieve the accuracy of 67.51%, 34.23%, and 59.40%. The Grid method experiences only two accuracy increases in the search. Suffering from the ineffective search, the Grid method fails to find an optimal architecture and hyperparameters to improve the search results in the next 20 trials and finally achieves an accuracy of 39.49%. This data demonstrates that most of the trials in existing AutoML pipelines are ineffective, and it is difficult for their random or statistical model-based search methods to effectively predict actions that improve the model performance. Even for successful trials, the accuracy improvement is not significant. As a comparison, the search conducted by DREAM gains 82.66% accuracy with five productive trials. Notably, its accuracy keeps increasing even after getting to 80% (outperforming any methods of existing pipelines).

3.2 Analysis of AutoML

To understand the root causes of the shortcomings, we study the AutoML results and state-of-the-art manually crafted models, log detailed training information, and make three key observations.

Observation I: Existing AutoML engines overlook valuable search spaces, which makes it difficult to find optimal models. The search space has significant impacts on the performance of the AutoML pipeline [109]. Existing AutoML engines have implemented common model architectures and data augmentation techniques, which have shown high effectiveness in practice, to guide the search engine to find optimal models. Such designs improved the performance of AutoML by prioritizing expert knowledge chosen search space. Unfortunately, existing engines overlook a set of other possible optimizations, i.e., training configurations such as weight initializers, optimizers, and learning rate scheduling strategies, which also have remarkable impacts on the output of AutoML [34, 44, 74, 85]. In many trials, we find that changing one of the training configurations will lead to a significant accuracy boost. Specifically, we collect 200 different models from the searches conducted by three AutoKeras methods on the CIFAR-10 [66] and STL-10 [35] datasets, of which 54 have achieved an accuracy improvement of more than 10% after only modifying the training configurations such as the learning rate scheduling strategy and optimizers. This demonstrates that expanding the search space and introducing valuable search actions can benefit AutoML engines.

Observation II: Existing search strategies ignore useful feedback and have a low probability of predicting the optimal action, which causes ineffective search issues. To predict the next action (i.e., how to change the model or hyperparameters), existing AutoML engines leverage random walk or statistical models. Considering the huge search space of AutoML, random methods have low probabilities of choosing an optimal action, which leads to ineffective searches. We use the Greedy method in AutoKeras to perform 10 searches of 30 trials each on the CIFAR-100 dataset [4]. Detailed training settings are consistent with Section 5.1. On average, each search only contains 3.1 effective trials that can improve model accuracy. Affected by the ineffective search trial, the average final search results are only 63.94%. Statistical methods (e.g., Bayesian) typically require training tens of thousands of models to fit the statistical models before making good predictions, which is not

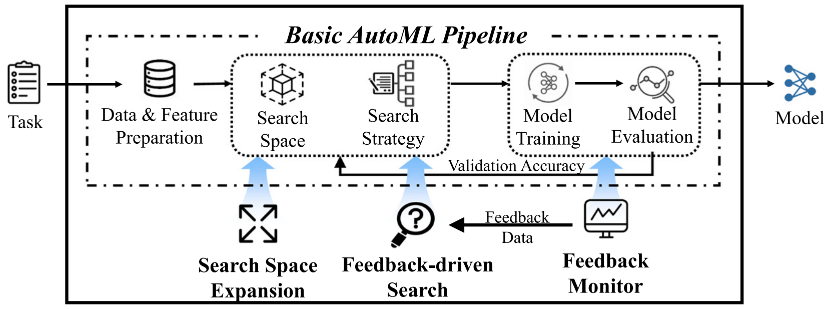


Fig. 3. Overarching design of DREAM.

practical. Moreover, whether these statistical models can accurately model the AutoML search procedure is questionable. Existing methods get non-optimal action predictions due to the lack of samples and doubtable models, which cause ineffective search issues. In software engineering research, prior works [78, 113] have observed that methods that leverage feedback data (e.g., the provenance of computing, loss values) can achieve better results. Such feedback is dynamic and context-aware, making it suitable for identifying the root causes of observed issues.

Observation III: Existing model evaluation component completely ignores the training procedure, which makes effective and efficient search infeasible. In existing AutoML pipelines, such as AutoKeras, Auto-Pytorch, and NNI, model evaluation calculates the validation accuracy or loss of the trained model as the score for the corresponding search trial and returns it to the model generation. Such a score from validation measures whether the search is successful and can be finished, and it is essential for AutoML evaluation. However, such a single metric is not sufficient to reflect the issues in the current search trial and provide the necessary information to repair AutoML shortcomings. Adding detailed feedback is critical for automatically repairing AutoML processes. Moreover, model evaluation always happens at the end stage of a trial, which makes it impossible to identify potential problems and provide timely feedback. Existing works [113] have shown that integrating model evaluation in training can help resolve training problems. The lack of detailed and timely feedback prevents the pipeline from effectively finding search actions to improve the model search, ultimately leading to performance and ineffective search issues.

4 Design

In this article, we present DREAM, an automatic repairing tool for the AutoML engines. It focuses on fixing performance and ineffective search issues. Figure 3 presents the overview of our design. It follows the original design of AutoML and adds three new mechanisms: *search space expansion* (enlarging the search space to allow finding optimal models), *feedback-driven search* (a novel search algorithm), and *feedback monitoring* (providing detailed and useful feedback of model training and evaluation, rather than validation accuracy only). We will introduce search space expansion in Section 4.1, and discuss the other two components in Section 4.2.

4.1 Search Space Expansion

The search space defines all feasible search actions in the AutoML pipeline in modifying the model architecture and hyperparameters, e.g., changing kernel size, optimizer, and architecture type. As discussed in Section 3.2, existing AutoML engines overlook many useful actions. In DREAM, we extend the search space by adding 36 feasible actions. Table 1 shows the added search actions and

Table 1. Newly Added Actions

Category	No.	Actions Description	Action Search Space
Model hyperparameters	1	Using other initializer (e.g., He Uniform) as substitution	["He Uniform", "Lecun Uniform", "Glorot uniform"]
	2	Using other activation function (e.g., SELU) as substitution	["SELU", "Tanh", "ReLU"]
Optimization configurations	3	Changing the value of `momentum` in SGD optimizer	["0.0", "0.1", "0.5", "0.9", "0.99"]
	4	Changing the value of `end learning rate` in AdamW optimizer	["1e-4", "1e-5", "2e-6", "1e-6", "0.0"]
	5	Changing the value of `weight decay rate` in AdamW optimizer	["0.001", "0.005", "0.01", "0.05", "0.1"]
Fine-tune strategy	6	Fine-tuning the pre-trained model with two-step training	["True", "False"]
	7	Fine-tuning the pre-trained model with three-step training	["True", "False"]
	8	Changing the epoch of different steps in training	["0.1", "0.2", "0.3", "0.4", "0.5"]
	9	Changing the learning rate of different steps in training	["1.0", "0.1", "0.01"]
	10	Freezing the pre-trained model in training	["all", "no", "bn"]

their feasible search space. The full action table can be found in our repository [16]. These new search actions can be categorized as follows:

— *Model hyperparameters*: Most AutoML engines use fixed model hyperparameters. For example, all searches in AutoKeras use the ReLU activation function and Xavier initializer. However, the same activation function and initializer can not work effectively on all models. Existing works have demonstrated that other activation functions and initializers have the opportunity to improve model performance [49, 61]. DREAM provides six new actions as additional optimization opportunities in the search that substitute the model activation functions and initializers to different ones, e.g., using He Uniform [49] and the SELU activation function (Rows 1 and 2 in Table 1).

— *Optimization configurations*: The appropriate optimizer configuration and learning rate schedule can play a positive role in model training [44, 74, 85]. Existing AutoML engines use fixed configurations for all searches, which may lead to poor search results. DREAM adds 15 search actions that modify the optimization setting (e.g., optimizer momentum) for different optimizers. For example, the `momentum` parameter in the SGD optimizer (Row 3 in Table 1), `end learning rate` (Row 4) and `weight decay rate` (Row 5) in the AdamW optimizer referring to the existing works [74, 88].

— *Fine-tuning strategy*: Multi-step fine-tune strategy is a commonly used fine-tune strategy for the pre-trained model, and its effectiveness has been verified in existing works [6, 10, 34]. DREAM supplements 15 search actions in five categories to search space to implement the multi-step fine-tuning. Specifically, DREAM adds the two-step and three-step fine-tuning strategies in training, as shown in Row 6 and Row 7 of Table 1. In addition, three actions are supplemented in the search space to control the transfer learning parameters, i.e., number of epochs, learning rate, and frozen status of the pre-trained model. These actions are shown from Rows 8 to 10.

4.2 Monitoring and Feedback-Driven Search

The feedback monitor in DREAM collects the feedback data from both model training and evaluation, and the feedback-driven search is a novel search algorithm leveraging such feedback. Similar to existing AutoML, model search, training, and evaluation are iterative. Algorithm 1 shows the process.

The input to search and repair components of DREAM is the same with existing AutoML: a dataset D, the maximum number of trials T_m, the target score S_t, and the seed model M. Lines 3 to 10 of Algorithm 1 show this iterative process. Firstly, in Line 4, the feedback monitor collects

Algorithm 1: Search and Repair

```
Input:   $D$ – the training dataset;            $T_m$ – the maximum number of search trials;
         $S_t$ – the target score in search;    $M$ – the seed model in a trial
Output:  $M_c$ – the selected model in search;

 1: procedure SearchAndRepair($D, T_m, S_t, M$)
 2:     $t \leftarrow 0$
 3:     while $t < T_m$ do
 4:         $F_t \leftarrow feedbackMonitor(D, M)$                    ▹ Monitor Training Feedback
 5:         $M_c, F_c, S_c \leftarrow update(M, F_t, S_t)$             ▹ Update Search Result
 6:         if $S_c \geq S_t$ then
 7:             return $M_c$                                         ▹ Early Stop
 8:         $\mathcal{A} \leftarrow selectAction(F_c, M_c)$   ▹ Select the Optimal Action Based on Feedback
 9:         $M \leftarrow buildModel(M_c, \mathcal{A})$
10:         $t \leftarrow t + 1$
11:     return $M_c$
```

feedback data F_t and model score S_t from the model training and evaluation in the tth trial. The feedback data will be used to repair the performance and ineffective search issues and conduct effective searches. In Line 5, DREAM update the candidate model M_c, corresponding feedback F_c and score S_c based on monitoring results to select the ones with the highest scores. If DREAM finds a satisfactory model before using up all budgets, it will stop the search and return the best model (Line 7). This is a typical early-stopping mechanism to optimize the search time. If not, DREAM will continue the feedback-driven search. DREAM first analyzes the feedback data F_c (Line 8), and then, based on the analysis, the search strategy dynamically selects the optimal action $\mathcal{A}$ that has the most probability to improve the candidate model M_c. Subsequently, model generation will leverage $\mathcal{A}$ and M_c to build the model for the next trial, as shown in Line 9 and the number of search trials t will increase by one (Line 10). The above process will continue until the number of trials t reaches T_m if early-stopping is not triggered.

4.2.1 Feedback Monitoring. Different from the simple feedback of the validation accuracy in the AutoML engines, the feedback monitor in DREAM collects various feedback data from the model training and evaluation. Detailed feedback data can help the feedback-driven search fix the shortcomings and conduct effective model searches. Referring to the existing work [113], the feedback monitor combines the model evaluation into each epoch in training to implement detailed and timely feedback in the pipeline, which helps repair shortcomings and improve model searches. The feedback in DREAM includes many aspects, which are shown as follows:

—Model architectures and configurations (e.g., model depth, activation functions, initializer).
—Training accuracy, loss value, and other training histories.
—Model gradients of each layer during training.
—Model weights and updates during training.

4.2.2 Feedback-Driven Search. The feedback-driven search leverages given feedback to select the optimal actions to improve the search effect, corresponding to Line 8 of Algorithm 1.

