FLAML: A FAST AND LIGHTWEIGHT AUTOML LIBRARY

Chi Wang [*1] **Qingyun Wu** [*1] **Markus Weimer** [1] **Erkang Zhu** [1]

ABSTRACT

We study the problem of using low computational cost to automate the choices of learners and hyperparameters for an ad-hoc training dataset and error metric, by conducting trials of different configurations on the given training data. We investigate the joint impact of multiple factors on both trial cost and model error, and propose several design guidelines. Following them, we build a fast and lightweight library FLAML which optimizes for low computational resource in finding accurate models. FLAML integrates several simple but effective search strategies into an adaptive system. It significantly outperforms top-ranked AutoML libraries on a large open source AutoML benchmark under equal, or sometimes orders of magnitude smaller budget constraints.

1 INTRODUCTION

It is predicted that in the next 10 years, hundreds of thousands of small teams will build millions of ML-infused applications – most just moderately remunerative, but with huge collective value (Agrawal et al., 2020). Operating by large teams of ML experts and running on massive dedicated infrastructures is not well justified for these new applications. That motivates fast and economical software solutions to Automated Machine Learning (AutoML): Given a training dataset and an error metric, use *low computational cost* to search for learner and hyperparameter choices and produce models optimizing the error metric in short time.

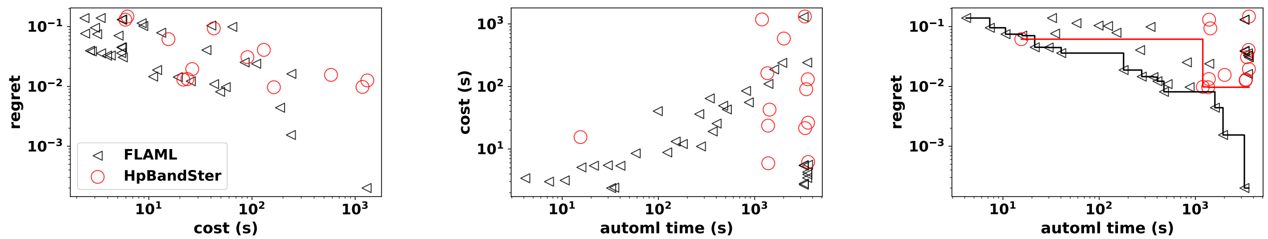
To provide a concrete context, let us consider the use of ML in database systems. The database community has grown an increasing interest of integrating data-driven decision making components fueled by machine learning techniques. For example, classification or regression models have been explored for indexing (Kraska et al., 2018; Galakatos et al., 2019), cardinality and selectivity estimation (Kipf et al., 2019; Dutt et al., 2019), query performance prediction (Marcus & Papaemmanouil, 2019), and workload forecasting (Ma et al., 2018). These models make predictions by learning from a large amount of labeled data, which are generated automatically by the system for each dataset or workload instance. For example, a selectivity estimation model can be built for each table or join expression using selectivity labels generated from synthetic queries or a given workload (Dutt et al., 2019; 2020), and the best model configurations vary per instance of training dataset. AutoML solutions for these applications are required to be fast and economic, as the system needs to select hyperparameters or learners frequently on different training data (for numerous tables, join expressions, and frequent updates), and continuously and timely deploy them (Renggli et al., 2019). Computational resource of the system is precious (e.g., for answering database queries), and only a small fraction can be allocated to AutoML, e.g., a few CPU minutes per selectivity estimation model.

A number of AutoML libraries have been developed, which usually involve multiple trials of different configurations. One drawback in existing solutions is they require long time or large amounts of resources to produce accurate models for large scale training datasets. For example, given one CPU hour, when tested on a recent large-scale AutoML benchmark (Gijsbers et al., 2019), the state-of-the-art solutions underperform a tuned random forest baseline on 36-51% of the tasks. And the ratio is even higher when the budget is smaller.

To address the problem systematically, it is desirable to factor the *trial cost*, i.e., the CPU cost of training and assessing the model error, explicitly in the AutoML problem. We recognize that the cost of one trial is jointly decided by the following variables: the choice of learner, a subset of the hyperparameters for the chosen learner, the size of the training data, and the resampling strategy. Those variables also affect the *trial error* (i.e., the assessed model error) jointly. Given an ad-hoc dataset, an AutoML solution that is only optimized for low trial error may invoke unnecessarily expensive trials, while a solution that is only optimized for low trial cost may keep making cheap but erroneous trials. Some hyperparameter optimization methods made an effort to balance the two objectives, but the scope is limited and most systems target resource-consuming clusters (Snoek

[*]Equal contribution [1]Microsoft Corporation, Redmond, WA, USA. Correspondence to: Chi Wang <wang.chi@microsoft.com>.

Proceedings of the 4th MLSys Conference, San Jose, CA, USA, 2021. Copyright 2021 by the author(s).



(a) Model auc regret vs. training cost for every trial

(b) Trial cost vs. the total time from start when each trial is finished

(c) Model auc regret vs. the total time from start when each trial is finished

Figure 1. Example of search performance for FLAML vs. a baseline in the same search space. Each marker corresponds to one trial of configuration evaluation in a particular method. Model auc regret=best_auc-model_auc. Each marker corresponds to one trial of configuration evaluation in a particular method. Subfigure (a) suggests that FLAML makes fewer expensive trials with high error (top right corner) than HpBandSter. Subfigure (b) further displays that the expense of trials made by FLAML grows gradually with total time spent, while for HpBandSter there is no such trend. As a result, subfigure (c) shows that FLAML outperforms in both early and late stages.

et al., 2012; Li et al., 2017; Falkner et al., 2018; Liaw et al., 2019; Li et al., 2020). No previous AutoML system handles the complex dependency among the multiple variables mentioned above. Though challenging, it is desired to have an economical system that holistically considers the multiple factors in the cost-error tradeoff, and handles different tasks robustly and efficiently.

We design and implement a lightweight Python library FLAML[1]. FLAML leverages the structure of the search space to choose a search order optimized for both cost and error. It iteratively decides the learner, hyperparameter, sample size and resampling strategy while leveraging their compound impact on both cost and error as the search proceeds. First, we analyze the relation of these factors and deduce desirable properties of an economical AutoML system. To satisfy these properties, we integrate several non-traditional search strategies judiciously because commonly employed strategies do not sufficiently exploit the analyzed relations of the multiple factors. Overall, the search tends to gradually move from cheap trials and inaccurate models to expensive trials and accurate models (a typical example is illustrated in Figure 1). FLAML is designed for robustly adapting to an ad-hoc dataset out of the box, without relying on expensive preparation such as meta-learning. In fact, our system has almost no computational overhead beyond the trial cost of each configuration.

We perform extensive evaluation using a recent open source AutoML benchmark (Gijsbers et al., 2019) plus regression datasets from a regression benchmark (Olson et al., 2017). With varying time budget from one minute to one hour, FLAML outperforms top three open-source AutoML libraries as well as a commercial cloud-based AutoML service in a majority of the tasks given equal or smaller budget, with significant margins. We study an application to selectivity estimation in the end.

[1] https://github.com/microsoft/FLAML

2 Related Work

First, we review the top-performing open-source AutoML libraries according to the AutoML Benchmark (Gijsbers et al., 2019). (1) Auto-sklearn (Feurer et al., 2015) is declared the overall winner of the ChaLearn AutoML Challenge 1 in 2015-2016 and 2 in 2017-2018. It employs Bayesian optimization (BO) (Hutter et al., 2011) for hyperparameter tuning and learner selection, and uses meta-learning to warm-start the search procedure with a few pipelines. (2) TPOT (Olson et al., 2016) (Tree-based Pipeline Optimization Tool) constructs machine learning pipelines of arbitrary length using scikit-learn learners and XGBoost and uses genetic programming for hyperparameter tuning. (3) H2O AutoML (H2O.ai) is a Java-based library. It performs randomized grid search for each learner in the H2O machine learning package, in addition to XGBoost. The learners are ordered manually and each learner is allocated a predefined portion of search iterations. They all use model ensembles to boost accuracy.

