`ChaCha` for Online AutoML

Qingyun Wu [1] **Chi Wang** [1] **John Langford** [1] **Paul Mineiro** [1] **Marco Rossi** [1]

Abstract

We propose the `ChaCha` (Champion-Challengers) algorithm for making an *online* choice of hyperparameters in online learning settings. `ChaCha` handles the process of determining a champion and scheduling a set of 'live' challengers over time based on sample complexity bounds. It is guaranteed to have sublinear regret after the optimal configuration is added into consideration by an application-dependent oracle based on the champions. Empirically, we show that `ChaCha` provides good performance across a wide array of datasets when optimizing over featurization and hyperparameter decisions.

1. Introduction

Motivation Online learning services (for example the Personalizer Azure cognitive service (Per, 2019)) have a natural need for "autoML" style learning which automatically chooses hyperparameter configurations over some set of possible choices. The well-studied setting of offline autoML strategies (Bergstra et al., 2015; Feurer et al., 2015; Li et al., 2017; Falkner et al., 2018; Huang et al., 2019; Elsken et al., 2019; Real et al., 2020) do not satisfy several natural constraints imposed by the online setting.

1. In online settings, computational constraints are more sharply bounded. Partly, this is for budgetary concerns and partly this is because any functioning online system must keep up with a constantly growing quantity of data. Hence, we seek approaches which require at most a constant factor more computation and more generally requires that the online learning algorithm keeps up with the associated data stream.
2. Instead of having a fixed dataset, online learning settings naturally have a specified data source with unbounded, and sometimes very fast, growth. For example, it is natural to have datasets growing in the terabytes/day range. On the other hand, many data sources are also at much lower volumes, so we naturally seek approaches which can handle vastly different scales of data volume.
3. Instead of evaluating the final quality of the model produced by a learning process, online learning settings evaluate a learning algorithm constantly, implying that an algorithm must be ready to answer queries at all times, and that the relevant performance of the algorithm is the performance at all of those evaluations.

It is not possible to succeed in online autoML by naively applying an existing offline autoML algorithm on the data collected from an online source. First, this approach does not address the computational constraint, as direct use of offline autoML is impractical when the dataset is terascale or above. Operating on subsets of the data would be necessary, but the dramatic potential performance differences in learning algorithms given different dataset sizes suggests that the choice of subset size is critical and data-dependent. Automating that choice is non-trivial in general. Second, this approach does not address the issue of online evaluation, as offline autoML algorithms are assessed on the quality of the final configuration produced. It is also not possible to succeed in online autoML via naive application of existing regret-minimizing online algorithms (Cesa-Bianchi & Lugosi, 2006): Instantiating a no-regret algorithm over all sets of possible configurations is computationally prohibitive.

How then can we best design an *efficient* online automated machine learning algorithm?

In the online setting there are no natural points for stopping training, evaluating a configuration, and trying the next configuration. If we keep evaluating a fixed set of configurations, other configurations are denied experience, which could lead to linearly increasing total regret. Exploration is however delicate because an unknown quantity of data may be necessary for a configuration to exhibit superior performance. Given this, a simple exploration approach which periodically switches between configurations in a batched epsilon-greedy style may also incur large total regret. A method that can allocate the limited computational power (at any time point) to learning models while maintaining good online performance (i.e., low regret) and working despite an unknown required example threshold is needed.

[1]Microsoft Research. Correspondence to: Qingyun Wu <qxw5138@psu.edu>, John Langford <jcl@microsoft.com>.

Proceedings of the 38th International Conference on Machine Learning, PMLR 139, 2021. Copyright 2021 by the author(s).

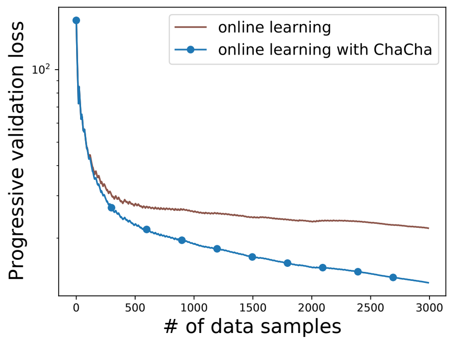


Figure 1. ChaCha improves online learning - a demonstration.

1.1. What we do

In Section 2, we define a new *online* automated machine learning setting with tight computational constraints, consistent with standard online learning settings. A key element in this setting is a multiplicative constant on the amount of computation available which corresponds to the maximum number of "live" models at any time point. In addition, we assume no model persistence is allowed. The other key element is the availability of a configuration oracle which takes as input a configuration and provides as output a set of possible configurations to try next. This oracle is designed to capture the ability of a domain expert or offline autoML methods to propose natural alternatives which may yield greater performance.

The oracle could propose more configurations than can be simultaneously assessed given the computational budget. How can we efficiently discover a better configuration while respecting the computational budget? It is critical to allocate computation across the possibilities in a manner which both does not starve potentially best possibilities and which does not waste either computation or experience. The ChaCha algorithm detailed in Section 3 is designed to do this using an amortized computation approach. Two other critical issues exist: How can a better configuration be reliably identified? And when evaluating multiple (potentially disagreeing) alternatives how should we make predictions? For the former, we use progressive validation sample complexity bounds while for the latter we identify a model using these bounds and predict according to it, providing benefit from a potentially better configuration before it is proved as such.

In Section 4 we analyze theoretical properties of ChaCha. We prove that as long as the oracle can successfully suggest the optimal configuration, ChaCha can always achieve a sublinear regret bound even in the worst case. In some ideal cases it can achieve a regret bound that matches the regret bound of the optimal configuration. This sanity-check analysis provides some reassurance that the algorithm behaves reasonably in many settings.

We test the ChaCha algorithm on a suite of large regression datasets from OpenML (Vanschoren et al., 2014) for two online autoML tasks. Figure 1 shows a demonstrative result obtained by ChaCha for tuning features interactions choices, eclipsing a widely used online learning algorithm. Further experimentation demonstrates ChaCha is consistently near-best amongst plausible alternatives.

1.2. Related work

There are many interesting autoML systems designed for the offline setting. A few representative examples from academia are Auto-sklearn (Feurer et al., 2015), Hyperopt (Bergstra et al., 2015), Hyperband (Li et al., 2017) and FLAML (Wang et al., 2021b). There are also many end-to-end commercial services such as Amazon AWS SageMaker, DataRobot, Google Cloud AutoML Tables, Microsoft AzureML AutoML and H2O Driverless AI. Notable hyperparameter optimization methods include: random search (Bergstra & Bengio, 2012; Li et al., 2017), Bayesian optimization (Snoek et al., 2012; Bergstra et al., 2015; Feurer et al., 2015; Falkner et al., 2018), local search (Koch et al., 2018; Wu et al., 2021) and methods that combine local and global search (Eriksson et al., 2019; Wang et al., 2021a). In addition, progressively increasing resource and early stopping methods (Li et al., 2017; Falkner et al., 2018; Huang et al., 2019) are usually found useful when training is expensive. Despite the extensive study, none of these methods is directly applicable to performing efficient autoML in the online learning setting.