Extracting useful information from the massive feedback data F is the key to performing an effective search. The search process first summarizes F into four categories, i.e., architectures (A), convergence status (C), gradients (G), and weights (W) to describe model training and evaluation from four different angles. Then, this search process can be formalized as finding the optimal action based on given A-C-G-W observations which are extracted from the model M and the corresponding feedback F. And finally, the fomular mentioned in Section 2 can be convert into:

$$\arg\max_{\mathcal{A}} \mathbb{P}(\mathcal{A}|M, F) = \arg\max_{\mathcal{A}} \mathbb{P}(\mathcal{A}|A, C, G, W).$$

We solve the search problem by calculating the conditional probability, $\mathbb{P}(\mathcal{A}|A, C, G, W)$ for observed architectures, convergence status, gradients, and weights.

Architectures. The DNN architecture has a significant influence on the model performance. For example, the residual block in ResNet can effectively speed up the convergence of the training and help deepen the model [50, 102]. Currently, DREAM supports text and image datasets on AutoKeras and NNI, and related model architectures in their search space can be roughly classified into six types, namely **ResNet Architecture (RA)**, **EfficientNet Architecture (EA)**, **XceptionNet Architecture (XA)**, **Ngram Architecture (NA)**, **Transformer Architecture (TA)**, and **Other Architecture (OA)**. The first three are widely used DNN architectures for image classification and regression tasks [33, 50, 98]. The following two are commonly used in various language models to process text data [27, 101], and the last one refers to the models without specific architectures.

Convergence Status. We categorize the convergence status of a model based on its validation accuracy and loss value into two types: **Slow Convergence (SC)** and **Normal Convergence (NC)**. SC is a typical root cause of poor performance in searching models. The model meets SC when the accuracy changes of two contiguous epochs in training are less than 0.01 [113], and we classify the remaining conditions as NC.

Gradients. Abnormal gradients in training can implicitly explain failed trials. We group gradients into four types based on existing works [20, 53, 81, 97], i.e., **Vanishing Gradient (VG)**, **Exploding Gradient (EG)**, **Dying-ReLU Gradient (DG)**, and **Normal Gradient (NG)**. When the ratio of gradients from the input layer to the output layer is smaller than $1e-3$, DREAM considers it as a VG case. In contrast, if this ratio is over 70, DREAM treats it as EG. DG happens when the number of neurons with zero gradients is 70% of the total [113]. The occurrence of VG, EG, and DG can cause various training problems, which could lead to ineffective searches.

Weights. Once model weights overflow (or implementation bugs happen), the model performance will be severely impacted. Therefore, the weight condition is divided into **Exploding Weight (EW)** and **Normal Weight (NW)**. When model weights contain `NaN` values, the model is considered as EW.

4.2.3 Conditional Probability Calculation. Based on the summarized observations from feedback data, DREAM selects an optimal action that is the most possible to improve the current search by calculating the conditional probability $\mathbb{P}(\mathcal{A}|A, C, G, W)$. In this process, we calculate the distribution of each search action for different *A-C-G-W* conditions so that the actions that help to improve the model performance can be directly selected in the search. Although the existing studies have mentioned that several actions can improve the model performance in some situations [45, 61, 76], it is still difficult to directly measure the actual effect of each action under different conditions.

To solve this problem, we construct large-scale experiments to calculate the conditional probability and evaluate the search priority of each action under different conditions in searches. We randomly generate and train a large number of models on image and text datasets and record the conditions summarized from the feedback of these models. We record the model accuracy in training as Acc_{ori}. Next, we search all feasible actions on each model one by one and generate new models to evaluate these actions. Each search action can be written as $O-V$, where O represents an object such as model architecture or hyperparameter this action will change, and V represents the new value assigned to the object. Then we train these new models and record accuracy as Acc_{O-V}. All the training processes mentioned here include 15 epochs, and the batch size is 32. After these training processes are finished, we calculate the change of accuracy from the pair $O-V$,

Table 2. Partial Search Priority of Actions

Conditions	Search Priority			
	1st	2nd	3rd	4th
EA-NC-DG-NW	trainable=True	imagenet size=True	block type=xception	optimizer=adam weight decay
EA-NC-EG-NW	trainable=True	block type=xception	imagenet size=True	version=b7
EA-SC-DG-NW	trainable=True	imagenet size=True	block type=xception	momentum=0.1
EA-SC-EG-NW	trainable=True	learning rate=0.01	imagenet size=True	block type=xception
RA-NC-NG-NW	block type=xception	trainable=True	imagenet size=True	optimizer=adam weight decay
RA-SC-NG-NW	learning rate=0.01	trainable=True	learning rate=0.1	learning rate=0.001
OA-NC-DG-NW	block type=xception	filters=256	optimizer=adam weight decay	end learning rate=1e=5
OA-NC-NG-NW	block type=xception	separable=False	optimizer=adam weight decay	end learning rate=1e=5
OA-SC-DG-NW	block type=xception	learning rate=0.01	learning rate=0.001	optimizer=sgd
OA-SC-EG-EW	num blocks=3	block type=xception	num layers=2	separable=True
XA-NC-NG-NW	pretrained=True	trainable=True	imagenet size=True	optimizer=adam weight decay

which is written as $\Delta Acc_{O-V} = Acc_{O-V} - Acc_{ori}$. Finally, we calculate the conditional probability $\mathbb{P}(\mathcal{A}|A, C, G, W)$ as follows,

$$\mathbb{P}(\mathcal{A}|A, C, G, W) = \frac{\sum_{i=0}^{n} \Delta Acc_{O-V}^{i}}{n},$$

where n represents the amount of models under the same *A-C-G-W* condition. Note that, a larger $\mathbb{P}(\mathcal{A}|A, C, G, W)$ means that the action with the object O and value V have more possibility to improve the current search performance (i.e., model accuracy). Therefore, we give such search action $O - V$ higher priority in the search process. We apply the evaluated search priority as the default setting in the search method of DREAM to guide the pipeline to search effectively. In each trial, our system automatically chooses the action with the highest conditional probability (under observed conditions *A-C-G-W*) to help generate new models.

We display part of the search priority under different conditions in Table 2 and the whole priority table with hundreds of actions can be found in our repository. The first column lists all *A-C-G-W* conditions in the experiments. The following columns respectively show the search actions with the highest priority under these conditions. It is worth noting that the action that is more likely to improve the search performance should be prioritized in the search. In the table, the left side of the equal sign for each action is the object, that is, the hyperparameter or model architecture that will be changed, and the right side is the newly assigned value.

During the search, if the action with the highest priority has already been applied in the current model, DREAM will select the action with the next highest priority, and so on. We provide a running case in "Example 1" that detailedly explains how the priorities specified by the conditional probabilities guide the searches in DREAM. Additionally, although there are 96 possible combinations of *A-C-G-W* mathematically, a considerable portion of these combinations cannot be encountered in practice due to the conflicts between some conditions. For instance, the NaN weights will cause the model to not update properly in the backward propagation when the EW condition is met. At this time, the model gradient will also be affected and produce NaN, and the model will fail to converge during training. Therefore, the NC and NG conditions cannot occur with EW simultaneously. Moreover, benefiting from the model architecture design, such as residual block, the NaN weight (i.e., EW) can hardly occur under EA, RA, and XA conditions. Finally, we have evaluated and constructed priority for 27 sets of *A-C-G-W* conditions through the large-scale experiments. In Section 5, we evaluate DREAM on four image datasets and two

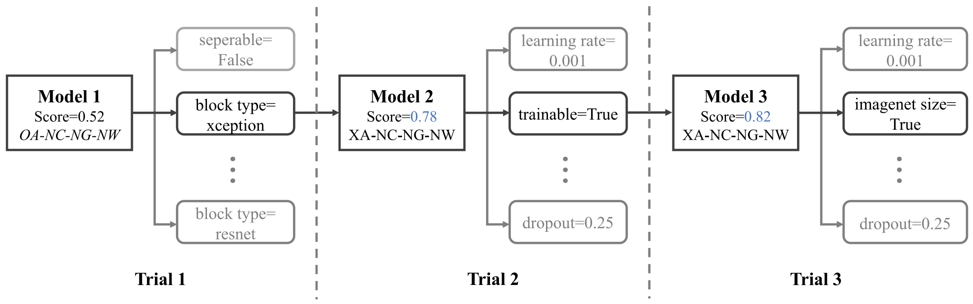


Fig. 4. The priority guides the search for actions to improve the model performance.

text datasets with diverse data content (e.g., news text, food images, car images) and tasks (i.e., regression and classification) to illustrate that these pre-calculated probabilities can effectively guide search and be generalized on a variety of data and tasks. In addition, the 27 sets of *A-C-G-W* condition combinations we constructed and evaluated have effectively covered all search trials in experiments (over 5,000 models), and no uncovered conditions are encountered. For those conditions not covered in our search priority, DREAM will choose random actions in the search. How to uncover new conditions and further update and refine the search priority of actions will be our future work.

Example 1. The case in Figure 4 provides an example of the feedback-driven search of DREAM and illustrates how the search in DREAM performs with the search priorities. For the "Model 1" with an initial score of 0.52 and the combination of conditions extracted from the internal feedback as "OA-NC-NG-NW," as shown in the 8th line of Table 2, the action with the highest priority is "block type=xception," which indicates that the object in this action is "block type" and the new value for it is "xception." It means that changing the value of "block type" of the current model to "xception" has the most potential to improve the model's performance. Therefore, DREAM applies this action and generate a new model "Model 2." After training, the search score of "Model 2" has increased to 0.78, and the combination of the conditions has turned to "XA-NC-NG-NW," whose priority is listed in the last line of Table 2. The action with the highest priority in current conditions is "pretrained=True." Since "pretrained" has already been set to "True" in "Model 2," this action is skipped and the second action "trainable=True" is selected, which increases the score of "Model 3" to 0.82. At this time, the current conditions keep "XA-NC-NG-NW," and the next action to be searched is "imagenet size=True." The search process described above is highlighted in Figure 4. The feedback-driven search in DREAM selects the actions one by one based on the conditions and the built-in priorities until the search is terminated. DREAM collects and extracts feedback and selects the search action that is the most potential to improve the model performance according to the conditions and search priorities. Compared with the existing methods, DREAM can use less time to find suitable actions in a huge search space to improve the model accuracy.

5 Evaluation

In this section, we aim to answer the following **Research Questions (RQs)**.

RQ1: How effective is DREAM in fixing AutoML shortcomings?
RQ2: How effective is each design in DREAM?
RQ3: What is the impact of different action priorities in searching models?

5.1 Setup

Datasets. We perform our experiments on four popular image datasets (i.e., CIFAR-100 [4], Food-101 [26], Stanford Cars [65], TinyImageNet [70]) and two text datasets (i.e., Reuters [17] and Text Retrieval Conference Question Classification dataset (Trec) [72]) to observe the effectiveness and generalizability on diverse tasks, data content and data types. CIFAR-100 is a widely-used colored image dataset used for object recognition and contains 100 categories. Food-101 is an image classification dataset with 101 food categories and a total of 101k images. In the experiments, we resize all images in the Food-101 dataset to the size of $300 * 300$. Stanford Cars is an image dataset with 16,185 images of 196 classes of cars. These images of cars are resized to $360 * 240$. TinyImageNet dataset consists of 100k colored images of 200 classes, and all images have been downsized to $64 * 64$. We use this dataset for both image classification and regression tasks. Reuters is a newswire dataset for document classification with 46 classes. The Trec dataset contains 5,952 labeled questions with 50 fine class labels. For a fair comparison, all strategies in experiments use the same data preprocessing procedures.

Baseline and Metric. We mainly compare DREAM with three implemented strategies in AutoKeras on six datasets for comparison, i.e., Greedy, Bayesian, and Hyperband. In addition, we compare DREAM with the TPE and Grid search methods implemented by Microsoft NNI [15] on three image and text datasets (i.e., CIFAR-100, TinyImageNet, and Reuters) to illustrate the commonality of AutoML pipeline shortcomings and the generalization of DREAM. Detail descriptions about each strategy are shown in Section 2, and we directly used configurations recommended by AutoKeras and NNI for these baseline methods. In the experiments, we use *Validation Accuracy* as the *Score* to evaluate the model performance on text and image classification tasks and use *Validation Loss* as the *Score* on the regression tasks (reducing loss in such tasks). We use *Hours* to refer to the *GPU Hours* in searches. To ensure that the models can be fully trained, the maximum epoch in training is set to 200. In addition, we enable the early stopping setting to speed up the training process. It is also the default setting in the AutoKeras engine to terminate each training process in advance and improve search efficiency.

