Ease.ml: Towards Multi-tenant Resource Sharing for Machine Learning Workloads

Tian Li[†], Jie Zhong[‡], Ji Liu[‡], Wentao Wu[*], Ce Zhang[†]
[†]ETH Zurich [‡]University of Rochester [*]Microsoft Research
litian@ethz.ch, jie.zhong@rochester.edu, jliu@cs.rochester.edu,
wentao.wu@microsoft.com, ce.zhang@inf.ethz.ch

ABSTRACT

We present `ease.ml`, a *declarative* machine learning service platform. With `ease.ml`, a user defines the high-level schema of an ML application and submits the task via a Web interface. The system then deals with the rest, such as model selection and data movement. The *ultimate* question we hope to understand is that, *as a "service provider" that manages a shared cluster of machines running machine learning workloads, what is the resource sharing strategy that maximizes the global satisfaction of all our users?*

This paper does *not* completely answer this general question, but focuses on solving the first technical challenge we were facing when trying to build `ease.ml`. We observe that resource sharing is a critical yet subtle issue in this multi-tenant scenario, as we have to balance between efficiency and fairness. We first formalize the problem that we call *multi-tenant model selection*, aiming for minimizing the total *regret* of all users running automatic model selection tasks. We then develop a novel algorithm that combines multi-armed bandits with Bayesian optimization and prove a regret bound under the multi-tenant setting. Finally, we report our evaluation of `ease.ml` on synthetic data and on two services we are providing to our users, namely, image classification with deep neural networks and binary classification with Azure ML Studio. Our experimental evaluation results show that our proposed solution can be up to 9.8× faster in achieving the same global average accuracy for all users as the two popular heuristics used by our users before `ease.ml`, and 4.1× faster than state-of-the-art systems.

PVLDB Reference Format:
Tian Li, Jie Zhong, Ji Liu, Wentao Wu, and Ce Zhang. Ease.ml: Towards Multi-tenant Resource Sharing for Machine Learning Workloads. *PVLDB*, 11 (5): 607 - 620, 2018.
DOI: https://doi.org/10.1145/3177732.3177737

1. INTRODUCTION

The past decade has witnessed the increasingly ubiquitous application of machine learning in areas beyond computer sciences. One consequence is that we can no longer assume a CS background from our users. As a result, how to make machine learning techniques more *accessible* and *usable* to non-computer science users has become a research topic that has attracted intensive interest from the database community [3, 6, 24, 39, 41].

Proceedings of the VLDB Endowment, Vol. 11, No. 5

DOI: https://doi.org/10.1145/3177732.3177737

Motivating Example. At ETH Zurich, we as a research group provide "data science services" to other research groups outside the computer science domain. These users provide us their data sets and the tasks they want to perform with machine learning. We host these data sets on our machines for our users to run their machine learning jobs. Although we also have students serving as "consultants" to answer user questions, it is still our users who run the machine learning systems by themselves in most cases. As of today, our infrastructure contains 24 TITAN X GPUs and 100 TB storage shared by more than ten research groups from areas that include astrophysics, biology, social sciences, meteorology, and material science. In this paper, we ask this question: *What is an efficient and effective way to enable multiple users sharing the same computational infrastructure to run machine learning tasks?*

Failed Experience 1. Our first strategy was to provide all users with `ssh` access to all our machines and a shared Google Calendar for resource allocation. However, even when everyone respects the resource allocation protocol (which rarely happens), our users compete for resources fiercely. This decentralized management strategy fell into chaos and failed in less than two weeks.

Failed Experience 2. We then resorted to classic resource managers and used `Slurm` for users to submit their jobs. Although this strategy isolates users from competing for resources, it poses a new problem for effective resource utilization. In almost all our applications, the user needs to conduct a series of explorations of different machine learning models. However, not all the explorations conducted by our users are necessary (either because the users lack machine learning knowledge or the explorations are automatic scripts conducting exhaustive searches). For example, one of our users used five GPUs for a whole week trying different models to improve a model that already had an accuracy of 0.99. Another user continued to use his resources, trying deeper and deeper neural networks even though much simpler networks already overfit on his data set. If these resources were allocated to other users, it would result in much better use of the computation time.

Motivated by these experiences, we designed `ease.ml`, a declarative machine learning service platform we built for our local collaborators at ETH Zurich that employs an automatic resource manager for multi-tenant machine learning workloads. Compared to existing multi-tenant resource managers [10, 20, 23], `ease.ml` is aware of the underlying workload and is able to integrate knowledge about machine learning to guide the exploration process more effectively. Compared to existing systems built for the single-tenant case such as Auto-WEKA [22, 40], Google Vizier [14], Spark TuPAQ [35], and Spearmint [33], `ease.ml` allows multiple users to share the same infrastructure and then tries to balance and optimize their use of it for the *average model accuracy* across all users.

Scope. The problem of multi-tenant resource management for machine learning workload is quite general. In this paper, we focus on a specific yet important respect regarding automatic model selection. We target application scenarios similar to the infrastructure we manage and the user behavior we have observed so far: Given a set of users (up to 20) with relatively *homogeneous* applications (e.g., image classification using different neural networks), and a shared computation cluster with 300TFLOPS capacity and 100TB storage, how to automatically choose the machine learning model to use for each user in order to maximize the *average model accuracy* across *all users*? Although using average accuracy as the optimization objective has several limitations (see Section 5.5), it is reasonable and useful in our scenarios. Moreover, we find that even supporting this simple metric is challenging.

Challenges. Automatic model selection is crucial to declarative machine learning services and has been studied intensively and extensively in the literature. To enable automatic model selection for multiple users competing for a shared pool of resources, we started from two previous approaches: (1) single-tenant model selection using *multi-armed bandits* [13, 14, 22, 33, 35]; and (2) *multi-task* Bayesian optimization and Gaussian Process [4, 19, 38]. When we tried to extend these methods to the particular multi-tenant scenario `ease.ml` was designed for, we faced two challenges.

(Optimization Objective) The optimization objective of single-tenant model selection is clear: Find the best model for the user as soon as possible (i.e., minimize the user's *regret* accumulated over time). However, the optimization objective becomes messy once we turn to the multi-tenant case. Previous work has proposed different objectives [38]. Unfortunately, none of these fit `ease.ml`'s application scenario as they do not capture the inherent dependence between multiple users in our model selection scenario. Thus, our first challenge was to design an appropriate global objective for `ease.ml` and then design an algorithm for this new objective.