An incremental data allocation strategy, which is conceptually similar to the adaptive resource scheduling strategy in this work, for learner selection is proposed in (Sabharwal et al., 2016). However, the method is designed to function only in the offline batch learning setting with a small fixed set of learners. Other research directions that share strong synergy with autoML in the online learning literature include parameter-free online learning (Chaudhuri et al., 2009; Luo & Schapire, 2015; Orabona & Pál, 2016; Foster et al., 2017; Cutkosky & Boahen, 2017) and online model selection (Sato, 2001; Muthukumar et al., 2019). Many of these works provide theoretical foundations for online model selection in different learning environments, including both probabilistic and adversarial. However, most of these works only address specific hyper-parameters, such as the learning rate. In addition, these works overlook the sharp constraints on computational power encountered in practice. (Dai et al., 2020) views a model in the context of a bigger system with a target of actively controlling the data collection process for online experiments, which is different from the goal of this work.

2. Learning Setting

Several definitions are used throughout. Examples are drawn i.i.d. from a data space $\mathcal{X} \times \mathcal{Y}$ with $\mathcal{X}$ the input domain and $\mathcal{Y}$ the output domain. A function $f : \mathcal{X} \rightarrow \mathcal{Y}$ maps input features to an output prediction. A learning algorithm $A : \mathcal{X} \times (\mathcal{X} \times \mathcal{Y})^* \rightarrow \mathcal{Y}$ maps a dataset and a set of input features to a prediction. A loss function $l : \mathcal{Y} \times \mathcal{Y} \rightarrow \mathbb{R}$ defines a loss for any output and prediction. $L_f := \mathbb{E}_{(X,Y)}[l(f(X), Y)]$ denotes the true loss of hypothesis f (under the i.i.d. assumption). $L^*_{\mathcal{F}} := \min_{f \in \mathcal{F}} \mathbb{E}[l(f(\mathbf{x}_t), y_t)]$ denotes the best loss achievable using the best fixed choice of parameters in a function class $\mathcal{F}$, which contains a set of functions. f^* is the best function given loss function l and the data distribution.

We consider the following online learning setting: at each interaction t, the learner receives a data sample X from the input domain, and then makes a prediction of the sample $A(X, D_t)$ based on knowledge from the historical data samples D_t. After making the prediction, the learner receives the true answer Y from the environment as a feedback. Based on the feedback, the learner measures the loss, and updates its prediction model by some strategy so as to improve predictive performance on future received instances. In such an online learning setting, we want to minimize the cumulative loss $\sum_{t=1}^{T} L_{A(\cdot, D_t)}$ from the online learner A over the whole interaction horizon T compared to the loss of the best function f^*. The gap between them is called 'regret' and defined as $R(T) := \sum_{t=1}^{T}(L_{A(\cdot,D_t)} - L_{f^*})$.

Similar to the case in offline machine learning, hyperparameters are also prevalent in online learning. A naive choice of the configuration $\tilde{c}$ may lead to a function class that does not contain the best function. In this case $R(T) := \sum_{t=1}^{T}(L_{\tilde{c},t} - L_{f^*}) \geq \sum_{t=1}^{T}(L^*_{\mathcal{F}_{\tilde{c}}} - L_{f^*}) = \Theta(T)$, i.e., linearly increasing regret is inevitable if the wrong hyperparameter configuration is used. We propose online autoML to make *online* choices of the configurations drawn from a configuration space $\mathcal{C}$. For a particular configuration $c \in \mathcal{C}$, we denote by A_c the corresponding online learner instance and $\mathcal{F}_c$ the corresponding function class for A_c. We denote by $D_{t,c}$ the data samples received by A_c up to time t. We denote $L_{c,t} := L_{A_c(\cdot, D_{t,c})}$. We choose $c_t \in \mathcal{C}$ for prediction at time t while operating online autoML. Correspondingly, the regret can be rewritten as $R(T) := \sum_{t=1}^{T}(L_{c_t,t} - L_{f^*})$.

To account for practical online learning scenarios, we target the online setting where (1) a maximum number of b "live" models are allowed to perform online learning at the same time; and (2) no model persistence or offline training is allowed, which means that once we decide to replace a 'live' model with a new one, the replaced model can no longer be retrieved. After each model replacement, the new live model needs to learn from scratch, i.e., $D_{t,c}$ is empty when a model is not live.

3. Method

Solving the online autoML problem requires finding a balance between searching over a large number of plausible choices and concentrating the limited computational budget on a few promising choices such that we do not pay a high 'learning price' (regret). `ChaCha` (short for **Cha**mpion-**Cha**llengers) is designed to this end with two key ideas: (1) a progressive expansion of the search space according to the online performance of existing configurations; (2) an amortized scheduling of the limited computational resources to configurations under consideration.

To realize these two ideas, we first categorize configurations under consideration in our method into one *Champion*, denoted by C, and a set of *Challengers*, denoted by $\mathcal{S}$. Intuitively, the *Champion* is the best proven configuration at the concerned time point. The rest of the candidate configurations are considered as *Challengers*. `ChaCha` starts by setting the initial or default configuration, denoted by c_{init} as the champion, and starts with an empty challenger set, i.e, $\mathcal{S} = \emptyset$ initially. As online learning goes, it (1) updates the champion when necessary and adds more challengers progressively; (2) assigns one of the b slots for 'live' models to the champion, and does amortized scheduling of the challengers for the remaining $b - 1$ slots when the number of challengers is larger than $b - 1$. Under this framework, in the special case where $b = 1$, our method degenerates to a vanilla online learning algorithm using the default configuration. In the case where $b > 1$, we are able to evaluate more challengers, which gives us a chance to find potentially better configurations. With b 'live' models running, `ChaCha` at each iteration selects one of them to do the final prediction. We present the framework of `ChaCha` in Figure 2 and provide the pseudocode description in Algorithm 1.