Implementation. The prototype of DREAM implements on top of AutoKeras 1.0.12 [13], NNI 3.0 [15] and TensorFlow 2.4.3 [18], and can be applied to alleviate and repair the shortcomings on the AutoKeras and NNI pipelines. To obtain the conditional probability, we randomly generate and train 300 different models on three image and text datasets different from those used in experiments (i.e., CIFAR-10 [66], STL-10 [35], and IMDB [2]) and record the conditions summarized from the feedback of these models. All searches in the experiments of this section start with the same code recommended by AutoKeras and NNI. The batch size is set to 32 in training. All experiments are conducted on a server with Intel(R) Xeon E5-2620 2.1GHz 8-core processors, 130 GB of RAM, and an NVIDIA RTX 3090 GPU running Ubuntu 20.04 as the operating system.

5.2 Effectiveness of DREAM

Experiment Design. To evaluate the effectiveness and generalization of DREAM in fixing AutoML's ineffective search and performance issues on diverse datasets and tasks, we conduct experiments with three AutoKeras search strategies and DREAM on four image datasets and two text datasets. The dataset tasks include image classification, text classification, and image regression. We perform five sets of searches on each dataset. In addition, to illustrate the severity of the AutoML shortcomings and the generalization of DREAM, we conducted experiments using two representative search methods from NNI on three image and text datasets and compared the search results repaired by DREAM. These searches are used to observe whether DREAM can effectively alleviate the shortcomings of AutoML pipelines and guide the searches to perform more effectively. To make a

Table 3. Comparison between AutoKeras Searches and DREAM on Image Classification Tasks

Dataset	No.	Initial Score(%)	Best Score in Search (%)				Time to Reach Target (hours)				Score in Same Trials (%)				Score in Same Time (%)			
			AKG	AKB	AKH	DREAM	AKG	AKB	AKH	DREAM	AKG	AKB	AKH	DREAM	AKG	AKB	AKH	DREAM
CIFAR -100	1	10.59	63.04	65.77	79.19	83.32	-	-	1.63	0.92	63.04	65.77	78.78	83.32	63.04	52.51	79.19	83.32
	2	20.74	65.38	78.88	81.57	85.85	-	47.52	2.45	2.34	65.38	78.88	36.35	85.85	59.51	45.29	81.57	83.55
	3	2.58	77.79	67.79	80.21	82.66	18.32	-	5.45	1.80	34.23	67.79	68.21	82.66	77.79	67.79	80.21	82.66
	4	8.19	42.57	76.52	78.89	82.62	-	17.97	1.36	0.53	37.63	64.49	73.15	82.62	42.57	76.52	78.89	82.62
	5	4.81	64.82	46.92	77.01	83.03	-	-	20.49	0.53	64.82	46.92	54.30	83.03	63.61	33.03	77.01	83.03
	Avg.	9.38	62.72	67.18	79.37	83.50	-	-	6.28	1.22	53.02	64.77	62.16	83.50	61.30	55.03	79.37	83.04
Food -101	6	8.37	72.67	57.47	59.42	83.13	40.35	-	-	21.66	72.67	57.47	51.59	83.13	61.68	57.47	59.42	71.56
	7	2.30	61.44	41.37	58.74	78.23	-	-	-	45.04	61.44	41.37	50.16	78.23	13.57	31.93	58.74	65.74
	8	8.48	68.80	75.72	55.61	82.62	-	82.24	-	13.34	68.80	75.72	55.61	82.62	13.54	35.94	55.61	72.03
	9	7.57	69.22	59.24	55.98	81.33	-	-	-	23.56	69.22	59.24	55.98	81.33	14.03	54.21	52.56	70.33
	10	6.73	64.83	57.32	56.22	80.61	-	-	-	30.33	64.83	57.32	51.26	80.61	50.93	33.29	56.22	66.10
	Avg.	6.69	67.39	58.22	57.19	81.18	-	-	-	26.79	67.39	58.22	52.92	81.18	30.75	42.57	56.51	69.15
Stanford Cars	11	0.74	30.76	79.64	55.57	85.02	-	9.48	-	4.03	24.75	79.64	48.33	85.02	30.76	77.60	55.57	83.79
	12	0.56	71.53	81.99	46.66	83.66	16.87	8.38	-	7.85	42.70	81.99	24.44	83.66	71.53	81.99	46.66	83.35
	13	0.80	60.33	16.40	40.72	83.48	-	-	-	17.57	60.33	16.40	20.17	83.48	7.61	16.40	40.72	70.61
	14	0.62	44.06	46.23	58.17	83.42	-	-	-	7.06	44.06	46.23	32.86	83.42	14.79	46.23	58.17	83.42
	15	0.68	81.31	8.29	43.01	84.10	19.57	-	-	5.33	39.42	8.29	16.58	84.10	81.31	8.29	43.01	84.10
	Avg.	0.68	57.60	46.51	48.82	83.94	-	-	-	8.37	42.25	46.51	28.48	83.94	41.20	46.10	48.82	81.05
Tiny ImageNet	16	21.57	29.34	21.57	76.98	83.80	-	-	1.71	2.60	29.34	21.57	71.51	83.80	27.38	21.57	76.98	77.67
	17	9.44	62.86	42.29	80.26	79.90	-	-	4.17	2.85	62.86	42.29	44.70	79.90	62.78	42.29	80.26	76.66
	18	4.84	47.44	66.39	80.09	83.95	-	-	10.00	2.54	47.44	66.39	61.04	83.95	42.13	41.70	80.09	77.02
	19	7.68	37.89	46.74	76.61	83.60	-	-	13.50	4.41	37.89	46.74	53.59	83.60	35.24	46.74	76.61	77.22
	20	8.54	63.20	75.89	77.10	85.41	-	39.35	7.63	2.44	63.20	75.89	67.14	85.41	63.20	38.28	77.10	85.41
	Avg.	10.41	48.14	50.57	78.21	83.33	-	-	7.40	2.97	48.14	50.57	59.59	83.33	46.15	38.12	78.21	78.79
Avg.		6.79	52.70	55.02	65.73	82.99	-	-	-	9.84	52.70	55.02	50.79	82.99	44.85	45.45	65.73	78.01

fair comparison, we randomly generated the initial model and ensured that each search strategy started with the same initial model. In our experiments, we set the same search budget for each method for a fair comparison. Specifically, each method searches for at least 24 hours and 25 trials on the image datasets. Due to the smaller size of the text datasets and model parameters, the training of the text model is significantly faster than that of the image model. Therefore, we allocate them a smaller time budget of 2 hours of searches, during which all methods perform at least 50 trials. The target score is set to 70% for the classification tasks. The experiments record the search results of all the strategies in the same trials and the time cost of each search strategy reaching the search accuracy target in the whole search process to evaluate the efficiency of DREAM. Other settings are described in Section 5.1.

Results. Tables 3, 4, 5, and 6 separately show the detailed results of the effectiveness in repairing AutoML shortcomings with the same search trials and the same time budgets on diverse image and text tasks. The first column shows the datasets and the second column lists the serial numbers of the searches so that we can specify the searches and analyze them. The third column shows the score of the initial model for all search strategies. The column "Best Score in Search" denotes the best score each method achieves in the whole search. The column "Time to Reach Target" signifies the time cost of each search strategy to achieve the target accuracy of 70%. The "-" in this column represents that the corresponding search strategy fails to reach the target score in the entire search. Since the regression task has no accuracy, the corresponding cells are all "-". When we calculate the average time to reach the target score on one dataset, we only consider the cases where all searches have reached the target accuracy. We use the above two metrics to capture and assess the ineffective search and performance issues of each method. In addition, we provide two metrics to fairly compare different methods and demonstrate the improvement of

Table 4. Comparison between AutoKeras Searches and DREAM on Image Regression Tasks

Dataset	No.	Initial Score	Best Score in Search				Score in Same Trials				Score in Same Time			
			AKG	AKB	AKH	DREAM	AKG	AKB	AKH	DREAM	AKG	AKB	AKH	DREAM
Tiny ImageNet (Regression)	21	3,315.44	3,123.58	3,091.24	1,562.03	1,309.26	3,123.58	3,091.24	3,005.70	1,309.26	3,143.82	3,153.15	1,562.03	1,410.08
	22	3,284.05	3,112.14	3,169.87	2,571.74	1,520.78	3,112.14	3,169.87	2,571.74	1,520.78	3,137.12	3,169.87	2,571.74	1,520.78
	23	3,310.87	2,551.09	2,681.96	2,323.93	1,336.47	2,551.09	2,681.96	2,323.93	1,336.47	3,065.81	2,682.46	2,323.93	1,459.76
	24	6,209.52	2,194.84	3,213.25	2,547.23	1,403.24	2,194.84	3,213.25	2,585.86	1,403.24	3,138.34	3,213.25	2,547.23	1,408.79
	25	3,317.44	2,255.18	3,230.37	1,768.35	1,283.09	2,255.18	3,230.37	2,582.02	1,283.09	2,396.53	3,230.37	1,768.35	1,370.39
Avg.		3,887.46	2,647.36	3,077.34	2,154.66	1,370.57	2,647.36	3,077.34	2,613.85	1,370.57	2,976.33	3,089.82	2,154.66	1,433.96

Table 5. Comparison between AutoKeras Searches and DREAM on Text Classification Tasks

Dataset	No.	Initial Score(%)	Best Score in Search (%)				Time to Reach Target (minutes)				Score in Same Trials (%)				Score in Same Time (%)			
			AKG	AKB	AKH	DREAM	AKG	AKB	AKH	DREAM	AKG	AKB	AKH	DREAM	AKG	AKB	AKH	DREAM
Reuters	26	61.67	82.04	80.47	82.38	82.77	2.77	3.23	5.64	6.41	82.04	80.47	81.37	82.44	81.14	80.47	82.38	82.77
	27	59.32	80.13	75.93	79.97	81.99	3.50	42.05	4.98	5.57	80.13	75.93	79.97	81.48	80.13	70.03	79.97	81.99
	28	56.79	80.47	82.04	81.48	83.17	7.04	4.96	2.21	5.55	80.47	82.04	81.48	82.15	80.47	82.04	81.48	83.17
	29	59.99	79.46	80.86	81.65	83.05	9.91	14.77	6.62	4.54	78.28	80.86	81.65	82.60	79.46	80.86	81.65	83.05
	30	62.12	77.95	80.02	80.98	83.22	12.67	3.15	6.60	5.84	77.95	80.02	79.97	83.22	77.05	80.02	80.98	83.22
	Avg.	59.98	80.01	79.87	81.29	82.84	7.18	13.63	5.21	5.58	79.78	79.87	80.89	82.38	79.65	78.69	81.29	82.84
Trec	31	27.06	71.72	76.78	72.00	79.31	32.24	20.56	10.74	2.92	71.25	76.78	72.00	77.62	71.72	72.57	72.00	79.31
	32	27.25	79.49	78.09	71.63	79.49	3.49	2.53	7.27	7.07	79.40	78.09	71.63	79.49	79.49	78.09	71.63	79.49
	33	25.56	79.12	78.65	76.12	79.40	20.97	30.63	2.39	6.98	75.84	78.65	76.12	79.40	79.12	78.65	76.12	79.40
	34	26.50	78.28	79.96	69.29	79.31	10.98	26.56	-	7.90	77.90	79.87	66.29	79.31	78.28	79.96	69.29	79.31
	35	26.12	79.21	77.81	74.81	79.49	19.43	1.64	18.72	5.85	79.21	77.81	74.81	79.49	79.21	77.81	74.81	79.49
	Avg.	26.50	77.57	78.24	72.77	79.40	17.42	16.38	-	6.14	76.72	78.24	72.17	79.06	77.57	77.42	72.77	79.40
Avg.		43.18	79.37	79.21	77.11	81.13	10.82	14.59	-	6.19	78.80	79.20	76.55	80.87	79.19	78.54	77.11	81.13