(Heterogeneous Costs and Performance) Most existing single-tenant model selection algorithms are aware of the execution cost of a given model. This is important as models with vastly different costs may have similar performance on a particular data set. Therefore, any multi-tenant model selection algorithm that is useful in practice should also be aware of costs. Previous work on cost-aware model-selection resorts to various heuristics to integrate the cost [33]. Moreover, theoretical analysis of many state-of-the-art algorithms is usually done in a "cost-oblivious" setting where costs are neglected. The question of how to integrate costs into model-selection algorithms while retaining the desirable theoretical properties remains open. Our second challenge was to develop cost-aware multi-tenant algorithms with theoretical guarantees.

Summary of Technical Contributions. We developed the novel framework for multi-tenant, cost-aware model selection in `ease.ml`. We summarize our contributions as follows.

C1. (System Architecture and Problem Formulation) Our first contribution is the architecture of `ease.ml` and the formulation of its core technical problem. To use `ease.ml`, the user provides the system with the *schema* and *data*: (1) the shape of the input, (2) the shape of the output, and (3) pairs of example inputs and outputs. For example, if the user wants to train a classifier for images into three classes, she would submit the following job to specify that the input is a $256 \times 256 \times 3$ array and the output has 3 values:

```
Input = [256, 256, 3] Output = [3].
```

`ease.ml` then automatically matches all *consistent* machine learning models (e.g., AlexNet, ResNet-18, GoogLeNet, etc.) that can be used for this job and explores these models. Whenever a new

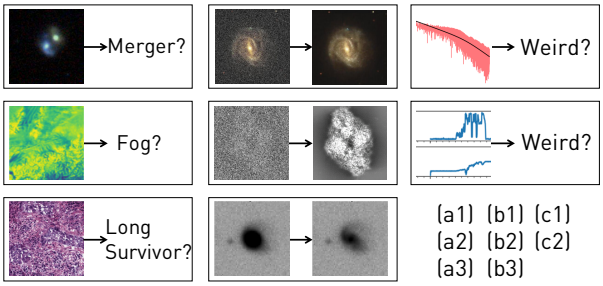

Figure 1: A gallery of `ease.ml` applications. (a) Classification tasks: (a1) Merger detection of multi-galaxy systems; (a2) Fog prediction for Zurich airport; (a3) Survival prediction for lung cancer patients. (b) Image translation: (b1) Deconvolution for astrophysics images; (b2) Denoising for cryo-electron microscopy; (b3) Quasar point sources separation. (c) Time series classification: (c1) Light curve classification; (c2) Data center anomaly detection. Applications that cannot be modeled by the current version of `ease.ml` usually involve reinforcement learning or sophisticated domain knowledge/constraints.

model finishes running, `ease.ml` returns the model to the user. With this simple abstraction, `ease.ml` supports a range of machine learning tasks such as time series classification or image translation (see Figure 6). In practice, we observe that more than 70% of our users' applications can be built in this way (see Figure1).

The core of the `ease.ml` architecture is an automatic scheduler that prioritizes the execution of different models for different users. We further formalize the scheduling problem as what we call *multi-tenant model selection*. Although similar problems have been explored in previous work [38], to the best of our knowledge we are the first to formulate the multi-tenant model selection problem in the context of a multi-tenant machine learning platform. As a consequence, we have come up with a new optimization objective based on real user requirements in practice.

C2. (Multi-tenant Model Selection Algorithms) Our second contribution is a novel algorithm for multi-tenant, cost-aware model selection. Our proposed algorithm adopts a two-phase approach. At each time step, it first determines the best model to run next for each user by estimating the "potential for accuracy improvement" for each model. It then picks the user with the highest potential in terms of the estimated accuracy improvement. For the first "model-picking" phase, we developed a cost-aware variant of the standard GP-UCB algorithm [37] for selecting the best model of each user. For the second "user-picking" phase, we developed a simple but novel criterion to decide on the best user to be scheduled next.

We studied the theoretical properties of the proposed algorithm as well as a variant that replaces the criterion in the "user-picking" phase with a round-robin strategy. In summary, we proved rigorous regret bounds for both algorithms: For the multi-tenant model selection problem with n users (each user has K candidate models to choose from), the total regret R_T of all users is bounded by

$$R_T < Cn^{3/2}\sqrt{T\log(KT^2)\log(T/n)},$$

where T is the total execution time and C is a constant. This is in line with the best-known bound for the standard GP-UCB algorithm. It implies that both algorithms are *regret-free*, i.e., $R_T/T \to 0$, which is a desired property for any practical algorithm.

We further analyzed the strength and weakness of both algorithms and designed a hybrid algorithm. The intuition is that, as the model selection procedure proceeds, the tolerance of estimation error (to distinguish between users) decreases. When the estimation error of Gaussian process exceeds the error tolerance, round robin

could potentially be better. The hybrid algorithm observes a similar regret bound but outperforms both of the original algorithms.

C3. (Evaluation) Our third contribution is an extensive evaluation of `ease.ml` on synthetic data and on real services that we are currently providing to our users. We collect two real data sets and compare `ease.ml`'s scheduler to both state-of-the-art systems and popular heuristics used by our users (prior to the availability of `ease.ml`), as well as variants of our algorithm that leverage classic scheduling policies in the "user-picking" phase such as round robin or random scheduling. We show that `ease.ml` is able to outperform these heuristic-based schedulers by up to 9.8× and outperform state-of-the-art model selection systems by up to 4.1×.

Limitations and Future Work. `ease.ml` has been running to support applications for our local collaborators [32]. As the number of users grows, we expect that the current framework, both from a theoretical and a practical perspective, needs to be improved. We highlight these limitations and future work in Section 5.5.

Overview. The rest of this paper is organized as follows. As a preliminary, in Section 2 we introduce model selection and multi-armed bandit, as well as their connections with `ease.ml`. We present the system architecture of `ease.ml` in Section 3. We next discuss the single-tenant model selection problem and propose a cost-aware variant of the standard GP-UCB algorithm in Section 4. We then formalize the multi-tenant model selection problem and present our solution and theoretical analysis in Section 5. We report experimental evaluation results in Section 6, summarize related work in Section 7, and conclude the paper in Section 8.

2. PRELIMINARIES

As a preliminary, we first introduce the single-tenant model selection problem and discuss how it can be modeled as a multi-armed bandit problem. We then introduce GP-UCB, a classic algorithm for solving the multi-armed bandit problem.