3.1. Progressive search space construction in `ChaCha`

Generating challengers with champion and `ConfigOracle`. `ChaCha` assumes the availability of a `ConfigOracle`. When provided with a particular input configuration c, `ConfigOracle` should produce a candidate configuration set that contains at least one configuration that is significantly better than the input configuration c each time a new configuration is given to it. Such a `ConfigOracle` is fairly easy to construct in practice. Domain expertise or offline autoML search algorithms can be leveraged to construct such a `ConfigOracle`. For example, when the configurations represent feature interaction choices, one typically effective way to construct the oracle is to add pairwise feature interactions as derived features based on the current set of both original and derived features (Luo et al., 2019). With the availability of such a `ConfigOracle`, we use a Champion as the 'seed' to the `ConfigOracle` to construct a search space which

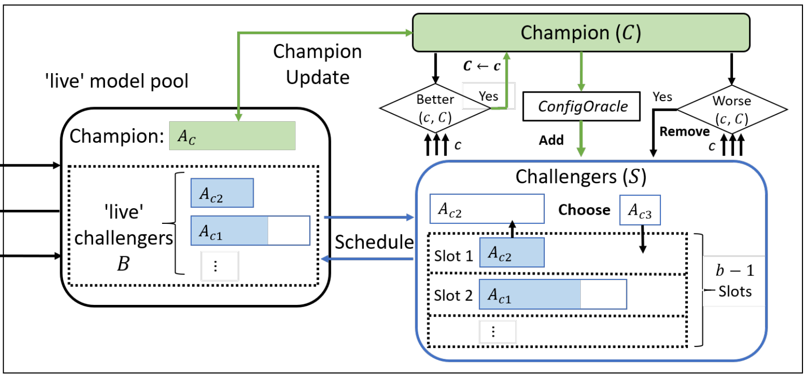


Figure 2. ChaCha online autoML framework. The arrows in black are executed at every iteration of online learning. The arrows in green show the operations that are triggered once the *Champion* is updated, and the arrows in blue show the scheduling of 'live' *challengers*. Each 'live' challenger is represented by a blue rectangle with shaped area, the length of which represents the assigned resource lease. The shaded area of each rectangle reflects how many samples are consumed by each corresponding learner.

is then expanded only when a new Champion is identified.

A progressive update of the champion using statistical tests. ChaCha updates the champion when a challenger is proved to be 'sufficiently better' than it. We use statistical tests with sample complexity bounds to assess the true quality of a configuration and promote new champions accordingly. The central idea of the statistical tests is to use sample complexity bounds and empirical loss to assess the 'true' performance $L^*_{\mathcal{F}_c}$ of the configuration c through a probabilistic lower and upper bound, denoted by $\underline{L}_{c,t}$ and $\overline{L}_{c,t}$ respectively.

Since we consider the online learning setting, we use progressive validation loss (Blum et al.) as the empirical loss metric. For a given configuration c, we denote by $L^{PV}_{c,t}$ the progressive validation loss of A_c on data sequence $D_{t,c}$.

$$L^{PV}_{c,t} := \frac{1}{|D_{t,c}|} \sum_{(X_i,Y_i)\in D_{t,c}} l(A(X_i, D_{t,c}[: i-1]), Y_i)$$

Without loss of generality, we assume the learning algorithm A can ensure that, for any $\delta > 0$, with probability at least $1-\delta, \forall t, |L^{PV}_{c,t} - L^*_{\mathcal{F}_c}| \leq (\texttt{comp}_{\mathcal{F}_c} \log(|D_{t,c}|/\delta)|D_{t,c}|^{p-1})$ where $\texttt{comp}_{\mathcal{F}_c}$ denotes a term related to the complexity of the function class $\mathcal{F}_c$, and $0 < p < 1$ characterizes the estimator error's dependency on the number of data samples. For example, one typical result is $p = \frac{1}{2}, \texttt{comp}_{\mathcal{F}} = \sqrt{d}$ for the d-dimensional linear regression functions (Shalev-Shwartz & Ben-David, 2014).

Based on the bound specified above and an union bound, we can obtain a probabilistic error bound $\epsilon_{c,t}$ which holds for all t and for all $c \in \mathcal{S}_t$. More formally, with probability at least $1-\delta, \forall t, \forall c \in \mathcal{S}_t, |L^{PV}_{c,t} - L^*_{\mathcal{F}_c}| \leq \epsilon_{c,t}$ with

$$\epsilon_{c,t} := \texttt{comp}_{\mathcal{F}_c}(\log(|D_{t,c}||\mathcal{S}_t|/\delta))|D_{t,c}|^{p-1}) \quad (1)$$

$\overline{L}_{c,t} := L^{PV}_{c,t} + \epsilon_{c,t}$ and $\underline{L}_{c,t} := L^{PV}_{c,t} - \epsilon_{c,t}$ are thus probabilistic upper and lower bounds of $L^*_{\mathcal{F}_c}$ for $t \in [T], c \in \mathcal{S}_t$. Based on these probabilistic bounds, we define the following two tests to help distinguish which configuration can lead to better (true) performance.

$$\texttt{Better}(c, C, t) := \mathbb{1}\{\overline{L}_{c,t} < \underline{L}_{C,t} - \epsilon_{C,t}\} \quad (2)$$

$$\texttt{Worse}(c, C, t) := \mathbb{1}\{\underline{L}_{c,t} > \overline{L}_{C,t}\} \quad (3)$$

ChaCha eliminates a challenger from consideration once the result of Worse test is positive and promotes a challenger to the new champion once the result of a Better test is positive (line 11-15 in Algorithm 1). When a new champion is promoted, a series of subsequent operations are triggered, including (a) an update of the ChaCha's champion and (b) a call of the ConfigOracle which generates a new set of challengers to be considered. These two operations are reflected in line 8 and line 16-17 of Algorithm 1 and the green arrows in Figure 2.

Note that when testing whether a challenger c should be promoted into a new champion using the Better test specified in Eq. (2), we require the gap between the lower and upper bounds to be at least $\epsilon_{C,t}$. This ensures that a challenger is promoted into a champion only when it is 'sufficiently' better than the old champion, a strategy which avoids the situation where we keep switching champions that are only slightly better than the old ones. That situation is undesirable for two reasons: (a) it does not guarantee any lower bound on the loss reduction and thus the true loss between the champion and the true best configuration may remain larger than a constant, which causes a linearly increasing regret in the worst case, and (b) since new challengers are generated and added into consideration, it makes the challenger pool unnecessarily large.

In summary, with the help of ConfigOracle and the statistical tests, ChaCha is able to maintain a set of challengers that are not yet 'sufficiently better' than the champion but also not significantly 'worse' than the champion.