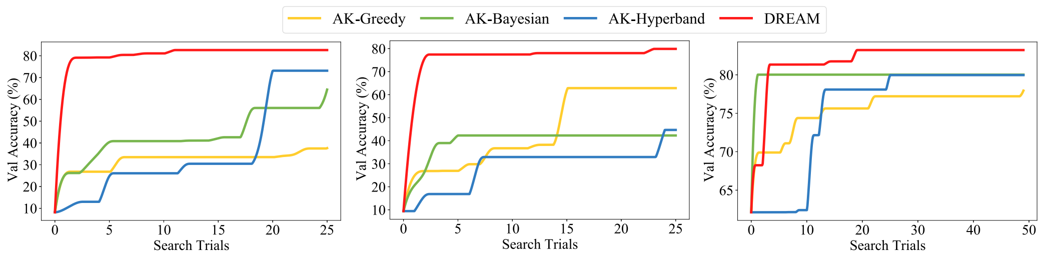
DREAM over existing pipelines. The column "Score in Same Trials" denotes the best score each method achieves within the same search trials (i.e., 25 trials for image datasets and 50 trials for text datasets). "Score in Same Time" indicates the best search score of each method within the same time budget in searches (i.e., 24 hours for image datasets and 2 hours for text datasets). In addition, the cells in blue separately correspond to the searches with the highest score reached in the given trials, the maximum improvement, the least time cost to reach the target, and the highest score in the given time budget. The columns "AKG," "AKB," and "AKH" refer to the abbreviation of the search strategies implemented in the AutoKeras pipeline, i.e., Greedy, Bayesian, and Hyperband. The columns "TPE" and "Grid" refer to the search strategies implemented in the Microsoft NNI pipeline. The column DREAM shows the search results of our system. To directly illustrate the effectiveness of DREAM in repairing the ineffective search issue of the AutoML pipeline, Figure 5 shows the comparison between search strategies of No. 4, No. 17, and No. 30 searches in the same search trials. We use "AK" and "NNI" in legends to represent the AutoML pipeline of each search strategy, namely AutoKeras and Microsoft NNI. The X-axis in the figure shows the number of the searched trials, and the Y-axis is the best score in searches. In addition, Figure 7 displays the comparison chart between the results of search strategies in No. 8, No. 13, and No. 37 searches within the same time budget to demonstrate the effectiveness of DREAM in repairing the performance issue. The X-axis in these figures is the search time, and the Y-axis is the best score during searches.

Analysis. The experiment results illustrate the effectiveness of DREAM in repairing the shortcomings of the AutoML pipelines and conducting effective model searches.

From Tables 3, 4, and 5, we can observe that the searches guided by internal training feedback in DREAM can search effectively on the datasets and obtain outstanding performance. The repaired

Table 6. Comparison between NNI Searches and DREAM

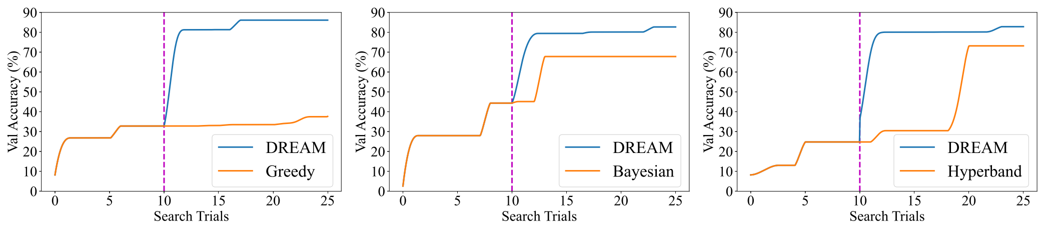
Data Type	Dataset	No.	Initial Score(%)	Best Score in Search (%)			Time to Reach Target (minutes)			Score in Same Trials (%)			Score in Same Time (%)		
				TPE	Grid	DREAM	TPE	Grid	DREAM	TPE	Grid	DREAM	TPE	Grid	DREAM
Image	CIFAR -100	36	45.29	67.26	45.38	82.43	-	-	139.80	67.26	45.38	82.43	61.29	45.38	82.43
		37	33.29	67.05	39.07	85.63	-	-	81.83	67.05	39.07	85.63	66.52	39.07	83.25
		38	22.82	67.51	39.49	80.64	-	-	82.37	67.51	39.49	80.64	61.82	39.49	80.64
		39	35.73	67.45	38.66	85.61	-	-	73.20	67.45	38.66	85.61	61.30	38.66	85.61
		40	31.12	66.39	42.30	83.57	-	-	59.40	66.39	42.30	83.57	61.31	42.30	83.57
		Avg.	33.65	67.13	40.98	83.58	-	-	87.32	67.13	40.98	83.58	62.45	40.98	83.10
	Tiny ImageNet	41	26.82	70.60	29.16	82.51	642.60	-	156.76	70.60	29.16	82.51	70.60	29.16	77.07
		42	27.23	70.33	29.60	83.86	647.28	-	158.49	70.33	29.60	83.86	70.33	29.60	83.70
		43	4.79	70.53	29.47	84.41	447.29	-	166.29	70.53	29.47	84.41	70.53	29.47	83.41
		44	26.70	70.57	30.10	83.67	458.92	-	149.24	70.57	30.10	83.67	70.57	28.82	74.26
		45	31.01	72.10	36.51	84.73	377.49	-	289.50	72.10	36.51	84.73	70.55	36.51	77.44
		Avg.	23.31	70.83	30.97	83.83	514.72	-	184.05	70.83	30.97	83.83	70.52	30.71	79.18
	Avg.		28.48	68.98	35.97	83.70	-	-	135.69	68.98	35.97	83.70	66.48	35.85	81.14
Text	Reuters	46	65.32	68.92	69.28	81.84	-	-	9.79	68.92	69.28	81.84	68.92	69.28	76.31
		47	63.62	70.44	68.43	88.46	8.88	-	5.33	70.44	68.43	80.43	70.44	68.43	88.46
		48	62.96	68.70	69.37	82.19	98.31	124.92	7.55	68.70	69.37	82.19	68.12	69.28	81.33
		49	63.22	68.70	68.66	81.04	-	-	6.53	68.70	68.66	81.04	67.94	68.25	72.26
		50	65.63	69.50	69.99	80.12	128.07	-	5.96	69.50	69.99	80.12	68.03	69.06	76.46
	Avg.		64.15	69.25	69.15	82.73	-	-	7.03	69.25	69.15	81.12	68.69	68.86	78.96



(a) No.4 Search on CIFAR-100 Dataset (b) No.17 Search on Tiny ImageNet Dataset (c) No. 30 Search on Reuters Dataset

Fig. 5. Evaluation results in the same search trials.

searches of DREAM achieve an average score of 82.99% on image classification tasks in the searches, which is significantly better than the score of other AutoKeras baselines (i.e., 52.70%, 55.02%, 65.73%). Figure 5(b) provides the search results of DREAM and AutoKeras methods on the Tiny ImageNet dataset within 25 trials to intuitively demonstrate the search effect of DREAM. The search in DREAM achieves a score of about 76% in the second trial. Then it continues to improve the performance in the 5th, 12th, and 23rd trials and finally reaches the score of 79.90%. By contrast, the best search strategy implemented in AutoKeras only achieved 62.86% in all 25 trials, and after the 16th trial, it couldn't improve anymore. In Figure 5(c), the baseline methods in AutoKeras improve the search score by up to 17.90% in 50 search trials but still fail to achieve a score exceeding 80%. In contrast, the search repaired by DREAM has exceeded 80% score in the 3rd trial and ultimately reached 83.22% accuracy after three fruitful trials. The feedback-driven search in DREAM can also conduct effective searches on image regression tasks. As shown in Table 4, the searches repaired by DREAM achieve a validation loss of 1370.57 in search, which is 36.39% lower than the best baseline method Hyperband. DREAM can also obtain better search results (i.e., 81.13%) than AutoKeras baselines on the text classification tasks, which are improvements of up to 4.02% compared to the AutoKeras methods. These results further demonstrate the effectiveness and generalizability of DREAM on



(a) DREAM VS Greedy (b) DREAM VS Bayesian (c) DREAM VS Hyperband

Fig. 6. The search results by DREAM after 10 trials' history.

diverse tasks, data types, and data contents. Note that compared with the repair results on the Stanford Cars dataset, DREAM has achieved smaller improvements on datasets such as Reuters and Trec. Our analysis shows that this is because the latter datasets are relatively simple, and the local optimal model searched by the AutoML pipeline can still achieve high accuracy, leading to a small improvement from DREAM's repair.

To further present the effectiveness of DREAM in repairing AutoML shortcomings, we repair some searches that are suffering from the ineffective search issue. These searches are guided by AutoKeras search strategies, but their performance cannot improve significantly after 10 search trials. DREAM takes their 10th trials as starting points and then repairs and conducts the searches for another 15 trials. Figure 6 shows a comparison of the searches improved by DREAM and the original searches on the CIFAR-100 dataset, where the purple dotted line represents the starting point of the repaired searches of DREAM. We can observe that the effectiveness of the repaired search is significantly improved. It can quickly improve the accuracy by over 75% within a few trials, far exceeding other search strategies in the AutoKeras pipeline. This observation further illustrates that DREAM is equally effective across different search starting points and has significant advantages over the baseline methods.

The column "Time to Reach Target" in Tables 3 and 5 show that all searches repaired by DREAM achieve the target accuracy of 70% in the search process. However, only 20/60 searches performed by the three search strategies in AutoKeras achieve the target accuracy, which suffers from poor search performance. Among them, the Hyperband search strategy with the best search performance only achieves the target score in 10/20 searches. The last column further demonstrates the effectiveness of DREAM in improving search performance. AutoKeras search strategies separately achieve an average accuracy of 44.85%, 45.45%, and 65.73% on image classification tasks within 24 hours, which are lower than the search results of DREAM within the same time budget (78.01%). In contrast, DREAM improves by 12.28% compared with the best AutoKeras method and takes only 9.84 hours on average to reach the target score. The improvement of DREAM in search efficiency can also be observed in text datasets. As shown in Table 5, the time cost of DREAM to reach the target score is reduced by 42.79% and 57.57% compared with Greedy and Hyperband in AutoKeras. In addition, DREAM separately improves the best score AutoKeras methods achieve in the given time budget by 1.94%, 2.59%, and 4.02%. Take Figure 7(a) for example, we can notice that the search repaired by DREAM reaches an accuracy of nearly 70% within 6 hours after starting searching and further improves and reaches the target score in 13.34 hours. Hyperband strategy with the best performance of the AutoKeras search strategies only achieves an accuracy of 55.61% in about 15 hours and ultimately fails to meet the search target. Bayesian search strategy, which solves the model searching as an ML problem and learns from the search histories, only obtains an accuracy of 35.94% in 24 hours and finally spends 82.24 hours to reach the target score. Compared with the Bayesian strategy, the efficiency of the search repaired DREAM improves by 5.16 times, and

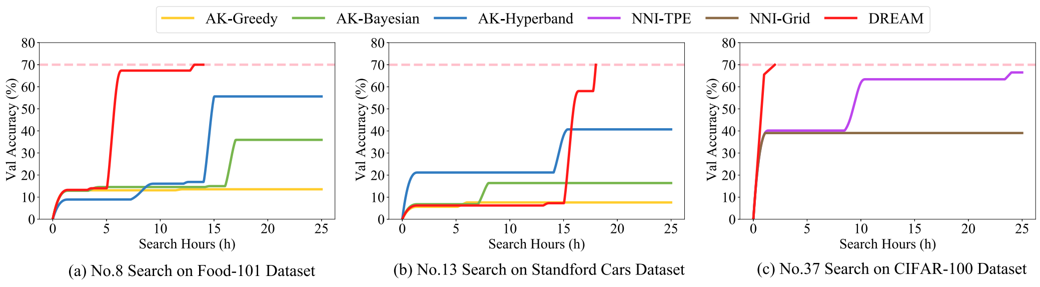


Fig. 7. The search results in the same time budget.

DREAM saves 24.14 kWh of electricity and emits 10.43 kg less CO_2 and carbon footprints in this single search. The above results prove the effectiveness of DREAM in searching models in the given time budget, and the searches in DREAM have significantly higher search performance and search efficiency.