A number of commercial platforms are available: Amazon AWS SageMaker (Liberty et al., 2020), DataRobot, Google Cloud AutoML Tables, Microsoft AzureML AutoML, Salesforce TransmogrifAI, H2O Driverless AI, Darwin AutoML and Oracle AutoML. They provide end-to-end AutoML service, i.e., directly consuming uncleaned raw data and then producing trained models and predictions.

To summarize the learnings from existing AutoML systems, the dominating approach is based on trials in a large search space. The order of the trials thus has a large impact in the search efficiency. Meta-learning is one technique often proposed to improve the search order, with the assumption that one can collect a large number of datasets and experiments for meta-training, and the performance of learners and hyperparameters from these experiments is indicative of their future performance in new datasets and tasks (Feurer et al., 2015; Fusi et al., 2018; Shang et al., 2019). In addition,

ensemble of multiple learners is often considered useful for boosting accuracy at the cost of increased inference latency (Erickson et al., 2020).

FLAML is designed to perform efficiently and robustly without relying on meta-learning or ensemble at first order, for several usability reasons. First, this makes FLAML an easy plug-in in new application scenarios, without requiring a developer to collect many diverse meta-training datasets before being able to use it. Second, it allows the user to easily customize learners, search spaces and optimization metrics and use FLAML immediately after the customization, without waiting for another expensive round of meta-learning if any of these changes. Third, our customers prefer single learners over ensembles due to the advantage in model complexity, inference latency, ease of deployment, debuggability and explanability. How to leverage meta-learning and ensemble with good usability is interesting future work.

One notable standalone subarea in AutoML is neural architecture search (NAS) (Elsken et al., 2019) which specifically targets neural networks. Most application scenarios of NAS involve unstructured data like images and text. While the search space and application scenario are different, our design principles in cost minimization might be applicable.

3 API, FORMULATION AND ANALYSIS

FLAML is implemented in Python because of its popularity in data science. It has a scikit-learn (Pedregosa et al., 2011) style API:

```
1 from flaml import AutoML
2 automl = AutoML()
3 automl.fit(X_train, y_train, task='
      classification')
4 prediction = automl.predict(X_test)
```

Additional settings include time budget, optimization metric, estimator list etc. It is easy to add customized learners or metrics in FLAML:

```
1 # MyLearner is a custom estimator class
2 automl.add_learner(learner_name='mylearner'
      , learner_class=MyLearner)
3 # mymetric is a custom metric function
4 automl.fit(X_train, y_train, metric=
      mymetric, time_budget=60,
      estimator_list=['mylearner','xgboost'])
```

The main innovation of FLAML is in its fit() method: automatically producing an accurate model (measured by a given error metric) for an ad-hoc featurized dataset[2].

[2]Given existing fast automatic featurization libraries such as *autofeat* (Horn et al., 2019) and *azureml-sdk* (Mukunthu et al., 2019), FLAML does not innovate on featurization techniques, though the system can easily support feature preprocessors.

Table 1. Notions and notations.

L	number of learners	l	learner
$\tilde{\epsilon}$	validation error	ϵ	test error
$\mathbf{h}$	hyperparameter values	χ	configuration
s	sample size	r	resampling strategy
M	trained model	κ	trial cost

3.1 Formulation

We consider L learners, each with its own set of hyperparameters. The learners can be customized by users, as long as they have well-defined train and prediction methods and search space of hyperparameters. We denote the search space of hyperparameters for learner l as H_l. For each trial, we can choose a learner l, the hyperparameters $\mathbf{h} \in H_l$, together with two other variables: sample size s of training data, and resampling strategy r. s is an integer to denote the number of examples in a sample of training data, and $r \in \{\text{cv}, \text{holdout}\}$ is a binary choice between k-fold cross-validation and holdout with ratio ρ.[3] A *learning configuration* is defined as a tuple $\chi = (l, \mathbf{h}, s, r)$. When we make a trial with χ, we can obtain a validation error $\tilde{\epsilon}(\chi)$ and a model $M(\chi)$. Depending on whether the resampling strategy is cross validation or holdout, the model M corresponds to training data of size s or $s \cdot (1-\rho)$, where ρ is the holdout ratio, and the error $\tilde{\epsilon}$ corresponds to cross validation error or error on the heldout validation data. $\tilde{\epsilon}$ is a proxy of the actual error $\epsilon(M)$ on unseen test data. The cost of the trial is mainly the CPU time of training and testing using cross-validation or holdout, denoted as $\kappa(\chi)$. The goal of fast and economical AutoML is to minimize the total cost before finding a model with the lowest test error. The total cost is expected to increase as the test error decreases, and desired to be approximately optimal.

3.2 Analysis

We first analyze the factors considered in our search sequence and several desirable properties of the search dynamics about them. Figure 2 summarizes the relations among several variables, using notations summarized in Table 1. The domain of the hyperparameters $\mathbf{h}$ depends on the learner l. The test error ϵ is not observable during AutoML. It is a blackbox function of the learner l, the hyperparameters $\mathbf{h}$, and the sample size s. It is approximated by the validation error $\tilde{\epsilon}$. We observe several non-blackbox relations among the variables, which are not first noticed by us but rarely leveraged by existing AutoML systems.

Observation 1 (Sample size + resampling $\rightarrow$ error) *First, it is reasonable to assume the test error ϵ, as well*

[3]In general, we can consider a large search space for the resampling strategy by making k and ρ variables as well. We make k and ρ constants in this work to simplify the problem.

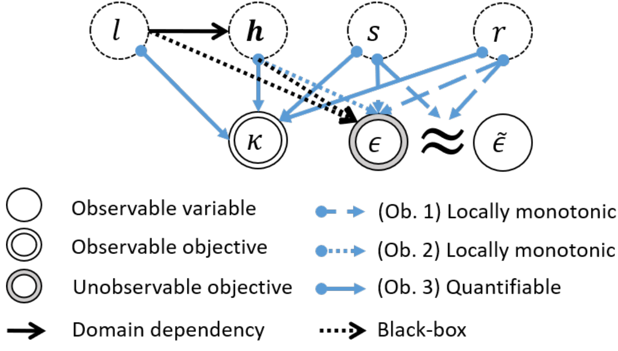


Figure 2. Relations among the variables.

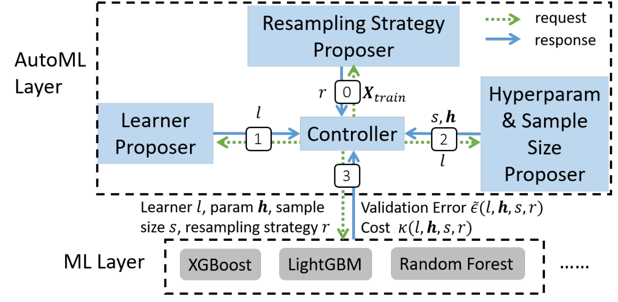


Figure 3. Major components in FLAML.

as the gap between ϵ and $\tilde{\epsilon}$, decreases or stays with the increase of the sample size s when all the other factors are fixed (Huang et al., 2019, pg. 2) ((Nakkiran et al., 2020) observes this after the sample size is above a threshold). Second, the gap between ϵ and $\tilde{\epsilon}$ is smaller for cross-validation than holdout, when all the other factors are fixed (Kohavi, 1995; Feurer et al., 2015).

Observation 2 (Hyperparameter + sample size → error) *Many learners have a subset of hyperparameters related to model complexity or regularization, e.g., the number of trees and the depth of each tree in tree-based learners. For a fixed sample size, ϵ does not necessarily reach its minimum at maximal complexity. Generally speaking, smaller sample size (in a local region) requires lower complexity and more regularization to avoid overfitting (Hastie et al., 2001; Nakkiran et al., 2020).*

Observation 3 (Quantifiable impact on cost) *For each fixed combination of learner l and resampling strategy r, the cost κ is approximately proportional to the sample size s and a subset of cost-related hyperparameters, such as the number of trees. When all the other factors are fixed, k-fold cross-validation roughly takes $\frac{k-1}{1-\rho}\times$ cost as holdout using holdout ratio ρ.*

Based on the joint effect of hyperparameter and sample size on error and cost (Observation 2 and 3), we have the following property.