Algorithm 1 ChaCha

```
1: Inputs: Initial configuration $c_{init}$, budget $b$.
2: Initialization: $C \leftarrow c_{init}$; $\mathcal{S} \leftarrow$ ConfigOracle($C$); $\hat{c} \leftarrow C$; $\mathcal{B} = \emptyset$
3: for t = 1, ... do
4:     $\triangleright$ Observe $X_t$
5:     $\mathcal{B} \leftarrow$ Schedule($b, \mathcal{B}, \mathcal{S}$)
6:     Predict $\hat{Y}_t \leftarrow A_{\hat{c}}(X_t, D_{t,\hat{c}})$, in which $\hat{c} \leftarrow \arg\min_{c \in \mathcal{B}+C} \overline{L}_{\mathcal{F}_c}$
7:     $\triangleright$ Receive $Y_t$ and incur loss $l(\hat{Y}_t, Y_t)$
8:     for $c \in \mathcal{B} + C$ do
9:         $\mathcal{D}_{t+1,c} \leftarrow \mathcal{D}_{t,c} + (X_t, Y_t)$ and update $A(\cdot, \mathcal{D}_{t+1,c})$
10:    $C_{old} \leftarrow C$
11:    for $c \in \mathcal{S}$ do
12:        if Better($c, C, t$) then
13:            $C \leftarrow c$
14:        if Worse($c, C, t$) then
15:            $\mathcal{S} \leftarrow \mathcal{S} - c$
16:    if $C \neq C_{old}$ then
17:        $\mathcal{S} \leftarrow \mathcal{S} +$ ConfigOracle($C$)
```

3.2. 'Live' challenger scheduling

Now that we have a set of challengers, if the number of 'live' model slots is larger than the number of challengers (either because we have a large b or because we have a small $|\mathcal{S}|$), we can evaluate all the challengers simultaneously. Otherwise we need to perform a careful scheduling. The scheduling problem is challenging since: (1) no model persistence is allowed in the online learning setting so frequent updates of the 'live' challengers is costly in terms of learning experience; (2) a blind commitment of resources to particular choices may fail due to those choices yielding poor performance. One principled way to amortize this cost is to use the doubling trick when allocating the sample resource: assign each challenger an initially small lease and successively double the lease over time. We adopt this amortized resource allocation principle and also add a special consideration of the challengers' empirical performance while doing the scheduling. The scheduling step is realized through a Schedule function in ChaCha.

Specifically, the Schedule function takes as input the budget b, the current 'live' challenger set $\mathcal{B}$, the candidate set $\mathcal{S}$, and provides as output a new set of live challengers (which can have overlap with the input $\mathcal{B}$). It is designed to eventually provide any configuration with any necessary threshold of examples required for a regret guarantee. We call this resource threshold a *resource lease* denoted by $\overline{n}_c$. Initially every configuration is assigned a particular minimum resource lease $\overline{n}_c = n_{min}$ (for example $n_{min} = 5 \times \#features$). When a configuration has been trained with $\overline{n}_c$ examples (line 6 in Algorithm 2), i.e., reaches its assigned resource lease, we double this resource lease (line 7 in Algorithm 2).

To avoid starving a challenger under consideration indefinitely, we remove the challenger which just reached its assigned resource lease, from the 'live' challenger pool and add the challenger with the minimum resource lease into the 'live' challenger pool (line 9 and line 11-13 of Algorithm 2 and Algorithm 3). In addition, to avoid throwing away valuable experience for a promising challenger, we adopt a more conservative exploration: We replace a 'live' challenger which reaches its assigned resource lease only if it is not among the top performing (according to loss upper bound) 'live' challengers (line 8 in Algorithm 2). In other words, we use half of the compute resources to exploit the candidates that have good performance for now, and another half to explore alternatives that may have better performance if given more resources.

With the b 'live' models running, at each interaction, ChaCha selects one of them to make the prediction (line 6 of Algorithm 1) following the structural risk minimization principle (Vapnik, 2013).

Algorithm 2 Schedule($b, \mathcal{B}, \mathcal{S}$)

```
1: Notions: $\overline{n}_c$ denotes the resource lease assigned to for $c$; $n_{c,t}$ denotes the resource consumed by $A_c$ by time $t$, e.g., $n_{c,t} = |D_{t,c}|$
2: for $c \in \mathcal{B}$ do
3:     if $c \notin \mathcal{S}$ and $|\mathcal{S}| > b$ then
4:         $\mathcal{B} \leftarrow \mathcal{B} - c$
5: for $c \in \mathcal{B}$ do
6:     if $n_{c,t} \geq \overline{n}_c$ then
7:         $\overline{n}_c \leftarrow 2\overline{n}_c$
8:         if $\overline{L}_{c,t} >$ median($\{\overline{L}_{c,t}\}_{c \in \mathcal{B}}$), and $|\mathcal{S}| > b$ then
9:             $\mathcal{B} \leftarrow \mathcal{B} - c$
10: while $|\mathcal{B}| < b - 1$ do
11:    $c \leftarrow$ Choose($\mathcal{S} \setminus \mathcal{B}$)
12:    $\mathcal{B} \leftarrow \mathcal{B} + c$
13:    $D_{t+1,c} \leftarrow \emptyset$ {Assuming no persistence of models}
14: return $\mathcal{B}$
```

Algorithm 3 Choose($\mathcal{S}$)

```
1: $\mathcal{C}^{\text{Pending}} \leftarrow \{c \in \mathcal{S} : \overline{n}_c$ has not been set$\}$
2: if $|\mathcal{C}^{\text{Pending}}| \neq 0$ then
3:     $c \leftarrow$ Random($\mathcal{C}^{\text{Pending}}$) and set $\overline{n}_c \leftarrow n_{\min}$
4:     return c
5: return $\arg\min_{c \in \mathcal{S}} \overline{n}_c$
```

4. Theory

For the convenience of analysis, we split the total time horizon T into $M+1$ phases, the index of which ranges from 0 to M. Phase 0 starts from $t=1$ and a new phase starts once a new champion is found. We denote by t_m and $t_{m+1}-1$ the starting and end time index of phase m respectively. Thus $N_m := t_{m+1} - t_m$ is the length of phase m. We denote by $\mathcal{S}_m$ the set of candidate configurations and C_m the champion configuration at phase m.

$$\underbrace{1, 2, \cdots, \cdots t_1 - 1}_{\text{phase } 0}, \cdots, \underbrace{t_m, \cdots, t_{m+1} - 1}_{\text{phase } m}, \cdots, \underbrace{t_M, \cdots, T}_{\text{phase } M}$$

We denote by c^* the optimal configuration, and thus $L_{f^*} = L^*_{\mathcal{F}_{c^*}}$. To make meaningful analysis, we assume c^* is added in the candidate configuration pool after the `ConfigOracle` is called a constant number of times.