Furthermore, the search results in Table 6 demonstrate that the ineffective search and performance issues are common shortcomings of the AutoML pipelines, and the mechanisms of DREAM can be generalized to other pipelines and effectively repair shortcomings. Even after using up all the search budget, the TPE and Grid searches in NNI can only achieve an average score of 68.98% and 35.97% on two image datasets and 69.25% and 69.15% on the text datasets. Only TPE can achieve the target accuracy of 70% in searches on the TinyImageNet dataset, which takes 514.72 minutes on average. In contrast, the repaired searches of DREAM have achieved a score of 83.70% on image datasets, showing an improvement of 14.72% over the search results of the best NNI method. Moreover, the searches of DREAM can achieve the target accuracy in all searches on image and text datasets with a time cost of 135.69 and 7.03 minutes, respectively. Take the No. 37 search in Figure 7(c) as an example. The TPE and Grid methods in NNI separately achieve scores of 66.52% and 39.07% after 24 hours of search, and none of them reaches the target. In contrast, the search repaired by DREAM achieves 76.44% at the second hour and then continuously searches for potential actions to improve the model and finally achieves 83.25% after four effective trials within 24 hours.

Answer to RQ1: AutoML pipelines suffer from ineffective search and performance issues. DREAM can effectively mitigate these shortcomings and search for a model with excellent performance on image and text datasets of different tasks and content within several search trials or hours, ultimately achieving search results that outperform baseline methods. Especially on large-scale datasets such as Stanford Cars and Tiny ImageNet, the repair effect of DREAM is more prominent.

5.3 Ablation Study

Experiment Design. DREAM provides three mechanisms to repair the performance issue and ineffective search issue in the AutoML engines, which are the search space expansion, the feedback-driven search, and the feedback monitoring. To understand the contribution of the solutions in DREAM, we conduct two ablation experiments on the CIFAR-100 and TinyImageNet datasets to study the performance of the search space expansion and the pre-built feedback-driven search in repairing. Considering that the feedback-driven search fixes the shortcomings together with the feedback monitoring, we treat these two mechanisms as one solution in the experiments. The first experiment uses the three search methods of the AutoKeras pipeline to search on the expanded search space to study whether the search space expansion in DREAM helps improve the search

Table 7. Ablation Study Results of Search Space Expansion

Dataset	Search Method	Initial Score (%)	Best Score in Search (%)			Score Improvement (%)			Time to Reach Target (hour)			Score in Same Time (%)		
			Ori.	Exp.	Ratio	Ori.	Exp.	Ratio	Ori.	Exp.	Ratio	Ori.	Exp.	Ratio
CIFAR-100	AKG	20.73	34.23	83.66	2.44	13.50	62.94	4.66	18.32	7.62	0.42	77.79	83.66	1.08
	AKB	22.26	67.79	80.47	1.19	45.53	58.21	1.28	-	14.30	-	67.79	80.47	1.19
	AKH	4.68	68.21	82.70	1.21	63.53	78.03	1.23	5.45	4.00	0.73	80.21	82.70	1.03
	Avg.	15.89	56.74	82.28	1.45	40.85	66.39	1.63	-	8.64	-	75.26	82.28	1.09
Tiny ImageNet	AKG	21.58	29.34	81.40	2.77	7.76	59.82	7.71	-	15.43	-	27.38	81.40	2.97
	AKB	22.24	22.24	78.07	3.51	0.00	55.84	-	-	47.05	-	21.57	21.81	1.01
	AKH	19.94	71.50	81.28	1.14	51.56	61.34	1.19	1.71	3.44	2.01	76.98	81.28	1.06
	Avg.	21.25	41.02	80.25	1.96	19.77	59.00	2.98	-	21.98	-	41.97	61.50	1.47
Avg.		18.57	48.88	81.27	1.66	30.31	62.70	2.07	-	15.31	-	58.62	71.89	1.23

performance. In the second experiment, we use the original search space in all searches and only compare the search strategy in DREAM with three AutoKeras search methods to explore whether the feedback-driven strategy can conduct more effective and efficient searches. All experiments search for at least 25 trials and 24 GPU hours.

Results. Table 7 shows the ablation experiment results of the search space expansion. The first three columns in this table show the dataset, search methods, and the score of the initial models. The column "Best Score" lists the best score each search reaches in the complete search, and "Score Improvement" shows the absolute accuracy improvement that the best score achieves. "Time to Reach Target" and "Score in Same Time" separately show the time cost to achieve the target score of 70% and the best score each method achieves in 24 hours. Columns "Ori.," "Exp.," and "Ratio" separately display the search results on the original and the expanded search space, and the ratio between the two results. Blue cells mark the best results between different search methods. In addition, Figure 8 shows the ablation experiment results of the feedback-driven search. The X-axis shows the searched trials, and the Y-axis is the best score during searches.

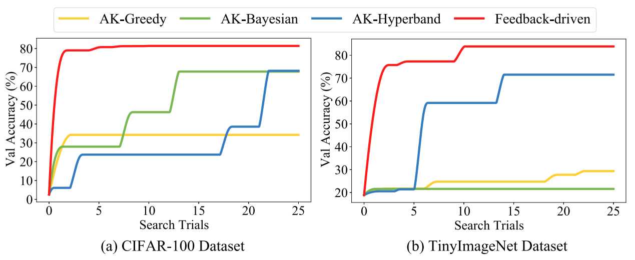


Fig. 8. Ablation study of feedback-driven search on two datasets.

Analysis. The experiment results illustrate the effectiveness of each solution in DREAM in searching on given tasks. Table 7 shows that the expanded search space can lead to much better models on the given datasets. It achieves an average accuracy of 81.27% on the six searches with three search strategies of the AutoKeras pipeline, which is significantly better than the 48.88% accuracy of the searches on the original search space. For instance, on the TinyImageNet dataset, the search on the expanded search space improves the initial score by 59.00%, which is 2.98 times the improvement of the original search space. In addition, we can observe that it is difficult for the search strategies to achieve the target accuracy rate of 70% in the original search space, let alone 80%. In contrast, after implementing the search space expansion, almost all methods reach 80% accuracy within the same search trials and search time, illustrating the role of the additional search actions in fixing search performance problems. The last two columns of Table 7 show that after expanding the search space, the search methods take an average of 15.31 hours to reach the target accuracy. On average, the search results within 24 hours can reach a score of 71.89%, which is an increase of 22.63% compared to the score before expansion. This shows that expanding the search space can help improve the search efficiency and performance of the AutoML pipeline. We have manually analyzed the optimal models found in the expanded search space. The analysis results indicate that all these models use additional training optimization search actions, such as fine-tuning with two-step training and three-step training. These expanded search actions greatly improve the model performance and help the models achieve accuracy far exceeding those of the models on the original search space.

Figure 8 shows that the feedback-driven search has significant improvement in the effectiveness and efficiency of searching models. This search method can quickly improve the accuracy to about 80% within several trials, while the baseline search strategies are difficult to achieve 70% accuracy during the whole 25 trials. Take Figure 8(a) as an example, the feedback-driven search process first searches for the actions that change the model architectures to *XceptionNet* and enables the trainable setting of the pre-trained model, which improves the model accuracy to 68.77%. Then it changes the value of `imagenet size` parameter in the model and applies other actions according to the priority under the current conditions to further improve the search score, and finally breaks the 80% accuracy in the fourth trial. Notably, the feedback-driven search takes 0.61 hours to reach 70% accuracy, while Hyperband, the best-performing search strategy in AutoKeras, requires 5.45 hours. The Bayesian search strategy doesn't even reach the target score in 43 hours of search.

In addition, the results in Table 7 further highlight the contribution of the feedback-driven search to solving the shortcomings of the AutoML pipelines and improving search efficiency. Replacing the feedback-driven search in DREAM with other search strategy leads to a significant decrease in search performance. Taking the results on the TinyImageNet dataset as an example, we can observe that compared with the search effect of DREAM in Table 3, replacing feedback-driven search with other baseline methods results in a 6.53× decrease in search efficiency (i.e., the time to reach the target is improved from 2.97 hours to 21.98 hours), and the best score within 24 hours decreases from 78.79% to 61.50%.

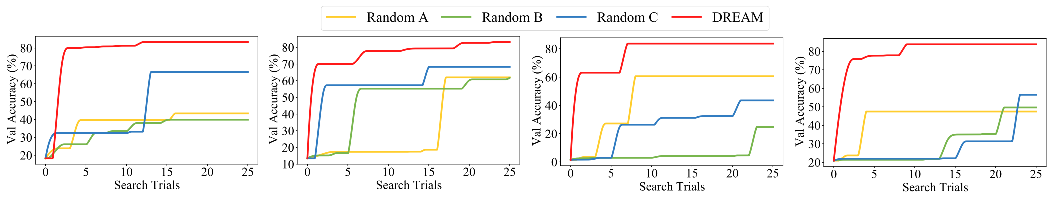
Answer to RQ3: The two solutions implemented in DREAM can effectively solve the shortcomings in the existing AutoML pipelines and have significant contributions to conducting effective searches. Specifically, the search space expansion can assist the search method to find a model architecture with much better performance. The feedback-driven search method can drastically improve the efficiency of the model searching process so that the models with excellent performance can be found in the early stage of the searches.

5.4 Effects of Different Priorities

Experiment Design. DREAM leverages a built-in search priority to determine the optimal action to improve the search under different conditions. To verify the effect of the search priority, we conduct a comparative experiment with different priority settings on four datasets. In the experiment, we randomly generate three different search priorities to compare with the built-in search priority in DREAM. Each search contains 25 trials, and the other configurations follow Section 5.1. The initial models and training configurations of the searches with different priorities are the same.

Results. The search results of different priorities on four datasets are shown in Figure 9. The red line represents the search results of the built-in search priority of DREAM, while the green, blue, and orange lines show the random search results.

Analysis. As shown in Figure 9, the search effect of DREAM is significantly better than the effect based on random priorities. Additionally, DREAM can often obtain outstanding search results within the first ten trials. In contrast, the searches with random priority are always difficult to



(a) CIFAR-100 Dataset (b) Food-101 Dataset (c) Stanford Cars Dataset (d) Tiny ImageNet Dataset

Fig. 9. The search results with different priorities on different datasets.

improve the training effect of the models. They often fail to achieve 60% accuracy in dozens of hours of searching. This significant difference illustrates the effectiveness of the built-in search priority of DREAM in repairing searches.

To further analyze the impact of priority on search performance, we manually analyze the searches in Figure 9 to understand how the priority in DREAM guides the search process. Taking Figure 9(d) as an example, the summarized feedback of the initial model is "RA-NC-NG-NW." Based on the built-in priority in Table 2, the search process in DREAM searches for "block type=xception" and "trainable=True" actions first. These actions change the model architecture and the `trainable` hyperparameter and significantly improve the model performance from 21.49% to 74.19% in the second trial, and the summarized conditions of the feedback turn to "XA-NC-NG-NW." Then DREAM tries to change the value of `imagenet size` hyperparameter and applies the actions that change the AdamW optimizer configuration, increasing the search score to 77.67%. At this time, the searches with random priorities attempt to search for actions such as adjusting the learning rate to 0.01, but only slightly improving the scores, no more than 50%. Subsequently, DREAM continues to search for actions such as changing the model architecture to *EfficientNet* and finally reaches 83.60% accuracy in the 18th trial. In contrast, the random priority-driven search with the best performance (i.e., "Random C" in Figure 9(d)) only obtains the best score of 56.54% by adjusting the learning rate of Adam optimizer to 0.001.

Answer to RQ4: The priority of the search actions has a significant impact on the search process in DREAM, and inappropriate priority can degrade the search results. The built-in search priority in DREAM which is evaluated and summarized from large-scale experiments can reasonably select actions in the search space based on the current conditions. Therefore, the search process in DREAM can conduct effective and efficient searches on the given tasks.

6 Related Work

In this article, we provide a search method based on the internal feedback and search action distribution to repair the shortcomings in AutoML pipeline and conduct an effective search on the given tasks. It is highly related to AutoML, debugging and optimizing AutoML, and DNN training and optimization.