Property 1 (SuitableSampleSize) *Small sample size can be used to train and compare low-complexity configurations, while large sample size is needed for comparing high-complexity configurations.*

From the compound impact of sample size and resampling strategy on error and cost (Observation 1 and 3), when sample size is small, cross-validation reduces the variance of validation error while the cost is bounded. When sample size is large, validation error from holdout is close to test error, and the cost is much lower than cross-validation. Since the trial-based AutoML requires a fair selection mechanism among all the configurations, we have:

Property 2 (Resample) *Cross-validation is preferred over holdout for small sample size or large time budget.*

From the target of error minimization and Observation 1, as well as the fact that the optimal choice of l^* is unknown, we derive the following property.

Property 3 (FairChance) *Given any search sequence prefix, every learner l should have a chance to be searched again, unless all the valid hyperparameter values of $\mathbf{h}$ have been searched using the full training data size in the prefix.*

From the target of cost minimization and Observation 3, we can derive the following property, which is in general difficult to achieve as the optimal configuration is unknown.

Property 4 (OptimalTrial) *The total cost of any search sequence prefix is desired to be approximately optimal (i.e., have a bounded ratio over the optimal cost) for the lowest error it achieved. Similar for the subsequence corresponding to each learner l.*

Although these properties are idealistic properties and they are not necessarily complete, they provide meaningful guidance in designing a low-cost AutoML system.

4 FLAML

We present our design following the guidelines. Section 4.1 presents an overview, and Section 4.2 details our search strategy used in each component respectively.

4.1 Design Overview

Our design is presented in Figure 3, with the purpose of easy realization of the desired properties described in our analysis. It consists of two layers, including a ML layer and an AutoML layer. The ML layer contains the candidate learners. The AutoML layer includes a learner proposer,

a hyperparameter and sample size proposer, a resampling strategy proposer and a controller. The order of the control flow is indexed on the arrows in Figure 3 as four steps. Steps 0-2 involve choosing the corresponding variables in each component. In step 3, the controller will invoke the trial using the selected learner in the ML layer, and observe the corresponding validation error $\tilde{\epsilon}$ and cost κ. Steps 1-3 are repeated by iterations until running out of budget. In a parallel environment, the controller can execute a new iteration of steps 1-3 before an iteration finishes if there are available resources. Changing one strategy inside each component does not affect the strategy of others. This design allows easy upgrade by incorporation of novel search schemes to replace each component.

Our system differs from previous work in multiple perspectives: (1) It is different in how we decouple the searched variables and search strategies (Table 2). For example, we couple the decision of $\mathbf{h}$ and s in our design to ensure sample size is decided together with the hyperparameters, which reflects Property 1. We decouple learner and hyperparameters and use the order of $\{l\} \rightarrow \{\mathbf{h}, s\}$ to respect domain dependency. (2) As the first trial-based library targeting ad-hoc data (including large-scale datasets) using low-cost, FLAML focuses on the core search efficiency and does not use meta-learning or ensemble. It is considered as future work to develop lightweight meta-learning and ensemble techniques for FLAML while keeping the system economic, generic and capable of handling ad-hoc datasets. (3) Since the commonly used search strategies are not designed to deeply exploit the compound relations of the multiple factors as analyzed in Section 3.2, we employ new search strategies as introduced in the next subsection.

4.2 Search Strategy

Before introducing our search strategies, we first introduce the notion of *estimated cost for improvement* (ECI) which will be used in the search strategies. For each learner $l \in [L]$, $ECI_1(l)$ (abrv. ECI_1) denotes our estimation of the cost for l to find a configuration with lower error than the current best error (denoted as $\tilde{\epsilon}_l$) under the current sample size. $ECI_2(l)$ (abrv. ECI_2) denotes our estimation of the cost to try the current configuration for l (which took κ_l cost) with increased sample size (multiplied by a factor of c). Finally, $ECI(l)$ (abrv. ECI) is our estimation of the cost it takes to find a configuration with l and lower error than the current best error among all the learners (denoted as $\tilde{\epsilon}^*$).

Let $K_1 > K_2$ be abbreviations of $K_1(l) > K_2(l)$, representing the total cost spent on l when the two most recent updates of best configurations happened for l respectively, K_0 (abbreviations of $K_0(l)$) be the total cost spent on l so far, and δ (abbreviations of $\delta(l)$) be the error reduction between the two corresponding best configurations. We set:

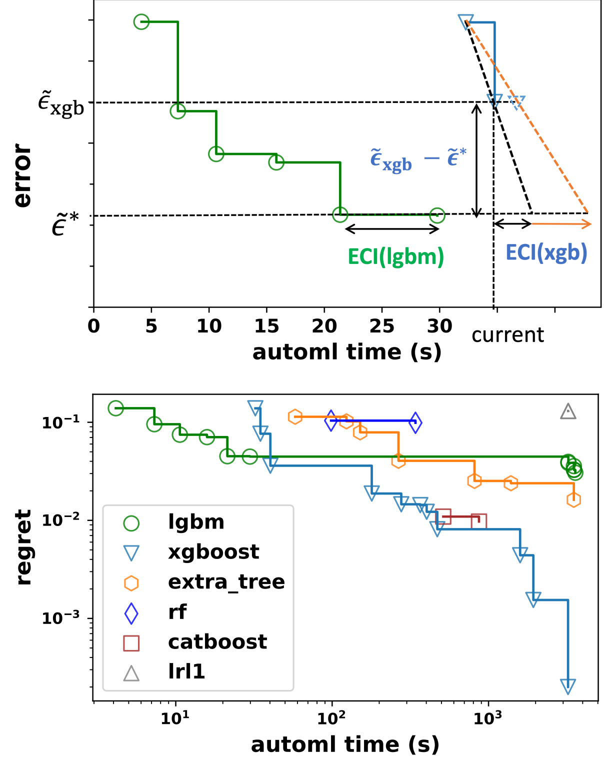


Figure 4. Illustration of ECI-based prioritization.

$$ECI_1 = \max(K_0 - K_1, K_1 - K_2), ECI_2 = c \cdot \kappa_l$$
$$ECI = \max\left(\frac{(\tilde{\epsilon}_l - \tilde{\epsilon}^*)(K_0 - K_2)}{\delta}, \min(ECI_1, ECI_2)\right) \quad (1)$$

The calculation of ECI_1 is based on the assumption that it takes higher cost to find an improvement in the later stage of search.[4] ECI_2 is set to be c times as large as the trial cost of the current configuration for l, because we expect the error of the current configuration to improve when given c times as large sample size. This simple cost estimation can be refined when the complexity of the training procedure is known with respect to sample size. It works well for the learners in our experiments which have linear complexity. ECI is calculated depending on whether l currently has the lowest error among all the learners:

(a) l currently has the best error among all the learners. In this case, by definition $ECI = \min(ECI_1, ECI_2)$.

(b) l does not have the best error among all the learners

[4]For learners which have not been tried in the search, the ECI_1 is set to the smallest trial cost for those learners. Since the smallest trial cost varies with input data, we first run the fastest learner and gets its smallest cost on the input data, and then set the ECI_1 for other learners as multiples of this cost using predefined constants.

Table 2. Comparison of search strategy.

Tool	Searched variable	Search strategy
Alpine Meadow (Shang et al., 2019)	$\{l\} \rightarrow \{\mathbf{h}\} \rightarrow \{s\}$	Meta learning → BO → Progressive sampling
Auto-sklearn (Feurer et al., 2015)	$\{l, \mathbf{h}\}$	Bayesian optimization, with meta-learning and ensemble
H2O AutoML (H2O.ai)	$\{l\} \rightarrow \{\mathbf{h}\}$	Enumeration → Randomized grid search, with ensemble
HpBandSter (Falkner et al., 2018; Li et al., 2017)	$\{l, \mathbf{h}\}, \{s\}$	Bayesian optimization, Hyperband
PMF-automl (Fusi et al., 2018)	$\{l, \mathbf{h}\}$	Collaborative filtering, with post-processing ensemble
TPOT (Olson et al., 2016)	$\{l, \mathbf{h}\}$	Genetic programming, with ensemble embedded
FLAML	$\{l\} \rightarrow$ $\{\mathbf{h}, s\}$	ECI-based sampling of learner choices → Randomized direct search, ECI-based choice of sample size

currently. For l to match the current best error $\tilde{\epsilon}^*$, it needs to improve its own error by at least $\delta = \tilde{\epsilon}_l - \tilde{\epsilon}^*$. To estimate the cost to fill that gap, we also need to estimate the *efficiency of improvement* v for l. That is, how fast l is expected to reduce the error in its own search sequence. We calculate v as: $v = \frac{\delta}{\tau}$, where $\tau = K_0 - K_2$ is the estimated cost spent on l for making the error reduction δ. In the special case where $\delta = 0$, i.e., the first configuration searched for l is the best configuration for l so far, we set $\delta = \tilde{\epsilon}_l$, and τ as the total cost spent on l. In our implementation, we double the cost to fill the gap as the estimated cost for finding an improvement because we assume the improvement with respect to cost has a diminishing return.