Single-tenant Model Selection. We focus on model selection problem defined as follows. For a user with a machine learning application, let there be K possible machine learning models she can choose. At time t, the user decides to train the model a_t. After training, the user observes the accuracy of the model as $x_{a_t,t}$. Therefore, the best model that the user has at time t has the quality $x_t = \max_t x_{a_t,t}$. The goal of model selection is to find a sequence of models to train in order to reach a higher x_t as fast as possible.

In `ease.ml`, we use the validation accuracy of a given model to avoid overfitting. In principle, one could use other accuracy estimators such as cross-validation accuracy or even the training accuracy in some cases. These estimators can simply be plugged into `ease.ml` and thus the problem of using which accuracy estimator is orthogonal to our proposed framework.

Multi-armed Bandit. Model selection can be treated as a multi-armed bandit problem [28]. Each model corresponds to an *arm* of the bandit, whereas the observed evaluation result of the model corresponds to the *reward* of playing the conceivable arm. A common optimization criterion is to minimize the cumulative *regret*.

Formally, let there be K arms and $[K]$ be the set of all arms. Let $x_{a_t,t}$ be the reward of playing the arm $a_t \in [K]$ at time t, and suppose that $x_{a_t,t}$ follows a distribution with the mean μ_{a_t}. Let $\mu_* = \max_{k\in[K]} \mathbb{E}(\mu_k)$ be the "best" solution in the expectation sense that is unknown to the algorithm. The *instantaneous* regret at time t if we play the arm a_t is $r_t = \mu_* - \mathbb{E}(x_{a_t,t})$ or equivalently $r_t = \mu_* - \mu_{a_t}$. The *cumulative* regret up to time T is defined as

$$R_T = \sum_{t=1}^{T} r_t = \sum_{t=1}^{T} \Big(\mu_* - \mathbb{E}(x_{a_t,t})\Big).$$

Algorithm 1 Single-tenant, cost-oblivious GP-UCB

Input: GP prior μ_0, σ, Σ, and $\delta \in (0,1)$
Output: Return the best algorithm among all algorithms in $a_{[1:T]}$

1: Initialize $\sigma_0 = \text{diag}(\Sigma)$
2: **for** $t = 1, 2, \cdots, T$ **do**
3: $\quad\beta_t \leftarrow \log(Kt^2/\delta)$
4: $\quad a_t \leftarrow \arg\max_{k\in[K]} \mu_{t-1}(k) + \sqrt{\beta_t}\sigma_{t-1}(k)$
5: $\quad$Observe y_t by playing bandit a_t.
6: $\quad\mu_t(k) = \Sigma_t(k)^\top (\Sigma_t + \sigma^2 I)^{-1} y_{[1:t]}$
7: $\quad\sigma_t^2(k) = \Sigma(k,k) - \Sigma_t(k)^\top (\Sigma_t + \sigma^2 I)^{-1}\Sigma_t(k)$
8: **end for**

Notation:

- $[K] = \{1, 2, \cdots, K\}$.
- $y_{[1:t]} = \{y_1, \cdots, y_t\}$.
- $\Sigma(i,j) = \Sigma_{i,j}$, for $i, j \in [K]$.
- $\Sigma_t(k) = [\Sigma(a_1,k), \cdots \Sigma(a_t,k)]^\top, k \in [K]$.
- $\Sigma_t = [\Sigma(i,j)]_{i,j\in a_{[1:t]}}$.

For different strategies of choosing different arms at each time t, they incur different cumulative regret. One desired property of the bandit problem is that asymptotically there is no cumulative regret, i.e., $\lim_{T\to\infty} R_T/T = 0$. The performance of a given strategy can be measured as the *convergence rate* of the cumulative regret.

Connections with `ease.ml`. In our model-selection setting, we can define a slightly different version of *regret* that is directly associated with the user experience in `ease.ml`:

$$R'_T = \sum_{t=1}^{T} \Big(\mu_* - \mathbb{E}(\max_t x_{a_t,t})\Big).$$

Intuitively, this means that the user experience of a single user in `ease.ml` relies not on the quality of the model the system gets at time t but on the best model so far up to time t because it is the "best model so far" that `ease.ml` is going to provide to its user. To capture the relation between the "`ease.ml` regret" R'_T and the classic cumulative regret R_T, observe that

$$R'_T \le R_T \quad \forall t, a_1, ..., a_t.$$

Because we are only interested in the upper bound of R'_T, for the rest of this paper, we will always try to find the upper bound for the standard cumulative regret R_T instead of R'_T directly.

2.1 Cost-Oblivious GP-UCB

We describe the standard GP-UCB algorithm [8] for a single-tenant bandit problem. The key idea of GP-UCB is to combine the Gaussian Process (GP), which models the belief at time t on the rewards of the arms $\{x_{k,t+1}\}$ at time $t+1$, with the upper confidence bound (UCB) heuristic that chooses the next arm to play given the current belief. Algorithm 1 presents the algorithm.

Gaussian Process. The Guassian Process component models the rewards of all arms at time t as a draw from a Gaussian distribution $\mathcal{N}(\mu_t, \tilde{\Sigma})$, where $\mu_t \in \mathbb{R}^K$ is the mean vector and $\tilde{\Sigma} \in \mathbb{R}^{K\times K}$ is the covariance matrix. Therefore, the marginal reward distribution of each arm is also Guassian: In Algorithm 1 (lines 4, 6, and 7), the arm k has distribution $\mathcal{N}(\mu_t(k), \sigma_t^2(k))$. The belief of the joint distribution keeps changing during the execution as more observations are available. Lines 6 and 7 of Algorithm 1 update this belief given a new observation y_t (line 5). Figure 4(a) illustrates the Gaussian Process, where the blue surface is the real underlying function and the orange surface is the current mean vector after multiple observations.