AutoML. For ML tasks, researchers propose various tools, such as Auto-WEKA [99], hyperopt-sklearn [64] and autosklearn [1, 40, 42]. Their optimization methods mainly contain Bayesian optimization and evolutionary algorithms [41, 93]. These works mainly focus on traditional ML models while DREAM focuses on searching DNN models. With the development of DNN, Common NAS algorithms include Bayesian optimization [37, 57, 59, 84, 106, 112], deep **Reinforcement Learning (RL)** [22, 87, 115, 116], evolutionary algorithms [46, 90, 96, 108], and gradient-based methods [28, 39, 48, 73, 77, 107]. HPO algorithms mainly based on Bayesian Optimization method [23, 24], gradient-based method [32, 86]. Based on the existing optimization methods on searching models, the open source AutoML pipelines for DNN are implemented, such as AutoNet [80], Auto-Pytorch [114], Microsoft NNI [15], and AutoKeras [13]. The most popular among them are Microsoft NNI and AutoKeras. NNI provides an AutoML toolkit for practitioners with certain expertise, including NAS and model compression AutoKeras is based on TensorFlow and implements 3 search algorithms (i.e., Greedy, Bayesian [57], Hyperband [71]) besides Random search. It provides a simple operation interface and implements a highly automated model search pipeline, making it easy for users without any professional knowledge to use. In this article, we choose to compare DREAM with AutoKeras because it provides a highly automated model search pipeline and an easy-to-use interface. DREAM is currently implemented on top of TensorFlow and AutoKeras. How to generalize and implement the search method of DREAM in other pipelines will be one of our future directions.

Debugging and Optimizing AutoML. Researchers have comprehensively studied the application of AutoML pipelines and systems as well as the problems and challenges faced therein [25, 29, 36, 100, 104]. Wang et al. [104] conduct an in-depth empirical study on a series of AutoML systems, such as AutoKeras and NNI, and build a taxonomy of challenges they faced, including search space customization, model reproduction, and data adaption. To overcome these challenges and optimize the AutoML pipelines, researchers have proposed a variety of methods [31, 43, 84, 92]. Nguyen et al. [84] propose a novel NAS method to mine and select the initial models from the GitHub repositories. In contrast, DREAM focuses on selecting the suitable search action to improve the current optimal model based on the training feedback. These two approaches are orthogonal and have the potential to be used in combination.

DNN Training and Optimization. To improve DNN training efficiency and performance, researchers have proposed various training strategies [30, 75], optimization methods [68, 82], etc. Lai et al. [68] propose a novel method that converts the weights of a trained model to initialize and warm up the training of a new model, thereby accelerating model convergence and reducing the total amount of training needed. This solution has the potential to work with the AutoML pipelines to accelerate the training of each search trial. In addition, the researchers also propose modifying the given model's activation, initializer, and architecture to improve model performance and results [49, 61, 77, 89]. The existing work has provided valuable inspiration for the search space expansion of DREAM. Luo et al. [77] use an encoder to embed/map the neural network architecture into a continuous space and then perform gradient-based optimization to find the optimal architecture. Their approach focuses on searching for the model architecture rather than building a comprehensive pipeline for both model architectures and hyperparameters like NNI and AutoKeras.

7 Discussion

AutoML Pipelines' Shortcomings. ❶ DREAM focuses on ineffective search and performance issues, which are common and serious shortcomings in the AutoML pipelines. In addition to the NNI and AutoKeras, other AutoML pipelines such as Auto-Pytorch [114] and AutoSklearn [1] also have issue reports related to inefficient search and failure to improve models [11, 12]. As we have discussed in Section 3.2, the root cause of these issues lies in the shortcomings in pipeline design, that is, the neglect of valuable search space and useful feedback from the training procedure. Existing pipelines (e.g., Auto-Pytorch and AutoKeras) generally use the basic structure in Figure 1 and leverage simple feedback (i.e., score) to guide the search of model architecture and hyperparameters. Moreover, they rely on random walks or statistical models in searches, which makes it difficult to effectively predict optimal search actions to improve models. To fix the AutoML shortcomings and conduct effective searches, DREAM has designed repair mechanisms on the top of the basic stages of AutoML pipelines, namely search space expansion, feedback-driven search, and feedback monitoring, which can be universally applied to different pipelines. ❷ Existing AutoML pipelines have other shortcomings, such as the lack of optimization and support for domain-specific tasks [104], which limits their usability. Specifically, some AutoML pipelines (e.g., NNI [15], TPOT [69]) require users to define the search space and the modules in the pipeline for downstream tasks. Such a design requires users to have certain knowledge of DL techniques, which violates the design intention of the AutoML pipeline. How to further improve the usability of AutoML systems, such as automatically generating search space based on the given dataset and task, will be a potential future direction.

DREAM Enhancement. ❶ DREAM currently supports repairing NNI and AutoKeras, which are two of the most popular AutoML pipelines. Note that the mechanisms implemented in DREAM are built on the AutoML basic pipeline rather than designed for a specific pipeline. How to extend DREAM to support more AutoML systems will be our future work. ❷ To improve the AutoML search results, DREAM focuses on the search space and search strategy and designs improvement mechanisms. Researchers have pointed out that optimization in model training, such as using the weights of existing models to warm up the new model training [68], is of great help to model performance, which has been overlooked in AutoML pipelines. How to integrate the existing model training optimization scheme into DREAM and further improve the search performance will be one potential enhancement direction of DREAM. ❸ Based on prior work [20, 53, 76, 81, 113], DREAM has designed 14 conditions from four angles to summarize and describe current model training, which roughly covers various situations encountered by the model during searches. More diverse angles and detailed conditions can help calculate and design corresponding priorities and more effectively guide the search. Expanding the conditions and refining the search priority will be a future direction. Note that adding new conditions does not require restarting the entire experiment in Section 7. It only requires obtaining several models with new conditions, then applying different search actions to modify the model and recording and measuring the impact on the training results (e.g., score) of the modified models.

Another Strategy in DREAM. DREAM utilizes the search space expansion and feedback-driven search to conduct effective searches based on greedy methods. These mechanisms in DREAM can also be applied to other search methods, such as beam search, which is a common heuristic greedy search strategy and widely used in NLP and speech recognition models to choose the best output [79, 95]. It builds the search tree with a breadth-first search over the search space and generates all successors of the states at each level of the tree and sorts them by their performance. The number of the stored states is equal to the width of the beam, which means that the larger the beam width, the fewer states to prune and the higher the search overhead. When the beam width is 1, the search process will be equivalent to the greedy search. To understand the actual effect of beam search as an alternative search method, we construct a comparative experiment on the CIFAR-100 dataset. The beam width is set to 3. We implement two kinds of beam searches in the experiments, namely a search trained directly on the full dataset without the subset and a search trained in two steps. For the latter, in the first step, we train nine searched models with the highest priorities, which are generated from the previous search, on a 10% subset of the CIFAR-100 dataset. Then in the second step, the three models with the best performance in the first step are reserved for the complete training. All hyperparameters in the models in searches have a chance of 0.01 to randomly mutate. We observe that the search performance of greedy search usually outperforms two-step beam search and complete beam search. Our analysis shows that the root cause of the performance difference is that the beam search trains more models, and most of them help little in improving the search performance. This results in that the search efficiency of beam search is worse than that of greedy search in the case of the same search space and search priorities. Even for the search that most of the models are trained on the subset, the time spent on beam search to reach 80%

accuracy is still much larger (e.g., nearly 6 hours) than the time spent on greedy search due to the time overhead of training additional models. Therefore, for the consideration of efficiency and effectiveness in search, we select the greedy method as the default strategy of DREAM.

DREAM and RL. RL is a family of ML that focuses on how to take actions based on the environment to maximize rewards [55, 58]. RL has been widely used in software engineering and software testing [21, 103]. The overall process of RL can be represented as follows. Given a state S in the environment, an agent selects an action a based on a state transition $P_a(S, S')$ and enters a new state S', receiving the reward r. The optimization problem is to find a reward model that maximizes the cumulative reward during the process. The high-level approach of DREAM is similar to RL. The best model in the search history of the AutoML pipeline and its corresponding feedback conditions can be considered as the state S. DREAM implements search space expansion and feedback-based search to select the search action a for S and obtain a new model and its corresponding training feedback S'. The reward a is the score improvement of the new model relative to the historical best model. DREAM utilizes pre-calculated search priorities as the reward model to guide the search and maximize the accumulation of rewards in the search process. Therefore, how to obtain and calculate search priority is vital for DREAM. As shown in Section 5.4, ineffective search priority can lead to poor search performance, with the search process failing to improve after dozens of search trials. We recommend that users follow to evaluate each search action on diverse datasets before deploying DREAM. In addition, RL techniques provide potential enhancement directions for DREAM. Currently, the search priorities in DREAM do not change based on rewards during the search process, and the greedy search it implemented emphasizes short-term rewards. Advanced RL methods (e.g., Q-learning [19, 105]) provide a feasible direction to dynamically adjust search priorities based on the given dataset during the search and further improve DREAM's performance.

8 Conclusion

This article presents DREAM, an automatic repair system for AutoML pipelines shortcomings, namely the performance issue and ineffective search issue. DREAM collects detailed feedback from the model training and evaluation. Then it repairs the shortcomings by expanding the search space and conducting a feedback-driven search. Our evaluation results on six image and text datasets and two AutoML pipelines show that DREAM can effectively and efficiently fix AutoML shortcomings, and the repaired search obtains better results than state-of-the-art AutoML pipelines.

Acknowledgment

The authors would like to thank the anonymous reviewers for their insightful comments and valuable suggestions.

References

[1] GitHub. 2020. Auto-Sklearn. Retrieved from https://github.com/automl/auto-sklearn

[2] IMDb. 2020. IMDb Datasets. Retrieved from https://www.imdb.com/interfaces/

[3] Precedence Research. 2022. Artificial Intelligence (AI) Software Market - Global Industry Analysis, Size, Share, Growth, Trends, Regional Outlook, and Forecast 2023–2032. Retrieved from https://www.precedenceresearch.com/artificial-intelligence-software-market

[4] CIFAR. 2022. CIFAR-100 Dataset. Retrieved from https://www.cs.toronto.edu/kriz/cifar.html

[5] Google Cloud. 2022. Google Cloud, Harvard Global Health Institute release improved COVID-19 Public Forecasts, share lessons learned. Retrieved from https://cloud.google.com/blog/products/ai-machine-learning/google-and-harvard-improve-covid-19-forecasts

[6] Keras. 2022. Image classification via fine-tuning with EfficientNet. Retrieved from https://keras.io/examples/vision/image_classification_efficientnet_fine_tuning

[7] Google Cloud. 2022. Imagia: Collaborating on AI-powered healthcare discovery. Retrieved from https://cloud.google.com/customers/imagia

[8] Google Cloud. 2022. Meredith Digital: Entering a new era of data-driven publishing. Retrieved from https://cloud.google.com/customers/meredith

[9] Microsoft. 2022. Providence creates chatbot to improve COVID-19 care and manage hospital resources. Retrieved from https://customers.microsoft.com/en-in/story/1402681562485712847-providence-health-provider-azure-en-united-states

[10] TensorFlow. 2022. Transfer learning and fine-tuning. Retrieved from https://www.tensorflow.org/guide/keras/transfer_learning

[11] GitHub. 2023. AutoPytorch selects only Dummy model. Retrieved from https://github.com/automl/Auto-PyTorch/issues/494

[12] GitHub. 2023. Long refit time of AutoSklearn models. Retrieved from https://github.com/automl/auto-sklearn/issues/1683

[13] GitHub. 2024. AutoKeras. Retrieved from https://github.com/keras-team/autokeras

[14] Google Cloud. 2024. Google Cloud AutoML. Retrieved from https://cloud.google.com/automl

[15] Microsoft. 2024. Microsoft NNI. Retrieved from https://github.com/microsoft/nni

[16] GitHub. 2024. Our repository with experiment data and the prototype of our system. Retrieved from https://github.com/shiningrain/DREAM

[17] David D. Lewis. 2024. Reuters-21578 dataset. Retrieved from http://www.daviddlewis.com/resources/testcollections/reuters21578/

[18] Martín Abadi, Paul Barham, Jianmin Chen, Zhifeng Chen, Andy Davis, Jeffrey Dean, Matthieu Devin, Sanjay Ghemawat, Geoffrey Irving, Michael Isard, et al. 2016. TensorFlow: A system for large-scale machine learning. In *Proceedings of the 12th* {USENIX} Symposium on Operating Systems Design and Implementation ({OSDI} '16), 265–283.