Lemma 1 *With $\epsilon_{c,t}$ being set as in Eq. 1 and $0 < \delta < 1$, $\forall m \in [M]$,*

Claim 1. *$\forall c \in \mathcal{S}_m$, if $L^*_{\mathcal{F}_c} - L^*_{\mathcal{F}_{C_m}} + 2\epsilon_{c,t} + 3\epsilon_{C_m,t} < 0$, with probability at least $1-\delta$, c can pass the `Better` test described in Eq. (2) when compared with C_m at time t.*

Claim 2. *When the `Better` test is positive, with probability at least $1-\delta$, $L^*_{\mathcal{F}_{C_m}} - L^*_{\mathcal{F}_{C_{m+1}}} > \epsilon_{C_m, t_{m+1}}$.*

Claim 3. *If $L^*_{\mathcal{F}_c} < L^*_{\mathcal{F}_{C_m}}$, with probability at least $1-\delta$, c will not pass the `Worse` test when compared with C_m.*

Proposition 1 *With a base learning algorithm having a sample complexity based error bound of $comp_{\mathcal{F}_c} \log(|D_{t,c}|/\delta)|D_{t,c}|^{p-1}$ and $\epsilon_{c,t}$ being set as in Eq. (1), with high probability, `ChaCha` can obtain,*

$$\sum_{m=0}^{M} \sum_{t=t_m}^{t_{m+1}-1} (L^*_{\mathcal{F}_{C_m}} - L^*_{\mathcal{F}_{c^*}}) \tag{4}$$
$$= \tilde{O}\Big(\max_{m \in [M]} \frac{|\mathcal{S}_m|}{b} comp_{\mathcal{F}_{c^*}} T^p + \sum_{m=0}^{M} comp_{\mathcal{F}_{C_m}} N_m^p \Big)$$
$$= \tilde{O}\Big(\max_{m \in [M]} \frac{|\mathcal{S}_m|}{b} comp_{\mathcal{F}_{c^*}} T^p + comp_{\mathcal{F}_{C_m}} T^{\frac{1}{2-p}} \Big)$$

Proof intuition of Proposition 1. The proof is mainly based on the first two claims of Lemma 1. Claim 1 of Lemma 1 ensures that, for all phases $m \in [M]$, the gap between 'true' loss of the champion and the best configuration in the pool is with high probability upper bounded by $\epsilon_{c^*,t}$ and $\epsilon_{C_m,t}$. Since $\epsilon_{c,t}$ shrinks with the increase of data samples for c, and our scheduling strategy ensures that the optimal configuration c^* receives at least $\frac{b}{4 \max_{m \in M} |\mathcal{S}_m|} T$ data samples, we can obtain an upper bound of $\tilde{O}(\max_{m \in [M]} \frac{|\mathcal{S}_m|}{b} \mathrm{comp}_{\mathcal{F}_{c^*}} T^p)$ for the summation term on $\epsilon_{c^*,t}$ over T iterations. The summation over $\epsilon_{C_m,t}$, i.e., $\sum_{m=0}^{M} \sum_{t=t_m}^{t_{m+1}-1} \epsilon_{C_m,t}$ is delicate because we must account for the case where the champion is updated frequently, which makes M and $\epsilon_{C_m,t}$ large. Fortunately, we designed the `Better` test such that we switch to a new champion only when it is sufficiently better than the old one which ensures an upper bound on the number of switches, i.e., M. Specifically, with Claim 2 of Lemma 1, and the fact that $\sum_{m=0}^{M} N_m = T$, we can prove $M = O(T^{\frac{1-p}{2-p}})$. With that we are able to prove $\sum_{m=0}^{M} \sum_{t=t_m}^{t_{m+1}-1} \epsilon_{C_m,t} = \tilde{O}(\sum_{m=0}^{M} N_m^p) = \tilde{O}(T^{\frac{1}{2-p}})$, which contributes to the second term of the bound in Eq. (4).

Remark 1 (Special cases of Proposition 1) *Proposition 1 provides an upper bound of $\sum_{m=0}^{M} \sum_{t=t_m}^{t_{m+1}-1} (L^*_{\mathcal{F}_{C_m}} - L^*_{\mathcal{F}_{c^*}})$ in the most general case. It provides guarantees even for the worst cases where the algorithm needs to pay high switching costs, i.e., with a large number of phases M. Such bad cases happen when the algorithm frequently finds new Champions from newly added candidates, whose accumulated sample number is still small and thus $|D_{t_{m+1}, C_m}|$ can only be lower bounded by 1, $\forall m \in [M]$. In the special case where $|D_{t_{m+1}, C_m}| \geq z |D_{t_m, C_{m-1}}|$, in which z is a constant larger than 1 for all m, we have $\sum_{m=0}^{M} N_m^p \log N_m = O(T^p \log T)$, which reduces the upper bound in Proposition 1 to $\tilde{O}(T^p)$. In this special case, the bound matches the order of regret (w.r.t. T) for using the optimal configuration.*

Without further assumptions, Proposition 1 is tight. For any small $\eta > 0$, we can construct a sequence of $N_m = \Theta(m^q)$, where $q = \frac{1-p}{\frac{1}{2-p} - p - \eta} - 1 = \frac{(1-p)(2-p)}{1-(2-p)(p+\eta)} - 1 > \frac{2-p}{1-p} - 1 = \frac{1}{1-p}$. Such a sequence satisfies Eq. (9), and makes $\sum_{m=0}^{M} N_m^p \log N_m = \Omega(T^{\frac{1}{2-p} - \eta})$.

Theorem 1 *Under the same conditions as specified in Proposition 1, `ChaCha` can achieve the following regret bound, $\sum_{t=1}^{T} (L_{\hat{c}_t, t} - L^*_{\mathcal{F}_{c^*}}) = \tilde{O}\big(\max_{m \in [M]} \frac{|\mathcal{S}_m|}{b} comp_{\mathcal{F}_{c^*}} T^p + comp_{\mathcal{F}_{C_m}} T^{\frac{1}{2-p}} \big)$.*

Proof intuition of Theorem 1. We decompose the regret in the following way $\sum_{t=1}^{T} (L_{\hat{c}_t, t} - L^*_{\mathcal{F}_{c^*}}) = \sum_{m=0}^{M} \sum_{t=t_m}^{t_{m+1}-1} (L_{\hat{c}_t, t} - L^*_{C_m}) + \sum_{m=0}^{M} \sum_{t=t_m}^{t_{m+1}-1} (L^*_{C_m} - L^*_{\mathcal{F}_{c^*}})$, the bound of which is quite straightforward to prove once the conclusion in Proposition 1 is obtained.