Combining the two cases we have Eq. (1). A visual demonstration is provided in Figure 4 corresponding to the same example in Figure 1. The figure on the top plots the best error per learner vs. automl time, and visualizes the ECI of two learners LightGBM and XGBoost based on Eq. (1) when the current time point is 35s. To illustrate that ECI is self-adjustable, we add a hypothetical new trial of XGBoost (the dashed triangle marker at 38s) which does not find a better model. In this case, ECI(xgb) will be increased as shown in the horizontal orange arrow, and the priority of XGBoost will be decreased. The figure on the bottom visualizes the search trajectory of each learner.

Step 0: The resampling strategy proposer chooses r. Resampling strategy is decided based on a simple thresholding rule. It is the simplest design which follows Property 2 (Resample). If the training dataset has fewer than 100K instances and # instances × # features / budget is smaller than 10M/hour, we use cross validation. Otherwise, we use holdout. This simple thresholding rule can be easily replaced by more complex rules, e.g., from meta learning. By default, FLAML uses 5-fold cross-validation and 0.1 as the holdout ratio.

Step 1: The learner proposer chooses l. With the concept of ECI introduced, we design a search strategy where each learner l is chosen with probability proportional to $1/ECI(l)$. There are several reasons why ECI is desirable in our system: (1) This design follows Property 3 and 4. Property 4 (OptimalTrial) suggests that we prioritize choices which are expected to improve the error using small cost, hence we assign choices with lower ECI higher probability. (2) Instead of directly choosing the learner with lowest ECI, we use randomization because Property 3 (FairChance) requires every learner to have chance to be searched again, and our estimation is not precise. Based on our sampling scheme, the expectation of ECI for the probabilistic choice is $E[ECI] = \sum_l \frac{ECI(l) \cdot ECI(l)^{-1}}{\sum_{l'} ECI(l')^{-1}}$ = the harmonic mean of all the ECIs. That means, the expected cost for improvement using our sampling scheme is still dominated by and close to the lowest ECIs. (3) With more observations about l being collected, ECI will be updated dynamically. The dynamic update of ECI leads to a self-correcting behavior: If our ECI is an overestimate, it will decrease; if it is an underestimate, it will increase. This can be reflected from the formula of ECI and Figure 4.

Although a related concept EIperSec (Expected Improvement per Second) was proposed in (Snoek et al., 2012), it is designed for a different context of Bayesian optimization and not applicable to our goal of learner selection.

Step 2: The hyperparameter and sample size proposer chooses $\mathbf{h}$ and s.

For hyperparameters, we adopt a recently proposed randomized direct search method (Wu et al., 2021), which can perform cost-effective optimization for cost-related hyperparameters. The algorithm uses a low-cost configuration as the start point. At each iteration, it samples a random direction $\mathbf{u}$ in a $(|\mathbf{h}| - 1)$-dimensional unit sphere and then decides whether to move to a new $\mathbf{h}$ along the randomly sampled direction (or the opposite direction) depending on the observed sign of change of validation error. The cost of the next trial can be upper bounded with respect to the cost of the best config of the considered learner. This upper bound of trial cost is guaranteed by the search procedure used, and increases only progressively if the best error is reduced. Step-size of the move is adjusted adaptively (large in the beginning to fast approach the required complexity) and the search is restarted (from randomized initial points) occasionally to escape local optima.

Though the randomized direct search method does not handle subsampling, it is a good option to use in our framework

Table 3. Details of the case study in Figure 1. It reveals that FLAML avoids trying unnecessarily expensive configs in the beginning more than HpBandSter though they are given the same search space. Even though FLAML eventually tries expensive configs, it chooses the more promising learner (in this example, XGBoost) based on the observed cost and error in early trials.

Iter	Time (s)	Learner	Config tried by FLAML	Error	Cost (s)
1	4	LightGBM	tree num: 4, leaf num: 4, min child weight: 20, learning rate: 0.1...	**0.3272**	3
...	...	...	...	...	...
9	40	XGBoost	tree num: 13, leaf num: 9, min child weight: 18, learning rate: 0.4...	**0.2242**	5
...	...	...	...	...	...
20	402	XGBoost	tree num: 76, leaf num: 116, min child weight: 3, learning rate: 0.2...	**0.2003**	26
...	...	...	...	...	...
26	1935	XGBoost	tree num: 548, leaf num: 247, min child weight: 1.1, learning rate: 0.02...	**0.1896**	238
27	3225	XGBoost	tree num: 1312, leaf num: 739, min child weight: 1.1, learning rate: 0.01...	**0.1882**	1290
...	...	...	...	...	...
Iter	Time (s)	Learner	Config tried by HpBandSter	Error	Cost (s)
1	16	XGBoost	tree num: 47, leaf num: 50, min child weight: 0.004, learning rate: 0.8...	**0.2497**	15
2	1193	XGBoost	tree num: 17863, leaf num: 2735, min child weight: 3.7, learning rate: 0.1...	**0.1979**	1177
3	1356	CatBoost	early stop rounds: 15, learning rate: 0.03...	**0.1978**	163
...	...	...	...	...	...
7	2011	XGBoost	tree num: 10369, leaf num: 369, min child weight: 0.1, learning rate: 0.4...	0.2036	583
8	3325	RF	tree num: 2155, max features: 0.36, criterion: entropy	0.2007	1313
...	...	...	...	...	...

due to two important reasons: (1) The method proved its effectiveness in controlling trial cost both theoretically and empirically. The theoretical analysis of this method shows its alignment with Property 4 (OptimalTrial), and its empirical study demonstrates superiority over Bayesian optimization methods including the one using EIPerSec when cost-related hyperparameters exist. (2) The method works without requiring the exact validation error of each trial as feedback, as long as the relative order of any two trials can be determined, and we can leverage that to modify the method to incorporate data subsampling. We make several important adjustments to enable data subsampling. Specifically, we begin with a small sample size (10K) for each l. For each requested l, we first make a choice between increasing the sample size and trying a new configuration with the current sample size, by comparing $ECI_1(l)$ and $ECI_2(l)$. When $ECI_1(l) \geq ECI_2(l)$, we keep the current hyperparameter values and increase the sample size. Otherwise, we stay with the current sample size, and generate hyperparameter values using the randomized direct search method described above. With this design, the system will adaptively change the sample size as needed. Once the sample size for a learner reaches the full data size, it keeps using that size until convergence for that learner. That reduces the risk of pruning good configurations by small sample size compared to multi-fidelity pruning. We reset the sample size to the initial value as the search for that learner is restarted.

The implementation of the randomized direct search method follows (Wu et al., 2021). At each iteration, a random direction is used first to train a model. If the error does not reduce, we train another model using the opposite direction. The initial stepsize is set to be $\sqrt{d}$. It will be decreased when the number of consecutively no improvement iterations is larger than 2^{d-1} until it reaches a lower bound, i.e., converges. Specifically, stepsize is discounted by a reduction ratio > 1, which is intuitively the ratio between the total number of iterations taken in total since the last restart of the search and the total number of iterations taken to find the best configuration since the last restart. We perform adaptive step-size adjustments and random restart only when the largest sample size is reached. Our system shuffles the data randomly in the beginning and to get a sample with size s, it takes the first s tuples of the shuffled data. Stratified shuffling is used for classification tasks based on the labels.