[19] Mohammed Akour, Mamdouh Alenezi, and Osama Alqasem. 2024. Enhancing software fault detection with deep reinforcement learning: A Q-learning approach. In *Proceedings of the 13th International Conference on Software and Computer Applications*, 97–101.

[20] Isac Arnekvist, J. Frederico Carvalho, Danica Kragic, and Johannes A. Stork. 2020. The effect of target normalization and momentum on dying ReLU. arXiv:2005.06195. Retrieved from https://arxiv.org/abs/2005.06195

[21] Mojtaba Bagherzadeh, Nafiseh Kahani, and Lionel Briand. 2021. Reinforcement learning for test case prioritization. *IEEE Transactions on Software Engineering* 48, 8 (2021), 2836–2856.

[22] Bowen Baker, Otkrist Gupta, Nikhil Naik, and Ramesh Raskar. 2017. Designing neural network architectures using reinforcement learning. In *Proceedings of the 5th International Conference on Learning Representations (ICLR '17)*. OpenReview.net. Retrieved from https://openreview.net/forum?id=S1c2cvqee

[23] James Bergstra, Rémi Bardenet, Yoshua Bengio, and Balázs Kégl. 2011. Algorithms for hyper-parameter optimization. In *Proceedings of the 24th International Conference on Neural Information Processing Systems*.

[24] James Bergstra, Dan Yamins, and David D. Cox. 2013. Hyperopt: A python library for optimizing the hyperparameters of machine learning algorithms. In *Proceedings of the 12th Python in Science Conference*, Vol. 13. Citeseer, 20.

[25] Sumon Biswas, Mohammad Wardat, and Hridesh Rajan. 2022. The art and practice of data science pipelines: A comprehensive study of data science pipelines in theory, in-the-small, and in-the-large. In *Proceedings of the 44th International Conference on Software Engineering*, 2091–2103.

[26] Lukas Bossard, Matthieu Guillaumin, and Luc Van Gool. 2014. Food-101–mining discriminative components with random forests. In *Proceedings of the European Conference on Computer Vision*. Springer, 446–461.

[27] Peter F. Brown, Vincent J. Della Pietra, Peter V. Desouza, Jennifer C. Lai, and Robert L. Mercer. 1992. Class-based n-gram models of natural language. *Computational Linguistics* 18, 4 (1992), 467–480.

[28] Han Cai, Ligeng Zhu, and Song Han. 2019. ProxylessNAS: Direct neural architecture search on target task and hardware. In *Proceedings of the 7th International Conference on Learning Representations (ICLR '19)*. OpenReview.net. Retrieved from https://openreview.net/forum?id=HylVB3AqYm

[29] Fabio Calefato, Luigi Quaranta, Filippo Lanubile, and Marcos Kalinowski. 2023. Assessing the use of AutoML for data-driven software engineering. In *Proceedings of the ACM/IEEE International Symposium on Empirical Software Engineering and Measurement (ESEM)*. IEEE, 1–12.

[30] Abdullah Caliskan, Mehmet Emin Yuksel, Hasan Badem, and Alper Basturk. 2018. Performance improvement of deep neural network classifiers by a simple training strategy. *Engineering Applications of Artificial Intelligence* 67 (2018), 14–23.

[31] José P. Cambronero, Jürgen Cito, and Martin C. Rinard. 2020. Ams: Generating automl search spaces from weak specifications. In *Proceedings of the 28th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering*, 763–774.

[32] Kartik Chandra, Erik Meijer, Samantha Andow, Emilio Arroyo-Fang, Irene Dea, Johann George, Melissa Grueter, Basil Hosmer, Steffi Stumpos, Alanna Tempest, et al. 2019. Gradient descent: The ultimate optimizer. arXiv:1909.13371. Retrieved from https://arxiv.org/abs/1909.13371

[33] François Chollet. 2017. Xception: Deep learning with depthwise separable convolutions. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 1251–1258.

[34] Francois Chollet. 2021. *Deep Learning with Python*. Simon and Schuster.

[35] Adam Coates, Andrew Ng, and Honglak Lee. 2011. An analysis of single-layer networks in unsupervised feature learning. In *Proceedings of the 14th International Conference on Artificial Intelligence and Statistics*, 215–223.

[36] Anamaria Crisan and Brittany Fiore-Gartland. 2021. Fits and starts: Enterprise use of automl and the role of humans in the loop. In *Proceedings of the CHI Conference on Human Factors in Computing Systems*, 1–15.

[37] Georgi Dikov and Justin Bayer. 2019. Bayesian learning of neural network architectures. In *Proceedings of the 22nd International Conference on Artificial Intelligence and Statistics.* PMLR, 730–738.

[38] Tobias Domhan, Jost Tobias Springenberg, and Frank Hutter. 2015. Speeding up automatic hyperparameter optimization of deep neural networks by extrapolation of learning curves. In *Proceedings of the 24th International Joint Conference on Artificial Intelligence*.

[39] Xuanyi Dong and Yi Yang. 2019. Searching for a robust neural architecture in four gpu hours. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 1761–1770.

[40] Matthias Feurer, Katharina Eggensperger, Stefan Falkner, Marius Lindauer, and Frank Hutter. 2020. Auto-Sklearn 2.0: Hands-free AutoML via Meta-Learning. arXiv:2007.04074. Retrieved from https://arxiv.org/abs/2007.04074

[41] Matthias Feurer and Frank Hutter. 2019. Hyperparameter optimization. In *Automated Machine Learning*. Springer, Cham, 3–33.

[42] Matthias Feurer, Aaron Klein, Katharina Eggensperger, Jost Tobias Springenberg, Manuel Blum, and Frank Hutter. 2019. Auto-sklearn: Efficient and robust automated machine learning. In *Automated Machine Learning*. Springer, Cham, 113–134.

[43] Yanjie Gao, Yonghao Zhu, Hongyu Zhang, Haoxiang Lin, and Mao Yang. 2021. Resource-guided configuration space reduction for deep learning models. In *Proceedings of the IEEE/ACM 43rd International Conference on Software Engineering (ICSE).* IEEE, 175–187.

[44] Rong Ge, Sham M Kakade, Rahul Kidambi, and Praneeth Netrapalli. 2019. The step decay schedule: A near optimal, geometrically decaying learning rate procedure for least squares. In *Proceedings of the 33rd International Conference on Neural Information Processing Systems*.

[45] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. 2016. *Deep Learning*. MIT Press.

[46] Zichao Guo, Xiangyu Zhang, Haoyuan Mu, Wen Heng, Zechun Liu, Yichen Wei, and Jian Sun. 2020. Single path one-shot neural architecture search with uniform sampling. In *Proceedings of the European Conference on Computer Vision*. Springer, 544–560.

[47] Isabelle Guyon and André Elisseeff. 2006. An introduction to feature extraction. In *Feature Extraction*. Springer, 1–25.

[48] Chaoyang He, Haishan Ye, Li Shen, and Tong Zhang. 2020. Milenas: Efficient neural architecture search via mixed-level reformulation. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 11993–12002.

[49] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. 2015. Delving deep into rectifiers: Surpassing human-level performance on imagenet classification. In *Proceedings of the IEEE International Conference on Computer Vision*, 1026–1034.

[50] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. 2016. Deep residual learning for image recognition. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 770–778.

[51] Xin He, Kaiyong Zhao, and Xiaowen Chu. 2021. AutoML: A survey of the state-of-the-art. *Knowledge-Based Systems* 212 (2021), 106622.

[52] Daniel Ho, Eric Liang, Xi Chen, Ion Stoica, and Pieter Abbeel. 2019. Population based augmentation: Efficient learning of augmentation policy schedules. In *International Conference on Machine Learning*. PMLR, 2731–2741.

[53] Sepp Hochreiter. 1991. Untersuchungen zu dynamischen neuronalen Netzen. *Diploma, Technische Universität München* 91, 1 (1991).

[54] Mei-Ling Huang, Yung-Hsiang Hung, W. M. Lee, Rong-Kwei Li, and Bo-Ru Jiang. 2014. SVM-RFE based feature selection and Taguchi parameters optimization for multiclass SVM classifier. *The Scientific World Journal* 2014 (2014), 1–1. Retrieved from https://onlinelibrary.wiley.com/doi/epdf/10.1155/2014/795624

[55] Yesmina Jaafra, Jean Luc Laurent, Aline Deruyver, and Mohamed Saber Naceur. 2019. Reinforcement learning for neural architecture search: A review. *Image and Vision Computing* 89 (2019), 57–66.

[56] M. Z. H. Jesmeen, J. Hossen, S. Sayeed, C. K. Ho, K. Tawsif, Armanur Rahman, and E. Arif. 2018. A survey on cleaning dirty data using machine learning paradigm for big data analytics. *Indonesian Journal of Electrical Engineering and Computer Science* 10, 3 (2018), 1234–1243.

[57] Haifeng Jin, Qingquan Song, and Xia Hu. 2019. Auto-Keras: An efficient neural architecture search system. In *Proceedings of the 25th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*. ACM, New York, NY, 1946–1956.

[58] Leslie Pack Kaelbling, Michael L. Littman, and Andrew W. Moore. 1996. Reinforcement learning: A survey. *Journal of Artificial Intelligence Research* 4 (1996), 237–285.

[59] Kirthevasan Kandasamy, Willie Neiswanger, Jeff Schneider, Barnabás Póczos, and Eric P. Xing. 2018. Neural architecture search with Bayesian optimisation and optimal transport. In *Proceedings of the Annual Conference on Neural Information Processing Systems*. Samy Bengio, Hanna M. Wallach, Hugo Larochelle, Kristen Grauman, Nicolò Cesa-Bianchi, and Roman Garnett (Eds.), 2020–2029. Retrieved from https://proceedings.neurips.cc/paper/2018/hash/f33ba15effa5c10e873bf3842afb46a6-Abstract.html

[60] Samina Khalid, Tehmina Khalil, and Shamila Nasreen. 2014. A survey of feature selection and feature extraction techniques in machine learning. In *Proceedings of the Science and Information Conference*. IEEE, 372–378.

[61] Günter Klambauer, Thomas Unterthiner, Andreas Mayr, and Sepp Hochreiter. 2017. Self-normalizing neural networks. In *Proceedings of the 31st International Conference on Neural Information Processing Systems*, 971–980.

[62] Aaron Klein, Stefan Falkner, Simon Bartels, Philipp Hennig, and Frank Hutter. 2017. Fast bayesian optimization of machine learning hyperparameters on large datasets. In *Artificial Intelligence and Statistics*. PMLR, 528–536.

[63] Aaron Klein, Stefan Falkner, Jost Tobias Springenberg, and Frank Hutter. 2016. Learning curve prediction with Bayesian neural networks. In *Proceedings of the International Conference on Learning Representations*.

[64] Brent Komer, James Bergstra, and Chris Eliasmith. 2019. Hyperopt-sklearn. In *Automated Machine Learning*. Springer, Cham, 97–111.

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

[66] Alex Krizhevsky and Geoffrey Hinton. 2009. *Learning Multiple Layers of Features from Tiny Images*. Master's thesis, University of Tront.

[67] Alexandre Lacoste, Alexandra Luccioni, Victor Schmidt, and Thomas Dandres. 2019. Quantifying the carbon emissions of machine learning. arXiv:1910.09700. Retrieved from https://arxiv.org/abs/1910.09700

[68] Fan Lai, Yinwei Dai, Harsha V. Madhyastha, and Mosharaf Chowdhury. 2023. {ModelKeeper}: Accelerating {DNN} training via automated training warmup. In *Proceedings of the 20th USENIX Symposium on Networked Systems Design and Implementation (NSDI '23)*, 769–785.

[69] Trang T. Le, Weixuan Fu, and Jason H. Moore. 2020. Scaling tree-based automated machine learning to biomedical big data with a feature set selector. *Bioinformatics* 36, 1 (2020), 250–256.