5. Empirical Evaluation

`ChaCha` is general enough to handle the online tuning of various types of hyperparameters of online learning algorithms in a flexible and extensible way. It can be tailored to satisfy customized needs through customized implementations of the `ConfigOracle`.

Out-of-the-box `ChaCha` with a default `ConfigOracle` We provide a default implementation of the `ConfigOracle` such that `ChaCha` can be used as an out-of-the-box solution for online autoML. This default `ConfigOracle` leverages an existing offline hyperparameter optimization method (Wu et al., 2021) for numerical and categorical hyperparameters suggestion. In addition to these two types of typically concerned hyperparameters, we further extend the `ConfigOracle` by including another important type of hyperparameter, whose configurations space can be expressed as a set of polynomial expansions generated from a fixed set of singletons. One typical example of this type of hyperparameter is the choice of feature interactions or feature crossing on tabular data (Luo et al., 2019), where raw features (or groups of raw features) are interacted, e.g., through cross product, to generate an additional set of more informative features. In this example, the set of raw feature units are the set of singletons, and each resulting feature set with additional feature interactions can be treated as one polynomial expansion based on the original singletons. This type of featurization related hyperparameter is of particular importance in machine learning practice: it has been well recognized that the performance of machine learning methods, including both offline batch learning and online learning, depends greatly on the quality of features (Domingos, 2012; Agarwal et al., 2014). The tuning of this type of hyperparameter is notoriously challenging because of the doubly-exponentially large search space: for a problem with m raw singletons, each interaction (up to a maximum order of m) involves any subset of the m raw singletons, implying a $(\sum_{i=0}^{m}\binom{m}{i} - m - 1 = 2^m - m - 1)$-dimensional binary vector space defining the set of possible interactions. Since any subset of the polynomial expansions is allowed, there are 2^{2^m-m-1} possible choices. For this type of hyperparameter, only heuristic search approaches such as Beam search (Medress et al., 1977), are available even in the offline learning setting. Inspired by these offline greedy search approaches and domain knowledge obtained from online learning practice, we realize the `ConfigOracle` for this type of hyperparameter in the following greedy approach by default in `ChaCha`: given the input configuration, `ConfigOracle` generates all configurations that have one additional second order interaction on the input configuration. For example, given a input configuration $C = \{e_1, e_2, e_3\}$, the `ConfigOracle`(C) outputs the following set of configurations $\{\{e_1, e_2, e_3, e_1e_2\}, \{e_1, e_2, e_3, e_1e_3\}, \{e_1, e_2, e_3, e_2e_3\}\}$. Under this design, when provided with an input configuration with k groups of features, the `ConfigOracle` generates a candidate set with $\frac{k(k-1)}{2}$ configurations.

5.1. AutoML tasks

Online Learning with Vowpal Wabbit. Our evaluation is performed using Vowpal Wabbit[1] (VW), which is an open-source online machine learning library. Users can tune various hyperparameters of online learning algorithms in VW depending on their needs, for example, namespaces interactions, learning rate, l1 regularization and etc.

Target hyperparameters to tune. We perform evaluation in two tuning scenarios to demonstrate both the effectiveness and the flexibility of `ChaCha`, including the tuning of namespaces interactions (a namespace is essentially a group of features), and the tuning of both namespace interactions and learning rate. The automatic tuning of namespace interactions is of significant importance in VW because (a) it can greatly affect the performance; (b) the challenge exists no matter what online learning algorithms one use; (c) it is not well handled by any of the parameter-free online learning algorithms developed recently. For these two tuning tasks, we use a default implementation of the `ConfigOracle` described earlier in this section. We use the default configuration in VW as the the initial configuration c_{init}: no feature interactions, and the learning rate is 0.5.

We use the VW default learning algorithm (which uses a variant of online gradient descent) as the base learner. We perform the main evaluation under the constraint that a maximum of 5 'live' learners are allowed, i.e., $b = 5$.

Baselines and Comparators. No existing autoML algorithm is designed to handle the online learning scenario, so we compare our method with several natural alternatives.

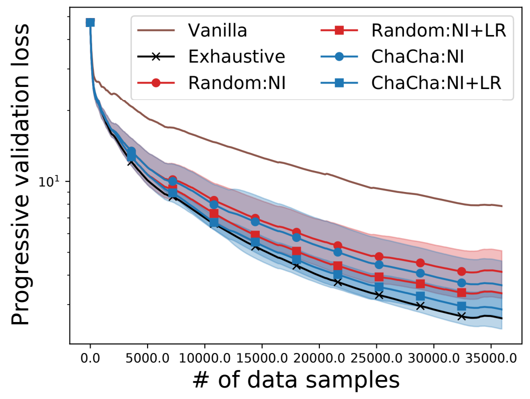
- `Random`: Run learners built on the initial configuration and $(b-1)$ randomly selected configurations from the first batch of candidate configurations generated by the same `ConfigOracle` used in `ChaCha`.

- `Exhaustive`: Exhaust all the configurations generated from `ConfigOracle`(c_{init}) ignoring the computational limit on the number of 'live' models. This should provide better performance than `ChaCha` on the first batch of configurations generated from the `ConfigOracle` with the initial configuration since it removes the computational limits of `ChaCha`. It is possible for `ChaCha` to perform better despite a smaller computational budget by moving beyond the initial set of configurations.

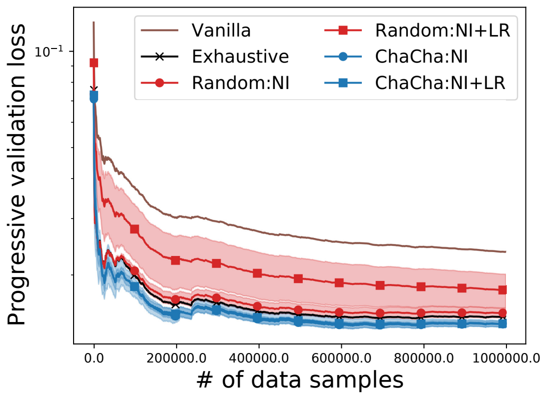
- `Vanilla`: Run the learner built on the initial configuration using c_{init} without further namespace interactions.

Note that `Vanilla` and `Exhaustive` are comparators to better understand the performance of the proposed method, rather than baselines in the context of online autoML with the same computational constraints.