Upper Confidence Bound (UCB). The UCB rule chooses the next arm to play with the following heuristic. At time $t-1$,

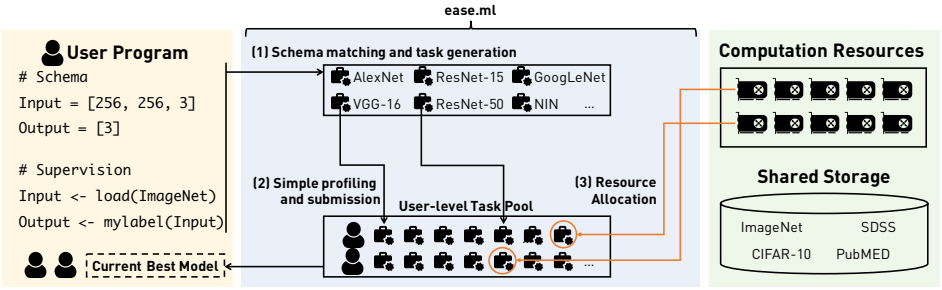

Figure 2: System architecture of `ease.ml`.

```
prog        ::=  {input: data_type, output: data_type}
data_type   ::=  {nonrec_field list, rec_field list}
nonrec_field ::= Tensor[int list] | field_name :: Tensor[int list]
rec_field   ::=  field_name
field_name  ::=  [a-z0-9_]*
```

Figure 3: Formal syntax of an `ease.ml` program

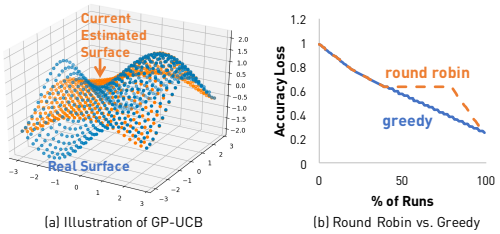

Figure 4: Illustration of (1) Gaussian process and (2) the difference between RoundRobin and Greedy.

choose the arm with the largest $\mu_k + \theta \cdot \sigma_k$ where μ_k and σ_k^2 are the current mean and variance of the reward distribution of the arm k (line 5 in Algorithm 1). This is the upper bound of the θ-confidence interval. Despite its simplicity, it exhibits a trade-off between *exploration* and *exploitation*. Intuitively, UCB favors arms with high reward (for exploitation) *and* high uncertainty (for exploration), which implies high risk but also high opportunity for gaining better reward. The choice of θ has an impact on the convergence rate.

Theoretical Analysis. The well-known cumulative regret of the UCB algorithm [8] is $R_T \leq C \cdot K \log T$, where C is some constant depending on the arm distribution. It is known to be one of the best upper bounds for the multi-bandit problem. This bound has a minor dependence on T but depends seriously on the distribution of all arms and the number of arms. This is mainly because the UCB algorithm does not consider dependence between arms. In order to make R_T/T converge to zero, we have to try at least K times. Therefore, the UCB algorithm must play all arms once or twice in the initial step. To utilize the dependence between arms, previous work has extended the UCB algorithm with the Gaussian Process, where the dependence between arms can be measured by a *kernel* matrix. The GP-UCB algorithm can achieve the regret

$$R_T \leq C \cdot \sqrt{T \log(KT^2) \log T},$$

where C is some constant that does not depend on the distribution of arms. We can see that this bound only has minor dependence on the number of arms but has a greater dependence on T. GP-UCB does not have to pull all arms once to initialize the algorithm. It can achieve a satisfactory average regret before all arms get pulled. GP-UCB is suitable when the variance of each arm is small.

The time complexity of Algorithm 1 is proportional to the number of iterations one needs to run, i.e., T. The above regret bound suggests that the average regret R_T grows *sub-linearly* with respect to T, which indicates the effectiveness of Algorithm 1.

3. SYSTEM ARCHITECTURE

The design goal of `ease.ml` is twofold: (1) provide an abstraction to enable more effective model exploration for our users, and (2) manage the shared infrastructure to enable more efficient resource utilization during the exploration process for *all* instead of *one* of our users. A similar, but vague, objective for `ease.ml` was published as a short vision paper [42]. This paper tackles the first concrete technical problem we faced in realizing this vision.

`ease.ml` provides a simple interface to the user: In `ease.ml`, the user thinks about machine learning as a *function approximator*. To specify such an approximator, the user provides the system with (1) the shape of the input, (2) the shape of the output, and (3) pairs of examples that the function aims to approximate. Figure 2 illustrates the system architecture of `ease.ml`.

3.1 Components and Implementations

We walk through each component of `ease.ml` and describe design decisions motivated by our observation of our users.

Input Program. The input of `ease.ml` is program written in a simple syntax defined in Figure 3. To specify a *machine learning task*, the user program (***prog***) contains the information about the *shape* and *structure* of an input object (e.g., images, time series) and an output object (e.g., class vector, images, time series).

The design goal of input and output objects is to provide enough flexibility to support most workloads that we observed from our users. In the current design, each object (***data_type***) contains two parts: (1) the "recursive" component (***rec_field***) and (2) the "nonrecursive" component (***nonrec_field***). The recursive component contains a list of named fields of the type of the same object, and the nonrecursive component contains a list of constant-sized tensors. This combination of recursive and nonrecursive components allows `ease.ml` to model a range of the workloads that our users need, including image, time series, and trees.

(Example) Figure 5 shows two example user programs for (1) image classification and (2) time series prediction. For image classification, each input object is a tensor of the size 256×256×3 and each output object is a tensor of the size 1,000 corresponding to 1,000 classes. Here the input and output objects contain only the nonrecursive component. For the time series prediction, each object contains a 1-D tensor and a "pointer" to another object of the same type. This forms a time series.

Code Generation and User Interaction. Given an input program, `ease.ml` conducts code generation to generate binaries that the user directly interacts with. Figure 5 illustrates the process.

In the first step of code generation, `ease.ml` translates the user programs into *system-data types*, data types that the rest of the system is able to understand. Figure 5 shows the system-data types

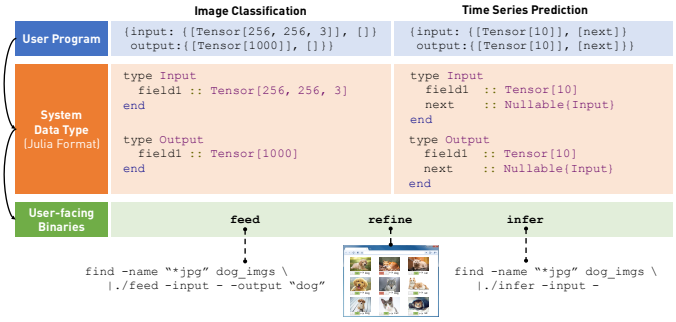

Figure 5: System walkthrough

Input Template	Type of Workload	Consistent Models
`Input : {[Tensor[A,B,C]], []}` `Output: {[Tensor[D]], []}`	Image/Tensor Classification	AlexNet, ResNet, GoogLeNet, SqueezeNet, VGG, NIN, BN-AlexNet
`Input : {[Tensor[A,B,C]], []}` `Output: {[Tensor[D,E,F]], []}`	Image/Tensor "Recovery"	Auto-encoder, GAN, pix2pix
`Input : {[Tensor[A], *], [a]}` `Output: {[Tensor[D]], []}`	Time Series Classification	RNN, LSTM, bi-LSTM, GRU
`Input : {[Tensor[A], *], [a]}` `Output: {[Tensor[B], *], [b]}`	Time Series "Translation"	seq2seq
`Input : {[Tensor[A], *], [a, c]}` `Output: {[Tensor[B]], []}`	Tree Classification	Tree-RNN, Tree kernel SVM
`Input : {[*], [*]}` `Output: {[Tensor[B]], []}`	General Classification	Bit-level RNN
`Input : {[*], [*]}` `Output: {[*], [*]}`	General Auto-encoder	Bit-level Auto-encoder