Advantages of our design. First, our search strategies are designed toward strong *final performance* (i.e., low final error) for ad-hoc datasets, which requires a large configuration search space. The random sampling according to ECI in Step 1 and the random restart in Step 2 help the method escape local optima. Second, our search strategies are designed toward strong *anytime performance* for ad-hoc datasets. The ECI-based prioritization in Step 1 favors cheap learners in the beginning but penalizes them later if the error improvement is slow. The hyperparameter and sample size proposer in Step 2 tends to propose cheap configurations at the beginning stage of the search, but quickly move to configurations with high model complexity and large sample size when needed in the later stage of the search. These designs make FLAML navigate large search space efficiently for both small and large datasets. Last, the computational overhead in the AutoML layer compared to the trial cost in the ML layer is negligible in our solution: ECI-based sampling, randomized direct search, and update of ECIs. The complexity of these operations for each iteration is linear with the dimensionality of hyperparameters, and does not depend on the number of trials.

5 Experiments

Our main empirical study is based on a combination of a recent open source AutoML classification benchmark (Gijsbers et al., 2019) and a regression benchmark (Olson et al., 2017), for a total of 53 datasets (39 classification + 14 regression). The two benchmarks can be found at: AutoML Benchmark (classification) - https://openml.github.io/automlbenchmark, and PMLB (regression) - https://github.com/EpistasisLab/penn-ml-benchmarks. The first benchmark collects 39 classification tasks that represent real-world data science problems of various sizes, domains and levels of difficulty from previous AutoML papers, competitions and benchmarks. In the second benchmark PMLB, most datasets are of small scale, from which we selected the regression datasets whose numbers of instances are larger than 10,000. That results in 14 regression tasks. The statistics of all the 53 datasets are listed in Table 6-8 in the appendix. The 53 datasets have #instance $\times$ #feature ranging from 2,992 to 85,920,000 and vary in the occurrence of numeric features, categorical features and missing values. The referred AutoML classification benchmark uses roc-auc and negative log-loss as the scores to evaluate the performance on binary classification tasks and multi-class classification tasks respectively. It calibrates the original scores using a constant class prior predictor (=0) and a tuned random forest (=1), the higher the better. The tuned random forest is a strong baseline taking a long time to finish, and achieving a score above 1 is not easy according to (Gijsbers et al., 2019). For regression tasks, we use the r2 score which is a metric bounded by 1 before calibration.

We compare FLAML (v0.1.3) to four trial-based AutoML libraries plus a hyperparameter optimization library designed for budget constrained scenarios: auto-sklearn (v0.9.0)[5], H2O AutoML (v3.30.0.3), TPOT (v0.10.1), cloud-automl (a commercial cloud-based AutoML service from one major cloud provider), and HpBandSter (v0.7.4, an implementation of BOHB which integrates Bayesian optimization with Hyperband). The first three are the top three performers reported by the AutoML benchmark (Gijsbers et al., 2019) and their performance is close to each other. HpBandSter uses the same search space and resampling strategy as those of FLAML (Table 5 in the appendix). It is worth noting that all the libraries use a different search space from each other by design except for HpBandSter and FLAML. The search space of FLAML (reported in the appendix) neither subsumes, nor is subsumed by the search space of auto-sklearn, cloud-automl, H2O AutoML or TPOT. It is very challenging, if not impossible, to equalize the search space due to the specific designs of each library (e.g., meta-learning by auto-sklearn and cloud-automl, and special grid search by H2O AutoML). We use their default setting and do not introduce our own bias.

[5] AutoSklearn2Classifier (Feurer et al., 2020) for classification.

All experiments are executed on an Ubuntu server with Intel Xeon Gold 6140 2.3GHz, and 512GB RAM. We use 1 CPU core for each compared solution and vary the time budget from one minute to one hour. We choose these settings because this work is mainly concerned about performance using low resource, while the numbers in (Gijsbers et al., 2019) are obtained using a rather generous budget (8 to 32 hours of total CPU time). Cloud-automl with 1m budget is not reported since it does not return within 2 minutes. As each dataset has been split into 10 folds by OpenML (Vanschoren et al., 2014), all the reported results are averaged over the 10 folds.

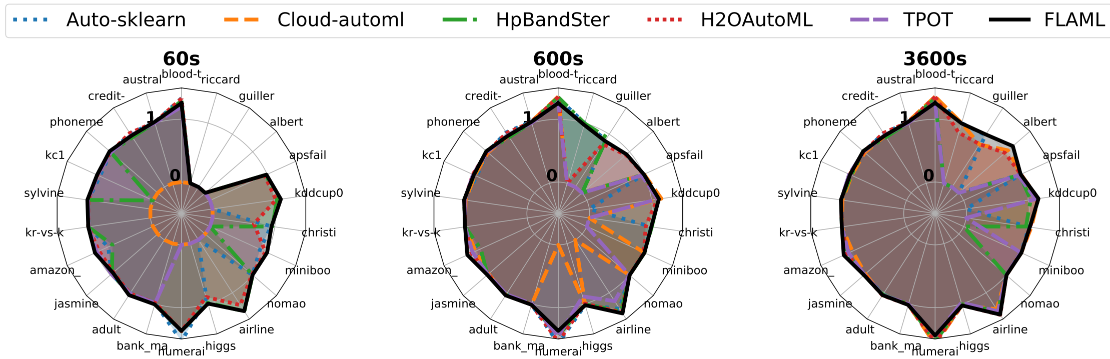
5.1 Comparative Study

The scaled scores of all the methods given different desired time budgets (1m, 10m and 1h) on all the datasets are shown in Figure 5. Each of the radar charts shows the scaled scores of different methods on a group of datasets (spokes on the radar chart) given a desired time budget. Results of all 53 datasets are summarized into 3 sub-figures (rows) according to their task types. Each row shows the performance comparison on the same group of datasets given different desired time budgets. Figure 6 presents the distribution of score difference between FLAML and each individual baseline, under equal budget (the first row) or smaller budget for FLAML (the second row).

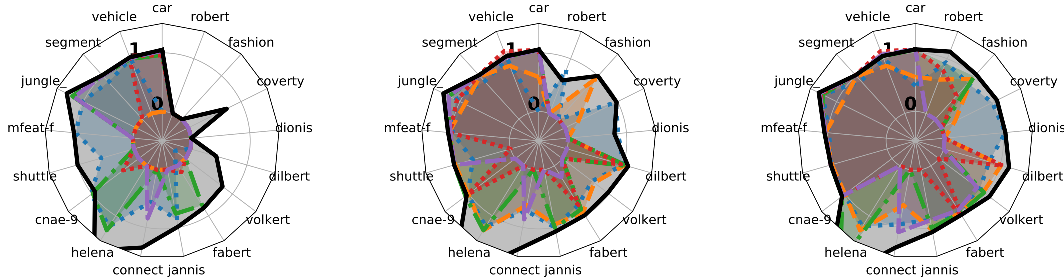
When using equal time budgets, FLAML clearly outperforms every competitor with large margins in most cases. In a small fraction of cases, FLAML underperforms by a small margin. Even with a smaller time budget, FLAML can be better than or equal to the others in many cases. For example, FLAML's 1m result is no worse than others' 10m result on 62%-83% datasets, and 72%-89% for 10m vs. 1h. In sum, FLAML demonstrates *significant margin over each competitor given equal budgets, and competitive performance given smaller budgets than competitors*. To be fair, the prior libraries are not primarily designed for the same low-resource setting as ours. We note that the search space of FLAML contains both cheap and expensive configurations, but our integrated search strategies make FLAML wisely prioritize them. As seen in Figure 1 and Table 3, FLAML's adaptive behavior is the key to strong anytime performance.