[70] Ya Le and Xuan Yang. 2015. Tiny imagenet visual recognition challenge. *CS 231N* 7, 7 (2015), 3.

[71] Lisha Li, Kevin Jamieson, Giulia DeSalvo, Afshin Rostamizadeh, and Ameet Talwalkar. 2017. Hyperband: A novel bandit-based approach to hyperparameter optimization. *The Journal of Machine Learning Research* 18, 1 (2017), 6765–6816.

[72] Xin Li and Dan Roth. 2002. Learning question classifiers. In *Proceedings of the 19th International Conference on Computational Linguistics (COLING '02)*. Retrieved from https://www.aclweb.org/anthology/C02-1150

[73] Hanxiao Liu, Karen Simonyan, and Yiming Yang. 2018. Darts: Differentiable architecture search. In *Proceedings of the 7th International Conference on Learning Representations (ICLR '19)*. OpenReview.net. Retrieved from https://openreview.net/forum?id=S1eYHoC5FX

[74] Ilya Loshchilov and Frank Hutter. 2019. Decoupled weight decay regularization. In *Proceedings of the 7th International Conference on Learning Representations (ICLR '19)*. OpenReview.net. Retrieved from https://openreview.net/forum?id=Bkg6RiCqY7

[75] Jing Lu, Gustavo Hernandez Abrego, Ji Ma, Jianmo Ni, and Yinfei Yang. 2021. Multi-stage training with improved negative contrast for neural passage retrieval. In *Proceedings of the 2021 Conference on Empirical Methods in Natural Language Processing*, 6091–6103.

[76] Lu Lu, Yeonjong Shin, Yanhui Su, and George Em Karniadakis. 2020. Dying ReLU and initialization: Theory and numerical examples. *Communications in Computational Physics* 28, 5 (2020), 1671–1706.

[77] Renqian Luo, Fei Tian, Tao Qin, Enhong Chen, and Tie-Yan Liu. 2018. Neural architecture optimization. In *Proceedings of the 32nd International Conference on Neural Information Processing Systems*, 7816–7827.

[78] Shiqing Ma, Yingqi Liu, Wen-Chuan Lee, Xiangyu Zhang, and Ananth Grama. 2018. MODE: Automated neural network model debugging via state differential analysis and input selection. In *Proceedings of the 26th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering (ESEC/FSE)*. ACM, New York, NY, 175–186.

[79] Clara Meister, Tim Vieira, and Ryan Cotterell. 2020. Best-first beam search. *Transactions of the Association for Computational Linguistics* 8 (2020), 795–809.

[80] Hector Mendoza, Aaron Klein, Matthias Feurer, Jost Tobias Springenberg, Matthias Urban, Michael Burkart, Maximilian Dippel, Marius Lindauer, and Frank Hutter. 2019. Towards automatically-tuned deep neural networks. In *Automated Machine Learning*. Springer, Cham, 135–149.

[81] John Miller and Moritz Hardt. 2018. Stable recurrent models. In *Proceedings of the 7th International Conference on Learning Representations (ICLR '19)*. OpenReview.net. Retrieved from https://openreview.net/forum?id=Hygxb2CqKm

[82] Deepak Narayanan, Aaron Harlap, Amar Phanishayee, Vivek Seshadri, Nikhil R Devanur, Gregory R. Ganger, Phillip B. Gibbons, and Matei Zaharia. 2019. PipeDream: Generalized pipeline parallelism for DNN training. In *Proceedings of the 27th ACM Symposium on Operating Systems Principles*, 1–15.

[83] Renato Negrinho, Matthew Gormley, Geoffrey J. Gordon, Darshan Patil, Nghia Le, and Daniel Ferreira. 2019. Towards modular and programmable architecture search. In *Proceedings of the 33rd International Conference on Neural Information Processing Systems*, 13733–13743.

[84] Giang Nguyen, Md Johirul Islam, Rangeet Pan, and Hridesh Rajan. 2022. Manas: Mining software repositories to assist automl. In *Proceedings of the 44th International Conference on Software Engineering*. 1368–1380.

[85] Jieun Park, Dokkyun Yi, and Sangmin Ji. 2020. A novel learning rate schedule in optimization for neural networks and it's convergence. *Symmetry* 12, 4 (2020), 660.

[86] Fabian Pedregosa. 2016. Hyperparameter optimization with approximate gradient. In *Proceedings of the International Conference on Machine Learning*. PMLR, 737–746.

[87] Hieu Pham, Melody Guan, Barret Zoph, Quoc Le, and Jeff Dean. 2018. Efficient neural architecture search via parameters sharing. In *International Conference on Machine Learning*. PMLR, 4095–4104.

[88] Ning Qian. 1999. On the momentum term in gradient descent learning algorithms. *Neural Networks* 12, 1 (1999), 145–151.

[89] Andrinandrasana David Rasamoelina, Fouzia Adjailia, and Peter Sinčák. 2020. A review of activation function for artificial neural network. In *Proceedings of the IEEE 18th World Symposium on Applied Machine Intelligence and Informatics (SAMI)*. IEEE, 281–286.

[90] Esteban Real, Sherry Moore, Andrew Selle, Saurabh Saxena, Yutaka Leon Suematsu, Jie Tan, Quoc V. Le, and Alexey Kurakin. 2017. Large-scale evolution of image classifiers. In *Proceedings of the International Conference on Machine Learning*. PMLR, 2902–2911.

[91] Yuji Roh, Geon Heo, and Steven Euijong Whang. 2019. A survey on data collection for machine learning: A big data-ai integration perspective. *IEEE Transactions on Knowledge and Data Engineering* 33, 4 (2019), 1328–1347.

[92] Ripon K. Saha, Akira Ura, Sonal Mahajan, Chenguang Zhu, Linyi Li, Yang Hu, Hiroaki Yoshida, Sarfraz Khurshid, and Mukul R. Prasad. 2022. SapientML: Synthesizing machine learning pipelines by learning from human-writen solutions. In *Proceedings of the 44th International Conference on Software Engineering*, 1932–1944.

[93] Bobak Shahriari, Kevin Swersky, Ziyu Wang, Ryan P. Adams, and Nando De Freitas. 2015. Taking the human out of the loop: A review of Bayesian optimization. *Proceedings of the IEEE* 104, 1 (2015), 148–175.

[94] Ahnjae Shin, Do Yoon Kim, Joo Seong Jeong, and Byung-Gon Chun. 2020. Hippo: Taming hyper-parameter optimization of deep learning with stage trees. arXiv:2006.11972. Retrieved from https://arxiv.org/abs/2006.11972

[95] Felix Stahlberg and Bill Byrne. 2019. On NMT search errors and model errors: Cat got your tongue?. In *Proceedings of the Conference on Empirical Methods in Natural Language Processing and the 9th International Joint Conference on Natural Language Processing (EMNLP-IJCNLP)*, 3356–3362.

[96] Masanori Suganuma, Shinichi Shirakawa, and Tomoharu Nagao. 2017. A genetic programming approach to designing convolutional neural network architectures. In *Proceedings of the Genetic and Evolutionary Computation Conference*. 497–504.

[97] David Sussillo and L. F. Abbott. 2014. Random walk initialization for training very deep feedforward networks. arXiv:1412.6558. Retrieved from https://arxiv.org/abs/1412.6558

[98] Mingxing Tan and Quoc Le. 2019. Efficientnet: Rethinking model scaling for convolutional neural networks. In *Proceedings of the International Conference on Machine Learning*. PMLR, 6105–6114.

[99] Chris Thornton, Frank Hutter, Holger H. Hoos, and Kevin Leyton-Brown. 2013. Auto-WEKA: Combined selection and hyperparameter optimization of classification algorithms. In *Proceedings of the 19th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, 847–855.

[100] Koen Van der Blom, Alex Serban, Holger Hoos, and Joost Visser. 2021. AutoML adoption in ML software. In *Proceedings of the 8th ICML Workshop on Automated Machine Learning (AutoML)*.

[101] Ashish Vaswani. 2017. Attention is all you need. arXiv:1706.03762. Retrieved from https://arxiv.org/abs/1706.03762

[102] Andreas Veit, Michael J. Wilber, and Serge Belongie. 2016. Residual networks behave like ensembles of relatively shallow networks. In *Proceedings of the 30th International Conference on Neural Information Processing Systems*, 550–558.

[103] Yao Wan, Zhou Zhao, Min Yang, Guandong Xu, Haochao Ying, Jian Wu, and Philip S. Yu. 2018. Improving automatic source code summarization via deep reinforcement learning. In *Proceedings of the 33rd ACM/IEEE International Conference on Automated Software Engineering*, 397–407.

[104] Chao Wang, Zhenpeng Chen, and Minghui Zhou. 2023. Automl from software engineering perspective: Landscapes and challenges. In *Proceedings of the IEEE/ACM 20th International Conference on Mining Software Repositories (MSR)*. IEEE, 39–51.

[105] Christopher J. C. H. Watkins and Peter Dayan. 1992. Q-learning. *Machine Learning* 8 (1992), 279–292.
[106] Colin White, Willie Neiswanger, and Yash Savani. 2021. BANANAS: Bayesian optimization with neural architectures for neural architecture search. In *Proceedings of the 35th AAAI Conference on Artificial Intelligence (AAAI '21), 33rd Conference on Innovative Applications of Artificial Intelligence (IAAI '21), The 11th Symposium on Educational Advances in Artificial Intelligence (EAAI '21).* AAAI Press, 10293–10301. DOI: https://doi.org/10.1609/AAAI.V35I12.17233
[107] Bichen Wu, Yanghan Wang, Peizhao Zhang, Yuandong Tian, Peter Vajda, and Kurt Keutzer. 2018. Mixed precision quantization of convnets via differentiable neural architecture search. arXiv:1812.00090. Retrieved from https://arxiv.org/abs/1812.00090
[108] Lingxi Xie and Alan Yuille. 2017. Genetic cnn. In *Proceedings of the IEEE International Conference on Computer Vision*, 1379–1388.
[109] Saining Xie, Alexander Kirillov, Ross Girshick, and Kaiming He. 2019. Exploring randomly wired neural networks for image recognition. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, 1284–1293.
[110] Ziang Xie, Sida I. Wang, Jiwei Li, Daniel Lévy, Aiming Nie, Dan Jurafsky, and Andrew Y. Ng. 2017. Data noising as smoothing in neural network language models. In *Proceedings of the 5th International Conference on Learning Representations (ICLR '17).* OpenReview.net. Retrieved from https://openreview.net/forum?id=H1VyHY9gg
[111] Jufeng Yang, Xiaoxiao Sun, Yu-Kun Lai, Liang Zheng, and Ming-Ming Cheng. 2018. Recognition from web data: A progressive filtering approach. *IEEE Transactions on Image Processing* 27, 11 (2018), 5303–5315.
[112] Arber Zela, Aaron Klein, Stefan Falkner, and Frank Hutter. 2018. Towards automated deep learning: Efficient joint neural architecture and hyperparameter search. arXiv:1807.06906. Retrieved from https://arxiv.org/abs/1807.06906
[113] Xiaoyu Zhang, Juan Zhai, Shiqing Ma, and Chao Shen. 2021. AUTOTRAINER: An automatic DNN training problem detection and repair system. In *Proceedings of the IEEE/ACM 43rd International Conference on Software Engineering (ICSE).* IEEE, 359–371.
[114] Lucas Zimmer, Marius Lindauer, and Frank Hutter. 2021. Auto-Pytorch: Multi-fidelity metalearning for efficient and robust AutoDL. *IEEE Transactions on Pattern Analysis and Machine Intelligence* (2021).
[115] Barret Zoph and Quoc V. Le. 2017. Neural Architecture Search with Reinforcement Learning. In *Proceedings of the 5th International Conference on Learning Representations (ICLR '17).* OpenReview.net. Retrieved from https://openreview.net/forum?id=r1Ue8Hcxg
[116] Barret Zoph, Vijay Vasudevan, Jonathon Shlens, and Quoc V. Le. 2018. Learning transferable architectures for scalable image recognition. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 8697–8710.

Received 20 April 2024; revised 19 October 2024; accepted 23 October 2024