Figure 6: Templates for candidate model generations. `A`, `B`, `C`, `D`, `E`, and `F` are natural number constants; `a`, `b`, and `c` are of the type *`field_name`*. `*` represents matching for arbitrary "tail" of an array. Matching order goes from top to bottom.

generated in Julia. The translation process is based on very simple operational semantics, and we thus omit the details here. One inherent assumption during the translation is that there is no reuse of objects, i.e., we can only generate types corresponding to DAG without loop (e.g., singleton, chains, and trees).

Given the system-data types generated by the translation procedure, `ease.ml` then generates three binaries and a Python library. The Python library shares the same functionality as the binaries but can be used in a programmable way. Figure 5 shows the three binaries, and Figure 2 illustrates one example in Python. The binaries and the Python library contain a unique identifier and an IP address mapped to the `ease.ml` server. All operations the users conduct with the generated binaries and Python library will be sent to the server, which hosts the shared storage and the pool of computation resources. There are three basic operations in `ease.ml`.

1. `feed`. The `feed` operator takes as input a set of input/output pairs. `ease.ml` provides a default loader for some popular Tensor types (e.g., loads JPEG images into Tensor[A,B,3]). To associate an output object and an input object, the user can simply pipe a pair of objects into the binary, as shown in Figure 5 or write a *labeling function* that maps an input object into an output object as shown in Figure 2. Every time the user invokes the `feed` operator, all data will be sent and stored in the centralized `ease.ml` server.

2. `refine`. The `refine` operator shows all input/output pairs that the user ever `feed`'ed into the system and allows the user to "turn on" and "turn off" each example. This is especially useful when the users want to conduct data cleaning on the training set to get rid of noisy labels introduced by weak or distant supervision.

3. `infer`. The `infer` operator takes as input an input object and outputs an output object using the *best model learned so far*.

Automatic Model Exploration. The above interface provides the user with a high-level abstraction in which the user only has a "view" of the best available model instead of *what* model the system trains and *when* and *where* a model is trained. To enable this interface, automatic model selection plays a central role.

(Candidate Model Generation: Template Matching) The first step of automatic model exploration is to generate a set of candidate models given a user program. The current version of `ease.ml` uses a template-matching approach. Figure 6 shows the current set of templates and the corresponding candidate models. The matching happens from the top to the bottom (from the more specific template to the more general template).

(Candidate Model Generation: Template Augmentation) In addition to template matching, candidate models can also come from `ease.ml`'s "template augmentation" process. The intuition is that the template itself might not provide all information needed to fully specify an application. For example, in machine translation, the same template can use different word embeddings and vocabularies; in image processing for scientific applications, there are multiple normalization strategies that can be applied to the input/output images. In `ease.ml`, a single template, as is illustrated in Figure 6, will be augmented into multiple candidate models with different choices in the above example applications. `ease.ml` is aware of a collection of popular choices: For word embeddings, `ease.ml` tries (1) GloVe embeddings [31] and (2) training the embedding from input corpus; for image normalization, `ease.ml` uses $f_k(x) = -x^{2k} + x^k$ with different k. These default choices do *not* cover all, potentially better, options; however, they provide a reasonable baseline for most of our users' applications.

(Automatic Model Selection) Given a set of candidate models, `ease.ml` decides on an order of execution. The current execution strategy of `ease.ml` is to use *all* its GPUs to train a single model (see Section 5.5 for a discussion about this design decision). In the near future, `ease.ml` will need to allow a more flexible resource sharing strategy to support a resource pool with hundreds of GPUs. Because there are different users using `ease.ml` at the same time, `ease.ml` also needs to decide which user to serve at the current time. This problem motivated the core technical problem of this paper, which we describe in detail in Sections 4 and 5.

Discussion: Hyperparameter Tuning. We highlight one design decision in `ease.ml` that is not optimal. `ease.ml` also conducts automatic hyperparameter tuning but treats it as part of the training procedure: For the model-selection subsystem, when it decides to train a given model, it will always train with automatic hyperparameter tuning. In the current version of `ease.ml`, the only hyperparameter to tune is the learning rate of stochastic gradient descent. Given the current pool of 24 GPUs, `ease.ml` automatically tries $\{0.1, 0.01, 0.001\}$ by using 8 GPUs for each. This strategy does not cause a problem because even for the largest dataset and models we have, training can be finished within 1.5 hours on a single GPU. However, a more optimal design would be to fuse the hyperparameter tuning subsystem with the model-selection subsystem to better utilize available resources.

4. SINGLE-TENANT MODEL SELECTION

We focus on the model-selection subsystem in `ease.ml`. We present a simple, but novel, cost-aware GP-UCB algorithm. Despite the simplicity of the algorithm, it provides a building block for the multi-tenant setting that we will discuss in the next section.

4.1 Cost-aware Single-tenant GP-UCB

The classic GP-UCB algorithm does not consider the *cost* of playing an arm. In the context of model selection, this could correspond to the execution time of a given model. As we will see, being

aware of cost becomes even more important in the multi-tenant setting as it is one of the criteria used to balance among users.

In this section, we extend the standard GP-UCB algorithm to a cost-aware version with a very simple twist. We then analyze the theoretical property of this cost-aware algorithm. We note here that, while the theoretical analysis is relatively simple and thus we do not claim a technical contribution for it, we did not see a similar algorithm or analysis in the literature.

Cost Estimation for Machine Learning Models. Usually, more information is available in the context of model selection than in general settings of multi-armed bandits. For example, because we are aware of both the model complexity and the data scale before training, we are able to estimate the training time reasonably well (assuming users want to run a fixed number of iterations). The idea behind the cost-aware extension is then to take advantage of these cost estimates. We associate each model k with c_k, the estimate of its execution time. We then normalize execution time across all users and all applications. We will now improve on the standard GP-UCB algorithm using estimated model execution time.