5.2 Ablation Study

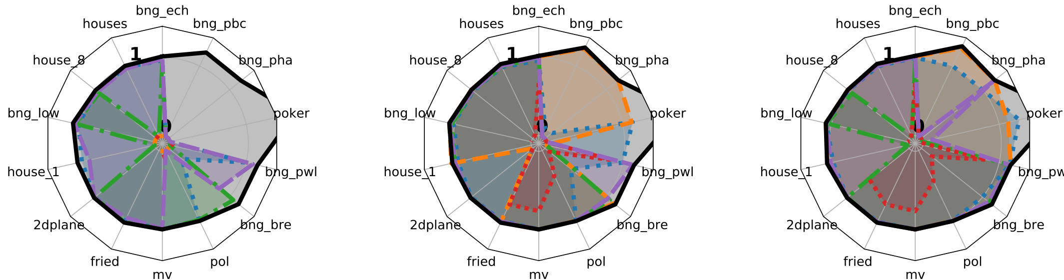
We study the effect of the three components in our system, by comparing FLAML with three alternatives: *roundrobin*, which takes trials for each learner l in turn; *fulldata*, which uses the full data for the trials; and *cv*, which uses cross validation for all the trials. Figure 7 plots the validation



(a) Binary classification datasets ordered by size counter clockwise, from smallest *blood-transfusion* to largest *riccardo*



(b) Multi-class classification datasets ordered by size counter clockwise, from smallest *car* to largest *robert*



(c) Regression datasets ordered by size counter clockwise, from smallest *bng_echomonths* to largest *bng_pbc*

Figure 5. Scaled scores of AutoML libraries on each dataset with each time budget. The longer is each spoke the better.

error of FLAML and its alternatives on a binary classification dataset *MiniBooNE*, a multi-class classification dataset *Dionis*, and a regression dataset *bng_pbc*. The figure shows how the error improves with respect to search time. We can see that when removing any of the three aspects of the FLAML search strategy, the search performance degrades. In particular, the gap between FLAML and *roundrobin* increases before converging due to FLAML's prioritization to more promising learners. The gap between FLAML and *fulldata* is large initially because the initial trials of FLAML are very fast using small sample of training data. That gap reduces later as FLAML increases its sample size.

The ablation study verifies the effectiveness of the search strategies. Figure 8 in the appendix plots the score difference between FLAML and these alternatives over all the datasets.

5.3 Application to Selectivity Estimation

As an example application to database systems, we evaluate the performance of AutoML libraries to the selectivity estimation task. Selectivity estimates are necessary inputs for a query optimizer to identify a good execution plan (Selinger et al., 1979). A good selectivity estimator should provide accurate and fast estimates for a wide variety of intermediate query expressions at reasonable construction overhead (Cormode et al., 2012). Estimators in most database systems make use of limited statistics on the data, e.g., per-attribute histograms or small data samples on the base tables (Oracle docs; SQL Server docs; Neumann & Freitag, 2020). (Dutt et al., 2019; 2020) developed low-overhead regression models which achieve much lower q-error (a relative error metric used by the selectivity estimation literature) than all

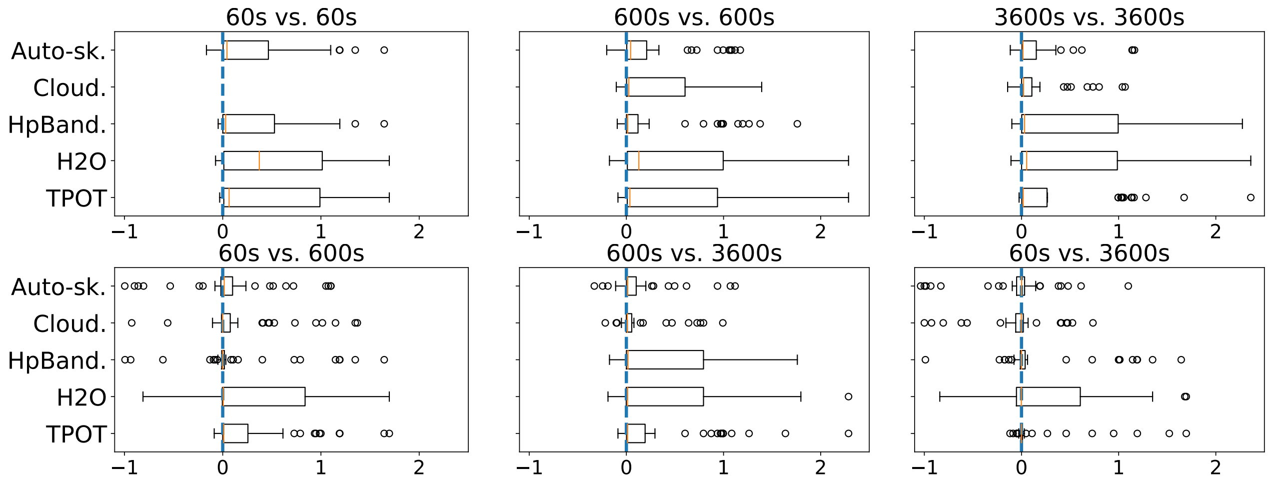
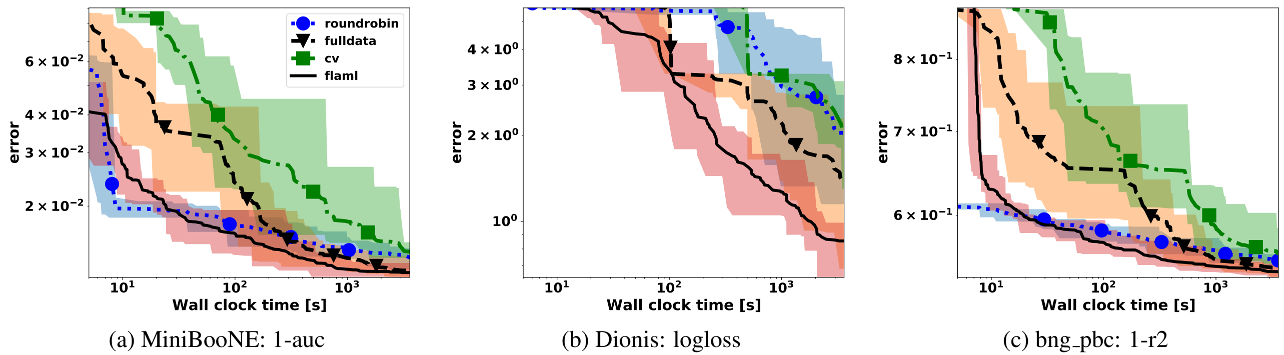


Figure 6. Box plot of scaled score difference between FLAML and other libraries when FLAML uses equal or smaller budget (positive difference meaning FLAML is better).



(a) MiniBooNE: 1-auc (b) Dionis: logloss (c) bng_pbc: 1-r2

Figure 7. Variations of FLAML. Lines – average error over 10 folds; shades – max and min.

Table 4. 95th-percentile q-error for selectivity estimation (search time listed if at least one method exceeds time limit). H2O AutoML cannot return a model with the given budget.

Dataset	FLAML	Auto-sk.	TPOT	Manual
2D-Forest	**1.41**	1.42	2.70	1.84
2D-Power	**2.03**	3.28	4.70	4.09
2D-TPCH	2.19(42s)	**2.11**(197s)	N/A	3.04
4D-Forest1	**2.91**	4.33	11.9	4.41
4D-Forest2	**4.40**(45s)	5.93(55s)	17.4(146s)	6.26
4D-Power	**2.42**(49s)	3.78(197s)	12.4(79s)	4.29
7D-Higgs	**3.16**(60s)	5.83(55s)	9.65(91s)	6.54
7D-Power	**4.25**(46s)	6.87(55s)	65.2(102s)	7.57
7D-Weather	**4.71**(54s)	6.44(55s)	30.1(118s)	6.84
10D-Forest	**9.09**(49s)	19.8(147s)	96.2(89s)	15.1

the other methods with similar inference time, including the ones used in commercial database products. The recommended learner is XGBoost with 16 trees and 16 leaves per tree. We denote this configuration as 'Manual'.

Table 4 compares the 95th-percentile q-error of the models found by different AutoML libraries with one CPU minute budget, using the same datasets from (Dutt et al., 2019). FLAML outperforms the other AutoML libraries as well as the manual configuration. On 10D-Forest, FLAML is the only AutoML solution that outperforms Manual.

6 Future Work

While FLAML has superior performance on a variety of benchmark tasks compared to the state of the art, it does not use meta learning to optimize per task instance based on previous experience. It is worthwhile to think how to leverage meta learning in the cost-optimizing framework without losing the robustness on ad-hoc datasets. Similarly, it is interesting to study the new tradeoff between cost and error when model ensembles are introduced.