[1] https://vowpalwabbit.org



(a) Progressive validation loss over time on dataset # 41506



(b) Progressive validation loss over time on dataset # 5648

Figure 3. Real time progressive validation loss on a small and large dataset. The shaded area shows the standard derivation of the loss over the 5 runs. The suffixes ':NI' and ':NI+LR' with `ChaCha` and `Random` denote the tuning task of namespace interactions, and both namespace Interactions and learning rate respectively. This legend syntax applies to all figures in this paper.

Datasets. We evaluate our method on a set of large scale (# of instance: 10K to 1M) regression datasets from OpenML (in total 40). On these datasets, we group the raw features into up to 10 namespaces. Among the 40 large scale openml datasets, 25 do not allow superior performance to `Vanilla` amongst the configurations given by `ConfigOracle`. We therefore focus on the 15 'meaningful' datasets where `Exhaustive` is better than `Vanilla`.

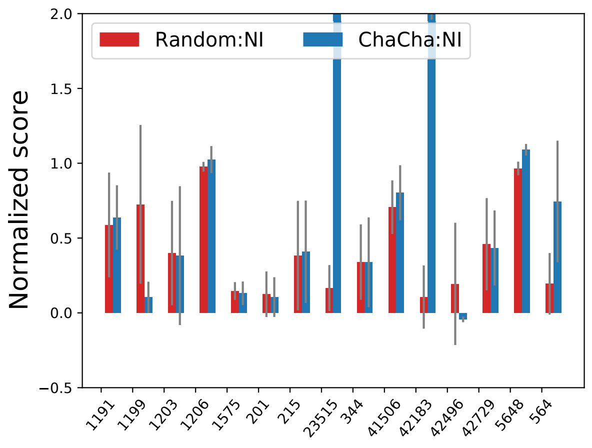
5.2. Results

Performance on OpenML datasets. Figure 3 shows typical results in terms of the progressive validation loss (mean square error) on two of the openml datasets. `ChaCha` is able to outperform all the competitors except `Exhaustive`. To show the aggregated results over all the openml datasets, we normalized the progressive validation loss of a particular algorithm using the following formula: Score(alg) := $\frac{L^{PV}(\texttt{Vanilla})-L^{PV}(\text{alg})}{L^{PV}(\texttt{Vanilla})-L^{PV}(\texttt{Exhaustive})}$. By this definition, Score(`Vanilla`) $=0$ and Score(`Exhaustive`) $=1$. The score is undefined when Score(`Exhaustive`) $=$ Score(`Vanilla`), so we only report results on the 'meaningful' datasets where the difference is nonzero. As the serving order of configurations from the candidate configuration pool is randomized, for all the experiments, we run each method 5 times with different settings of random seed (except `Vanilla` and `Exhaustive` as they are not affected by the random seeds) and report the aggregated results. Figure 4(a) and Figure 4(b) show the final normalized scores for two tuning tasks on the 15 datasets after running for up to 100K data samples (or the maximum number of data samples if it is smaller than 100K). We include the final normalized score on datasets with larger than 100K samples in the appendix. Results in Figure 4 show that `ChaCha` has significantly better performance comparing to `Random` over half of the 15 'meaningful' datasets.

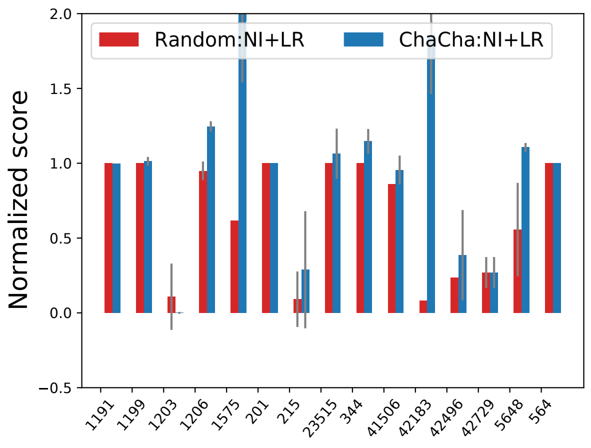
Analysis and ablation. We now investigate several important components in `ChaCha` to better understand the effectiveness of it.

(1) Search space construction in `ChaCha`. `ChaCha` uses the champion and `ConfigOracle` to gradually expand the search space, which leads to a provable improvement on the overall quality of configurations while keeping the size still manageable. The method `Random` and `Exhaustive` use a straightforward way to take advantage of the `ConfigOracle`: they consider the configurations generated by the `ConfigOracle` based on the initial configuration. `Random` works reasonably well if the configuration pool contains many configurations better than the initial configuration. However as expected, it has very large variance. In addition, it is not able to further improve beyond `Exhaustive`. The fact that in many cases, `ChaCha` has better performance than `Exhaustive` indicates that there is indeed a need for expansion of the search space. In addition to the performance boost comparing to `Exhaustive` on almost half of the datasets, `ChaCha` shows almost no performance degeneration comparing to both `Vanilla` and `Random`.

(2) The scheduling of 'live' models. Another important component of `ChaCha` is the scheduling of 'live' models. `ChaCha` always keeps the champion 'live' and schedules $b-1$ 'live' challengers in an amortized manner. When scheduling the challengers, to avoid throwing away useful experiences due to the 'live' challengers swapping, `ChaCha` keeps the top-performing half of the 'live' challengers running. Results in Figure 5 show the two variants of `ChaCha`



(a) Namespace interactions tuning



(b) Namespace interactions and learning rate tuning

Figure 4. Results on openml datasets for two tuning tasks with error-bars showing the standard derivation over the 5 runs.

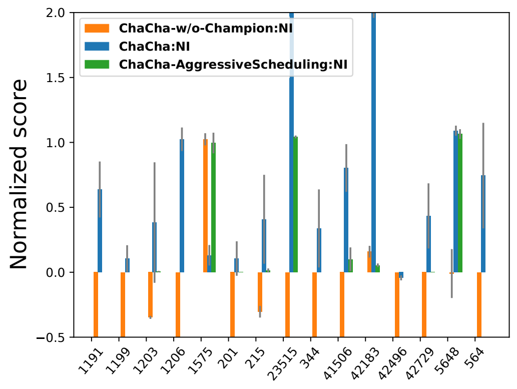


Figure 5. Ablations of ChaCha in namespace interactions tuning.

(for the tuning of namespace interactions) without these two designs. ChaCha-AggressiveScheduling denotes the variant of ChaCha in which we do the scheduling purely based on the doubling resource lease without regard for their empirical performance. ChaCha-w/o-Champion is a variant of ChaCha where we further do not intentionally keep the champion running. The bad results of ChaCha-w/o-Champion shows the necessity of keeping a learner with sufficiently large training samples 'live'. The comparison between ChaCha-AggressiveScheduling and ChaCha shows the benefit of balancing the long-term and short-term utility of configuration exploration.