A Simple Twist. Based on Algorithm 1, the idea is simple. We just replace Line 4 in the following (difference in red font)

$$a_t \leftarrow \arg\max_{k\in[K]} \mu_{t-1}(k) + \sqrt{\beta_t / c_k}\sigma_{t-1}(k),$$

where c_k is arm k's cost. This introduces a trade-off between cost and confidence: Everything being equal, the slower models (larger c_k) have lower priority. However, if it has very large potential reward (larger $\sigma_{t-1}(k)$), even an expensive arm is worth a bet.

Theoretical Analysis. The following theorem bounds the cumulative regret for our simple cost-aware GP-UCB extension. We first define the cost-aware cumulative regret as $\tilde{R}_T = \sum_{t=1}^{T} c_{a_t} r_t$.

THEOREM 1. *If $c^* = \max_{k\in[K]}\{c_k\}$ and in Algorithm 1 $\beta_t = 2c^* \log\left[\frac{\pi^2 K t^2}{6\delta}\right]$, with probability at least $1-\delta$, the cumulative regret of single-tenant, cost-aware GP-UCB is bounded above by*

$$\tilde{R}_T < \sqrt{T \cdot I(T)}$$

where $I(T) = \frac{4c^*\beta_T}{\log(1+\sigma^{-2})}\sum_{t=1}^{T}\log(1+\sigma^{-2}\sigma_{t-1}^2(a_t))$. Moreover, we have the bound for the instantaneous regret:

$$\mathbb{P}\left(\min_{t\in[T]} r_t \le \sqrt{\frac{\tilde{I}(T)}{\sum_{t=1}^{T} c_{a_t}}}\right) \ge 1-\delta,$$

where $\tilde{I}(T) = I(T)/c^*$.

Note that $I(T)$ is proportional to the information gain, and the order is at most $\log T$ (see Theorem 5 in [37] for details). Therefore, this theorem is in line with the classical GP-UCB result and yields the desired *regret-free* property that $R_T/T \to 0$ as $T \to \infty$.

In the same spirit, this theorem also suggests that the regret at iteration t (the minimal regret up to iteration t) converges to 0 with respect to the running time $\sum_{t=1}^{T} c_{a_t}$ with high probability.

5. MULTI-TENANT MODEL SELECTION

We now present the multi-tenant model-selection algorithm in `ease.ml`. This section is organized as follows.

- We formulate the problem by extending the classic single-tenant regret into a multi-tenant, cost-aware form. We then discuss the difference compared to one very similar formulation [38].
- We start from a very simple strategy that we call ROUNDROBIN. ROUNDROBIN schedules each user in a round-robin way while each user uses their own single-tenant GP-UCB during their allocated time slices. We proved a regret bound for this simple strategy.
- We then present an improved algorithm that we call GREEDY. GREEDY schedules each user by maximizing their potential contribution to the global objective. This is a novel algorithm. We also prove a regret bound for this strategy.
- The theoretical bound and empirical experiments show a trade-off between ROUNDROBIN and GREEDY. Last, we present a HYBRID strategy that balances ROUNDROBIN and GREEDY. HYBRID is the default multi-tenant model-selection algorithm in `ease.ml`.

5.1 Problem Formulation

In the multi-tenant setting, `ease.ml` aims to serve n different users instead of a single user. Without loss of generality, assume that each user $i \in [n]$ has her own machine-learning task represented by a different data set and that each user i can choose K^i machine-learning models. All users share the same infrastructure, and *at a single time point only a single user can be served.* Figure 7 illustrates a canonical view of this problem.

Multi-tenant Regrets. We extend the definitions of instantaneous and cumulative regrets for the multi-tenant setting. The key difference compared to the single-tenant setting is that, at round t, there are users who are not served. In this case, how should we define the regret for these unscheduled users?

The intuition is that these unscheduled users should also incur a penalty — because these users are not being served, they do not have a chance to get a better model. Instead, they need to stick with the same model as before. Before we describe our extension for the multi-tenant regret, we introduce notation as follows. At round t,

1. I_t: The user that the system chooses to serve.
2. $a_t^{I_t}$: The arm played by the chosen user I_t at round t.
3. t^i: The last round a user i gets served, i.e., $t^i = \arg\max\{t' : i = I_{t'}, 1 \le t' \le t\}$.
4. $a_{t^i}^i$: The arm played by a user i at the last round when she was served.
5. c_k^i: the cost of a tenant i choosing a model $k \in [K^i]$.
6. $C_t := c_{a_t^{I_t}}^{I_t}$: the cost of the algorithm chosen at round t.
7. μ_*^i: the best possible quality that a user i can get.
8. $X_t^i := x_{a_{t^i}^i, t^i}$: the rewards user i gets at time t^i.

We define the cumulative, multi-tenant, cost-aware, regret as

$$R_T = \sum_{t=1}^{T} C_t \left(\sum_{i=1}^{n} r_{t^i}^i\right),$$

where $r_{t^i}^i = \mu_*^i - \mathbb{E}(X_t^i)$ is the regret of a user i for continuing using the model chosen at the last time she got served.

Ease.ml Regret. Similar to the single-tenant case, we can define a variant of the cumulative regret R_t that directly relates to `ease.ml`'s design of always returning the *best model so far*:

$$R'_T = \sum_{t=1}^{T} C_t \left(\sum_{i=1}^{n}\left(\mu_*^i - \mathbb{E}(\max_t X_t^i)\right)\right) < R_T.$$

The Problem of "First Come First Served". One of the most straightforward ideas of serving multiple users might be the "first come first served" (FCFS) strategy in which the system will serve the tenant who comes into the system first until it finds an optimal algorithm. The system then moves on to serve the next user. This strategy incurs a terrible cumulative regret of order T.

(Example) Intuitively, it is easy to see why the FCFS strategy fails. Consider two users, each of which has the best possible model quality 100. Each user has three models:

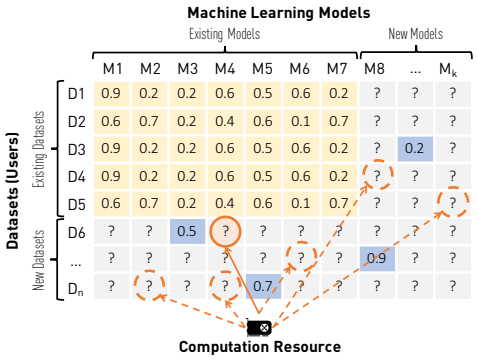

Figure 7: Multi-tenant Model Selection in `ease.ml`.

```
U1 = {M1:  90, M2:  95, M3:  100},
U2 = {M1:  70, M2:  95, M3:  100}.
```

Assume `U1` comes into the system slightly earlier and the system decides to try `M1` for `U1` in the first round. Then, in round $t = 1$, `U1` incurs regret $r^1_{t=1} = 100 - 90 = 10$ and `U2` incurs regret $r^2_{t=1} = 100$ as it does not have a model to use. In the second round, the FCFS strategy would continue to serve `U1`. Assume the system tried `M2`. Then, in round $t = 2$, `U1` incurs regret $r^1_{t=2} = 100-95 = 5$ and `U2` continues to incur regret $r^2_{t=2} = 100$ because it still has not been served. The accumulative regret among all users has already become 215 at round $t = 2$. On the other hand, if the system decides to serve `U2` in the second round, `U1` would incur regret $r^1_{t=2} = 10$, which is the same as the first round, and `U2` will incur regret $r^2_{t=2} = 100 - 70 = 30$. In this case, the accumulative regret among all users would become 150 at round $t = 2$.