Acknowledgments

The authors appreciate suggestions from Surajit Chaudhuri, Nadiia Chepurko, Alex Deng, Anshuman Dutt, Johannes Gehrke, Silu Huang, Christian Konig, and Haozhe Zhang.

REFERENCES

Agrawal, A., Chatterjee, R., Curino, C., Floratou, A., Gowdal, N., Interlandi, M., Jindal, A., Karanasos, K., Krishnan, S., Kroth, B., et al. Cloudy with high chance of dbms: A 10-year prediction for enterprise-grade ml. In *CIDR'20*, 2020.

Cormode, G., Garofalakis, M., Haas, P. J., and Jermaine, C. Synopses for massive data: Samples, histograms, wavelets, sketches. *Found. Trends Databases*, 4(1–3): 1–294, January 2012.

Dutt, A., Wang, C., Nazi, A., Kandula, S., Narasayya, V. R., and Chaudhuri, S. Selectivity estimation for range predicates using lightweight models. *PVLDB*, 12(9):1044–1057, 2019.

Dutt, A., Wang, C., Narasayya, V., and Chaudhuri, S. Efficiently approximating selectivity functions using low overhead regression models. In *46th International Conference on Very Large Data Bases*, 2020.

Elsken, T., Metzen, J. H., and Hutter, F. Neural architecture search: A survey. *Journal of Machine Learning Research*, 20(55):1–21, 2019.

Erickson, N., Mueller, J., Shirkov, A., Zhang, H., Larroy, P., Li, M., and Smola, A. Autogluon-tabular: Robust and accurate automl for structured data. *arXiv:2003.06505*, 2020.

Falkner, S., Klein, A., and Hutter, F. BOHB: Robust and efficient hyperparameter optimization at scale. In *ICML*, 2018.

Feurer, M., Klein, A., Eggensperger, K., Springenberg, J., Blum, M., and Hutter, F. Efficient and robust automated machine learning. In *NIPS*, 2015.

Feurer, M., Eggensperger, K., Falkner, S., Lindauer, M., and Hutter, F. Auto-sklearn 2.0. *arXiv:2007.04074*, 2020.

Fusi, N., Sheth, R., and Elibol, M. Probabilistic matrix factorization for automated machine learning. In *Advances in Neural Information Processing Systems*, 2018.

Galakatos, A., Markovitch, M., Binnig, C., Fonseca, R., and Kraska, T. Fiting-tree: A data-aware index structure. In *SIGMOD*, 2019.

Gijsbers, P., LeDell, E., Thomas, J., Poirier, S., Bischl, B., and Vanschoren, J. An open source automl benchmark. In *AutoML Workshop at ICML 2019*, 2019. URL `http://arxiv.org/abs/1907.00909`.

H2O.ai. H2o automl. `http://docs.h2o.ai/h2o/latest-stable/h2o-docs/automl.html`. 2019-10-29.

Hastie, T., Tibshirani, R., and Friedman, J. *The Elements of Statistical Learning*. Springer New York Inc., New York, NY, USA, 2001.

Horn, F., Pack, R., and Rieger, M. The autofeat python library for automatic feature engineering and selection. In *Machine Learning and Knowledge Discovery in Databases. ECML PKDD 2019*, 2019.

Huang, S., Wang, C., Ding, B., and Chaudhuri, S. Efficient identification of approximate best configuration of training in large datasets. In *Proceedings of the 33rd AAAI Conference on Artificial Intelligence (AAAI)*, 2019.

Hutter, F., Hoos, H. H., and Leyton-Brown, K. Sequential model-based optimization for general algorithm configuration. In Coello, C. A. C. (ed.), *Learning and Intelligent Optimization*, 2011.

Kipf, A., Kipf, T., Radke, B., Leis, V., Boncz, P., and Kemper, A. Learned cardinalities: Estimating correlated joins with deep learning. In *CIDR*, 2019.

Kohavi, R. A study of cross-validation and bootstrap for accuracy estimation and model selection. In *IJCAI'95*, 1995.

Kraska, T., Beutel, A., Chi, E. H., Dean, J., and Polyzotis, N. The case for learned index structures. In *SIGMOD*, 2018.

Li, L., Jamieson, K., DeSalvo, G., Rostamizadeh, A., and Talwalkar, A. Hyperband: A novel bandit-based approach to hyperparameter optimization. In *ICLR'17*, 2017.

Li, L., Jamieson, K., Rostamizadeh, A., Gonina, E., Ben-tzur, J., Hardt, M., Recht, B., and Talwalkar, A. A system for massively parallel hyperparameter tuning. In *Proceedings of Machine Learning and Systems*, 2020.

Liaw, R., Bhardwaj, R., Dunlap, L., Zou, Y., Gonzalez, J. E., Stoica, I., and Tumanov, A. Hypersched: Dynamic resource reallocation for model development on a deadline. SoCC'19, 2019.

Liberty, E., Karnin, Z., Xiang, B., Rouesnel, L., Coskun, B., Nallapati, R., Delgado, J., Sadoughi, A., Astashonok, Y., Das, P., et al. Elastic machine learning algorithms in amazon sagemaker. In *SIGMOD'20*, 2020.

Ma, L., Aken, D. V., Hefny, A., Mezerhane, G., Pavlo, A., and Gordon, G. J. Query-based workload forecasting for self-driving database management systems. In *SIGMOD*, 2018.

Marcus, R. C. and Papaemmanouil, O. Plan-structured deep neural network models for query performance prediction. *PVLDB*, 12(11):1733–1746, 2019.

Mukunthu, D., Shah, P., and Tok, W. *Practical Automated Machine Learning on Azure: Using Azure Machine Learning to Quickly Build AI Solutions*. O'Reilly Media, Incorporated, 2019.

Nakkiran, P., Kaplun, G., Bansal, Y., Yang, T., Barak, B., and Sutskever, I. Deep double descent: Where bigger models and more data hurt. In *ICLR*, 2020.

Neumann, T. and Freitag, M. J. Umbra: A disk-based system with in-memory performance. In *CIDR*, 2020.

Olson, R. S., Urbanowicz, R. J., Andrews, P. C., Lavender, N. A., Kidd, L. C., and Moore, J. H. Automating biomedical data science through tree-based pipeline optimization. In Squillero, G. and Burelli, P. (eds.), *Applications of Evolutionary Computation*, pp. 123–137. Springer International Publishing, 2016.

Olson, R. S., La Cava, W., Orzechowski, P., Urbanowicz, R. J., and Moore, J. H. Pmlb: a large benchmark suite for machine learning evaluation and comparison. *BioData Mining*, 10(1):36, Dec 2017. ISSN 1756-0381. doi: 10.1186/s13040-017-0154-4. URL https://doi.org/10.1186/s13040-017-0154-4.

Oracle docs. Optimizer statistics (release 18). https://docs.oracle.com/en/database/oracle/oracle-database/18/tgsql/optimizer-statistics.html#GUID-0A2F3D52-A135-43E1-9CAB-55BFE068A297.

Pedregosa, F., Varoquaux, G., Gramfort, A., Michel, V., Thirion, B., Grisel, O., Blondel, M., Prettenhofer, P., Weiss, R., Dubourg, V., Vanderplas, J., Passos, A., Cournapeau, D., Brucher, M., Perrot, M., and Duchesnay, E. Scikit-learn: Machine learning in python. *JMLR*, 12:2825–2830, November 2011.

Renggli, C., Karlaš, B., Ding, B., Liu, F., Wu, W., and Zhang, C. Continuous integration of machine learning models with ease.ml/ci: Towards a rigorous yet practical treatment. In *SysML Conference (SysML 2019)*, 2019.

Selinger, P. G., Astrahan, M. M., Chamberlin, D. D., Lorie, R. A., and Price, T. G. Access path selection in a relational database management system. SIGMOD'79, 1979.

Shang, Z., Zgraggen, E., Buratti, B., Kossmann, F., Eichmann, P., Chung, Y., Binnig, C., Upfal, E., and Kraska, T. Democratizing data science through interactive curation of ml pipelines. In *SIGMOD*, 2019.