We include more evaluation details, the setting of ChaCha and additional results in the appendix.

6. Conclusion

In this work, we propose a novel solution ChaCha for Online AutoML. ChaCha is a first of its kind autoML solution that can operate in an online learning manner. It respects the sharp computational constraints and optimizes for good online learning performance, i.e., cumulative regret. The sharp computational constraint and the need for learning solutions that have a good guarantee on the cumulative regret are unique properties of online learning and not addressed by any existing autoML solution. ChaCha is theoretically sound and has robust good performance in practice. ChaCha provides a flexible and extensible framework for online autoML, which makes many promising future work about online autoML possible. For example, it is worth studying how to make ChaCha handle online learning scenarios with bandit feedback.

Software and Data

Our method is open-sourced in the AutoML Libriary FLAML[2]. Please find a demonstration of usage in this notebook[3]. All the datasets are publicly available in OpenML[4].

Acknowledgements

The authors would like to thank Akshay Krishnamurthy for the discussions about this work and the anonymous reviewers for their revision suggestions.

[2]https://github.com/microsoft/FLAML/tree/main/flaml/onlineml

[3]https://github.com/microsoft/FLAML/blob/main/notebook/flaml_autovw.ipynb

[4]https://www.openml.org/search?type=data

References

Personalizer azure cognitive service, 2019.

Agarwal, A., Beygelzimer, A., Hsu, D. J., Langford, J., and Telgarsky, M. J. Scalable non-linear learning with adaptive polynomial expansions. In *Advances in Neural Information Processing Systems*, 2014.

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

Bergstra, J., Komer, B., Eliasmith, C., Yamins, D., and Cox, D. D. Hyperopt: a python library for model selection and hyperparameter optimization. *Computational Science & Discovery*, 8(1):014008, July 2015.

Blum, A., Kalai, A., and Langford, J. Beating the hold-out: Bounds for k-fold and progressive cross-validation. In *Proceedings of the Twelfth Annual Conference on Computational Learning Theory (COLT) 1999*.

Cesa-Bianchi, N. and Lugosi, G. *Prediction, learning, and games*. Cambridge University Press, 2006. ISBN 978-0-521-84108-5.

Chaudhuri, K., Freund, Y., and Hsu, D. A parameter-free hedging algorithm. *arXiv preprint arXiv:0903.2851*, 2009.

Cutkosky, A. and Boahen, K. A. Stochastic and adversarial online learning without hyperparameters. In *Advances in Neural Information Processing Systems*, 2017.

Dai, Z., Chandar, P., Fazelnia, G., Carterette, B., and Lalmas, M. Model selection for production system via automated online experiments. In *Advances in Neural Information Processing Systems*, 2020.

Domingos, P. A few useful things to know about machine learning. *Communications of the ACM*, 55(10):78–87, 2012.

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

Eriksson, D., Pearce, M., Gardner, J., Turner, R. D., and Poloczek, M. Scalable global optimization via local bayesian optimization. In *Advances in Neural Information Processing Systems*, 2019.

Falkner, S., Klein, A., and Hutter, F. BOHB: Robust and efficient hyperparameter optimization at scale. In *International Conference on Machine Learning (ICML)*, 2018.

Feurer, M., Klein, A., Eggensperger, K., Springenberg, J., Blum, M., and Hutter, F. Efficient and robust automated machine learning. In *Advances in Neural Information Processing Systems*, 2015.

Foster, D. J., Kale, S., Mohri, M., and Sridharan, K. Parameter-free online learning via model selection. In *Advances in Neural Information Processing Systems*, 2017.

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

Koch, P., Golovidov, O., Gardner, S., Wujek, B., Griffin, J., and Xu, Y. Autotune: A derivative-free optimization framework for hyperparameter tuning. In *Proceedings of the 24th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, 2018.

Li, L., Jamieson, K., DeSalvo, G., Rostamizadeh, A., and Talwalkar, A. Hyperband: A novel bandit-based approach to hyperparameter optimization. In *The International Conference on Learning Representations (ICLR)*, 2017.

Luo, H. and Schapire, R. E. Achieving all with no parameters: Adanormalhedge. In *Conference on Learning Theory*, 2015.

Luo, Y., Wang, M., Zhou, H., Yao, Q., Tu, W.-W., Chen, Y., Dai, W., and Yang, Q. Autocross: Automatic feature crossing for tabular data in real-world applications. In *Proceedings of the 25th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, 2019.

Medress, M. F., Cooper, F. S., Forgie, J. W., Green, C., Klatt, D. H., O'Malley, M. H., Neuburg, E. P., Newell, A., Reddy, D., Ritea, B., et al. Speech understanding systems: Report of a steering committee. *Artificial Intelligence*, 9 (3):307–316, 1977.

Muthukumar, V., Ray, M., Sahai, A., and Bartlett, P. Best of many worlds: Robust model selection for online supervised learning. In *The 22nd International Conference on Artificial Intelligence and Statistics*, 2019.

Orabona, F. and Pál, D. Coin betting and parameter-free online learning. *arXiv preprint arXiv:1602.04128*, 2016.

Real, E., Liang, C., So, D., and Le, Q. Automl-zero: evolving machine learning algorithms from scratch. In *International Conference on Machine Learning (ICML)*, 2020.

Sabharwal, A., Samulowitz, H., and Tesauro, G. Selecting near-optimal learners via incremental data allocation. In *Proceedings of the AAAI Conference on Artificial Intelligence*, 2016.

Sato, M.-A. Online model selection based on the variational bayes. *Neural computation*, 13(7):1649–1681, 2001.

Shalev-Shwartz, S. and Ben-David, S. *Understanding Machine Learning: From Theory to Algorithms*. Cambridge University Press, USA, 2014. ISBN 1107057132.

Snoek, J., Larochelle, H., and Adams, R. P. Practical bayesian optimization of machine learning algorithms. In *Advances in Neural Information Processing Systems*, 2012.

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.

Vapnik, V. *The nature of statistical learning theory*. Springer science & business media, 2013.

Wang, C., Wu, Q., Huang, S., and Saied, A. Economical hyperparameter optimization with blended search strategy. In *The International Conference on Learning Representations (ICLR)*, 2021a.

Wang, C., Wu, Q., Weimer, M., and Zhu, E. Flaml: A fast and lightweight automl library. In *The Conference on Machine Learning and Systems (MLSys)*, 2021b.

Wu, Q., Wang, C., and Huang, S. Frugal optimization for cost-related hyperparameters. In *Proceedings of the AAAI Conference on Artificial Intelligence*, 2021.