This example shows the importance of choosing the next user to serve. The goal of `ease.ml` is to design an algorithm that automatically manages resource sharing among multiple users.

Discussion: Relation to Multi-task Model Selection. We compare our multi-tenant regret with the most similar multi-task regret recently studied by Swersky et al. [38]. Swersky et al. focus on k-fold cross validation. They take advantage of the correlation between folds by skipping the evaluations for certain folds for the same model. Here, each fold corresponds to one of our tenants. The multi-task regret optimized is $\sum_{t=1}^{T}\sum_{i=1}^{n} f(x,i,t)$, where $f(x,i,t)$ is the regret incurred at time t for fold i after choosing model x. The key difference is that, in their setting, choosing a better x lowers the regret for *all* users, while in our setting users who do not get served continue to incur the same regret as before. Directly using their algorithms would cause a problem in `ease.ml`. Consider a case when two users are strongly correlated. Swersky et al. would try to skip evaluating one user completely simply because it does not provide much new information. `ease.ml`, on the other hand, must balance between two users as it cannot simply reuse the model trained for user i to serve a different user j. As a result, the `ease.ml` algorithm is also very different.

5.2 Round-Robin GP-UCB

We start from a very simple strategy. Instead of serving users in an FCFS manner, what if we serve them in a "round robin" way? More precisely, with n users, at round t the system chooses to serve one user: $i = t \mod n$. When that user gets served, she runs one iteration of her own GP-UCB algorithm. Intuitively, this strategy enforces the absolute "fairness" among all users.

Theoretical Analysis. Surprisingly, the simple ROUNDROBIN strategy already has a much better regret bound compared to FCFS.

THEOREM 2. *Given $\delta \in (0,1)$, set $\beta^i_t = 2c^* \log\left[\frac{\pi^2 n K^* t^2}{6\delta}\right]$ with $c^* = \max_{i\in[n],k\in[K^i]}\{c^i_k\}$ and $K^* = \max_{i\in[n]}\{K^i\}$. Then the cumulative regret of* ROUNDROBIN *is bounded by*

$$R_T \le \sqrt{nT}\sum_{i=1}^{n}\sqrt{I_i([T(i)])}$$

with probability at least $1-\delta$, where

$$I_i([T(i)]) = \frac{8(c^*)^2\beta^*}{c_* \log(1+(\sigma^*)^{-2})} \sum_{t\in T(i)} \log\left(1+(\sigma^i)^{-2}(\sigma^i_{t-1}(a^i_t))^2\right),$$

with $c_* = \min_{i\in[n],k\in[K^i]}\{c^i_k\}$, $\beta^* = 2c^*\log\left[\frac{\pi^2 n K^* T^2}{6\delta}\right]$, $\sigma^* = \max_{i\in[n]}\{\sigma^i\}$, and $T(i)$ is the set of time points when tenant i is picked up to time T, i.e., $T(i) = \{s : 1 \le s \le T, I_s = i\}$.

In the ROUNDROBIN case, each tenant is picked for the same number of rounds, that is, all $|T(i)|$'s are roughly the same, bounded by $\lceil T/n \rceil$. Then $I_i([T(i)])$ can be uniformly upper bounded by $\beta^* \log(T/n)$ for commonly used covariance (kernel) matrices; see Theorem 5 in [37]. Consequently, the cumulative regret is upper bounded, up to a constant, by

$$n^{1.5}\sqrt{\beta^* T \log|T(i)|} = n^{1.5}\sqrt{\beta^* T \log\frac{T}{n}}. \quad (1)$$

Thus, when $T \to \infty$, we have $R'_T/T < R_T/T \to 0$.

Practical Considerations. From a theoretical point of view, ROUNDROBIN only has slightly worse theoretical bound than the GREEDY scheduler used by `ease.ml`. Practically, it outperforms FCFS significantly. However, there is still room to improve.

The enforcement of absolute fairness with a ROUNDROBIN strategy means that it may waste resources on users who already have reached their optimal solution. For example, consider two users `U1` and `U2` that both have the best possible quality 100. Assume that `U1` has already reached accuracy 99 while `U2` is still at 70. ROUNDROBIN will still schedule both users fairly. However, it is clear in this case that a better scheduler would put more resources on `U2`. This requires us to automatically balance between different users and to disproportionately serve users who the system believes could contribute more to the "global satisfaction" of all users. However, reaching this goal is not trivial. For each user, we do not know the real "best possible quality" beforehand. Thus, we need to estimate the potential for improvement for each user from their execution history and fuse these estimations in a comparable and balanced way across all users in the system. This motivates the novel GREEDY algorithm we designed for `ease.ml`.

5.3 Greedy GP-UCB

We now describe the GREEDY algorithm, an algorithm that improves over ROUNDROBIN by dynamically allocating resources.

Intuition. The intuition behind GREEDY is to have each user run their own GP-UCB. At round t, when the system tries to decide which user to serve, it estimates the "potential" of each user regarding how much each user could further benefit from another model-selection step. The system then chooses a user with high-enough potential. The technical difficulty is how to estimate the potential for each user in a way that is comparable among all users.

Algorithm 2 GREEDY, cost-oblivious, multi-tenant GP-UCB

Input: GP prior $\{\mu_0^i\}_{i=1}^n$, $\{\sigma^i\}_{i=1}^n$, $\{\Sigma^i\}_{i=1}^n$, and $\delta \in (0,1)$
Output: Return the best algorithm among all algorithms in $a^i_{[1:T]}$ for all users $[n]$.