Snoek, J., Larochelle, H., and Adams, R. P. Practical bayesian optimization of machine learning algorithms. In *NIPS*, 2012.

SQL Server docs. Statistics in Microsoft SQL Server 2017. https://docs.microsoft.com/en-us/sql/relational-databases/statistics/statistics?view=sql-server-2017.

Vanschoren, J., van Rijn, J. N., Bischl, B., and Torgo, L. Openml: Networked science in machine learning. *SIGKDD Explor. Newsl.*, 15(2):49–60, June 2014.

Wu, Q., Wang, C., and Huang, S. Frugal optimization for cost-related hyperparameters. In *AAAI'21*, 2021.

Table 5. Default search space in FLAML. S denotes the number of training instances. Bold values indicate initialization corresponding to lowest complexity and cost. lr - logistic regression.

Learner	Hyperparameter	Type	Range
XGBoost	tree num	int	[**4**, min(32768,S)]
	leaf num	int	[**4**, min(32768,S)]
	min child weight	float	[0.01, **20**]
	learning rate	float	[0.01, 1.0]
	subsample	float	[0.6, 1.0]
	reg alpha	float	[1e-10, 1.0]
	reg lambda	float	[1e-10, 1.0]
	colsample by level	float	[0.6, 1.0]
	colsample by tree	float	[0.7, 1.0]
LightGBM	tree num	int	[**4**, min(32768,S)]
	leaf num	int	[**4**, min(32768,S)]
	min child weight	float	[0.01, **20**]
	learning rate	float	[0.01, 1.0]
	subsample	float	[0.6, 1.0]
	reg alpha	float	[1e-10, 1.0]
	reg lambda	float	[1e-10, 1.0]
	max bin	float	[7, 1023]
	colsample by tree	float	[0.7, 1.0]
CatBoost	early stop rounds	int	[**10**, 150]
	learning rate	float	[0.005,0.2]
sklearn random forest	tree num	int	[**4**, min(2048,S)]
	max features	float	[0.1, 1.0]
	split criterion	cat	{gini, entropy}
sklearn extra trees	tree num	int	[**4**, min(2048,S)]
	max features	float	[0.1, 1.0]
	split criterion	cat	{gini, entropy}
sklearn lr	C	float	[0.03125, 32768]

APPENDIX

The default search space of FLAML is shown in Table 5.

For the learners which have not been tried in the search, $ECI_1(l)$ is set to the smallest trial cost for each learner l. Since the smallest trial cost varies with input data, we first run the fastest learner and get its smallest cost on the input data, and then set the ECI_1 for other learners as multiples of this cost using predefined constants. These constants are easy to set as we only need to calibrate the running time of the fastest configuration of each learner offline. We use the following constants: {'lightgbm':1, 'xgboost':1.6, 'extra tree':1.9, 'rf':2, 'catboost':15, 'lr':160 }. Meta-learning can be potentially applied here to have a more instance-specific prediction of the running time of the initial configuration. We use $c = 2$ as the multiplicative factor of sample size in the experiments.

FLAML is designed to work with low resource consumption, and the extra computation in FLAML beyond trial cost is negligible. So it tries one configuration at a time and lets the learner consume all the given resources (cores and RAM). Since we start search from inexpensive models for every learner, this design minimizes the latency between two iterations so that the proposers can get feedback as early

Table 6. Binary classification datasets.

name	task id	# instance	# feature
adult	7592	48842	14
Airlines	189354	539383	7
Albert	189356	425240	78
Amazon_employee_access	34539	32769	9
APSFailure	168868	76000	170
Australian	146818	690	14
bank_marketing	14965	45211	16
blood-transfusion	10101	748	4
christine	168908	5418	1636
credit-g	31	1000	20
guillermo	168337	20000	4296
higgs	146606	98050	28
jasmine	168911	2984	144
kc1	3917	2109	21
KDDCup09_appetency	3945	50000	230
kr-vs-kp	3	3196	36
MiniBooNE	168335	130064	50
nomao	9977	34465	118
numerai28.6	167120	96320	21
phoneme	9952	5404	5
riccardo	168333	20000	4296
sylvine	168853	5124	20

Table 7. Multi class classification datasets.

name	task id	# instance	# feature
car	146821	1728	6
cnae-9	9981	1080	856
connect-4	146195	67557	42
Covertype	7593	581012	54
dilbert	168909	10000	2000
Dionis	189355	416188	60
fabert	168852	8237	800
Fashion-MNIST	146825	70000	784
Helena	168329	65196	27
Jannis	168330	83733	54
jungle_chess_2pcs...	167119	44819	6
mfeat-factors	12	2000	216
Robert	168332	10000	7200
segment	146822	2310	19
shuttle	146212	58000	9
vehicle	53	846	18
volkert	168810	58310	180

Table 8. Regression Datasets.

name	task id	# instance	# feature
2dplanes	2306	40768	10
bng_breastTumor	7324	116640	9
bng_echomonths	7323	17496	9
bng_lowbwt	7320	31104	9
bng_pbc	7318	1000000	18
bng_pharynx	7322	1000000	11
bng_pwLinear	7325	177147	10
fried	4885	40768	10
house_16H	4893	22784	16
house_8L	2309	22784	8
houses	5165	20640	8
mv	4774	40768	10
poker	10102	1025010	10
pol	2292	15000	48

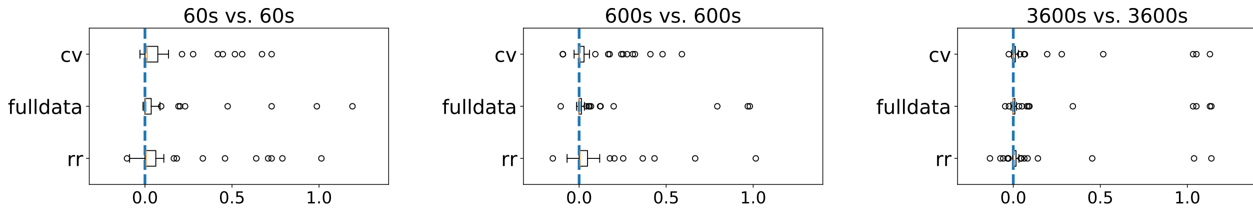


Figure 8. Score difference for FLAML vs. its own alternatives.

Table 9. % of tasks where FLAML has better or matching error vs. the baselines when using smaller time budget.

FLAML vs. Baseline	1m vs 10m	10m vs 1h	1m vs 1h
FLAML vs. Auto-sklearn	65%	79%	58%
FLAML vs. Cloud-automl	62%	79%	48%
FLAML vs. HpBandSter	63%	89%	65%
FLAML vs. H2OAutoML	71%	72%	50%
FLAML vs. TPOT	83%	85%	65%

as possible. When abundant cores are available and a learner cannot consume all of them, we can extend FLAML to work with multiple search threads in parallel. After choosing one learner based on ECI to perform one search iteration, if there are extra available resources, we can sample another learner by ECI, and so on. When one search iteration for a learner finishes, the resource is released and we can select a learner again using updated ECIs. One learner can also have multiple search threads by using different starting points of the hyperparameters. Due to the search space decomposition and the randomized direct hyperparameter search strategy used, the multiple search threads are largely independent and do not interfere with each other.

Stacked ensemble can be added as a post-processing step like existing libraries (H2O.ai). It requires remembering the predictions on cross-validation folds of the models to ensemble. And extra time needs to be spent on building the ensemble and retraining each model. FLAML does not do it by default to keep the overhead low, but it offers the option to enable it when storage and extra computation time are not concerns.

Richer types of ML tasks or model assessment criteria can be allowed via customized learners and metric functions. For example, one may search for the cheapest model with error below a threshold using our framework.

In Table 9, each row shows the percentage of datasets where FLAML is better than or equal to a particular baseline with smaller time budget. For example, '1m vs 10m' in the header means FLAML's time budget is one minute and the concerned baseline's time budget is ten minutes. We use a tolerance ratio of 0.1% to exclude the marginal differences on the scaled scores, i.e., when the difference between two scores is within the tolerance ratio, they are considered as close enough. FLAML's performance in one *minute* is already better than or equal to auto-sklearn, H2O AutoML and TPOT's performance in one *hour* on more than half of the tasks.