1: **for** $i = 1, \ldots n$ **do**
2: Initialize $\sigma_0^i = \text{diag}(\Sigma^i)$ and $t_i = 1$
3: Run one step of GP-UCB (i.e., lines 3 to 5 of Algorithm 1) for the user i to obtain $\sigma^i_{t_i}$ and $\mu^i_{t_i}$
4: **end for**
5: **for** $t = 1, \ldots, T$ **do**
6: Refine regrets for users $i \in [n]$ with new observations

$$\tilde{\sigma}^i_{t_i-1} = \min\left\{B_{t_i-1}(a^i_{t_i-1}), \min_{t'<t_i-1} y^i_{t'} + \tilde{\sigma}^i_{t'}\right\} - y^i_{t_i-1}$$

where $B_t(k) = \mu_{t-1}(k) + \sqrt{\beta_t}\sigma_{t-1}(k)$.
7: Decide the candidate set

$$V_t := \left\{i \in [n] : \tilde{\sigma}^i_{t_i-1} \geq \frac{1}{n}\sum_{i=1}^n \tilde{\sigma}^i_{t_i-1}\right\}$$

8: Select a user (using any rule) from V_t indexed by j
9: Update $\beta^j_{t_j}$ by

$$\beta^j_{t_j} = \log(K^j t_j^2/\delta)$$

10: Select the best algorithm for the user j

$$a^j_{t_j} = \arg\max_{k\in[K^j]} \mu^j_{t_j-1}(k) + \sqrt{\beta^j_{t_j}}\sigma^j_{t_j-1}(k)$$

11: Observe $y^j_{t_j}(a^j_{t_j})$
12: Update $\sigma^j_{t_j}$ and $\mu^j_{t_j}$ (see lines 4 and 5 in Algorithm 1)
13: Increase the count of the user j by $t_j \leftarrow t_j + 1$
14: **end for**

The Greedy Algorithm. Algorithm 2 shows the details of the GREEDY algorithm. For simplicity, these are shown only for the cost-oblivious version of the algorithm. The cost-aware version simply replaces all occurrences of $\sqrt{\beta}$ by $\sqrt{\beta/c}$, where c is the corresponding cost. The GREEDY algorithm consists of two phases. In the first phase (lines 6 to 8), we determine which user to schedule next (i.e., the *user-picking* phase). In the second phase (lines 9 to 12), we determine which model to run for this user (i.e., the *model-picking* phase). The model-picking phase is straightforward. We choose the model with respect to the single-tenant GP-UCB criterion. (Compare lines 9 to 12 in Algorithm 2 to lines 3 to 5 in Algorithm 1.) The user-picking phase is more sophisticated.

One idea for user selection is to compare the best models from different users and pick "the best of the best." However, in what sense are the best models comparable? To understand the subtlety here, consider two models `M1` and `M2` from two users. Suppose that, at a certain time step t, the mean quality of `M1` and `M2` is 90 and 70, respectively. Moreover, assume that the Gaussian variances of `M1` and `M2` are the same. The UCB criterion will then favor `M1` over `M2`. In the single-tenant case, this makes perfect sense: `M1` is a clear win over `M2`. In the multi-tenant case, this is perhaps indefinite because it is possible that `M1` is working on an easy data set whereas `M2` is working on a hard one. Therefore, the mean quality in the UCB criterion (i.e., the μ part in the single-tenant GP-UCB algorithm) is not a reliable indicator of the potential model improvement when comparing different users. Based on this observation, we choose to omit the mean quality in the UCB criterion and focus on the observed variance.

This leads to the specific user-selection strategy illustrated in lines 6 to 8 of Algorithm 2. In line 6, we first compute more accurate regrets for users with new observations. Clearly, line 6 represents a recurrence relation on the *empirical* confidence bounds $y^i_t + \tilde{\sigma}^i_t$ that replace the means in the upper confidence bounds by the actual observations y. Assuming that user i was scheduled at time $t-1$, the empirical confidence bound for user i after time $t-1$ (i.e., at time t) is either the updated upper confidence bound B_{t-1} or the minimum empirical confidence bound before time t, whichever is smaller. Intuitively, the empirical confidence bound tries to tighten the upper confidence bound by utilizing the observed reward more directly. (The UCB criterion merely uses the observations to update the parameters of the Gaussian Process.) In lines 7 and 8, we further use the empirical variances $\tilde{\sigma}^2$ of the users to determine which user to schedule next. Specifically, we first identify a set of candidate users whose empirical confidence bounds are above the average and then pick one user from the candidates.

Strategy for Line 8. By choosing a user with a confidence bound above the average, we can reduce the time-averaging regret. It is interesting that the regret bound remains the same regardless of the rule for picking a user from the candidates (line 8), though in practice a different rule may make a difference. For example, picking the user with the maximum empirical variance may be better than randomly picking a user. In `ease.ml`, we use a rule that picks the user with the maximum gap between the largest upper confidence bound and the best accuracy so far. Nevertheless, the existence of an optimal rule for deciding on the best candidate user in the practical sense remains as an open question.

Theoretical Analysis. We can prove the following regret bound for the GREEDY algorithm.

THEOREM 3. *Given $\delta \in (0,1)$, set $\beta^i_t = 2c^* \log\left[\frac{\pi^2 n K^* t^2}{6\delta}\right]$. Then the cumulative regret of* GREEDY *is bounded by*

$$R_T \leq n\sqrt{T}\sqrt{\sum_{i=1}^n I_i([T(i)])}$$

with probability at least $1-\delta$, where

$$I_i([T(i)]) = \frac{4c^*\beta^*}{\log(1+(\sigma^*)^{-2})} \sum_{t\in T(i)} \log\left(1 + (\sigma^i)^{-2}(\sigma^i_{t-1}(a^i_t))^2\right),$$

with c^*, K^*, σ^*, β^*, and $T(i)$ defined in Theorem 2.

$I_i([T(i)])$ is proportional to the information gain for each user i. In particular, if the kernel function or covariance matrix is linear (see Theorem 5 in [37]), then for each $i \in [n]$, the order of $I_i([T(i)])$ is at most $\beta^* \log(|T(i)|)$. Since

$$\sum_{i=1}^n \log(|T(i)|) = \log\prod_{i=1}^n |T(i)|,$$

with the constraint $\sum_{i=1}^n |T(i)| = T$, we have

$$\sum_{i=1}^n \log(|T(i)|) \leq n\log\left(\frac{T}{n}\right).$$

In this case, the total regret is bounded (up to some constant) by

$$n^{3/2}\sqrt{\beta^* T \sum_{i=1}^n \log(|T(i)|)} \leq \underbrace{n^{3/2}\sqrt{\beta^* T \log\left(\frac{T}{n}\right)}}_{\text{the regret for RR, see (1)}}.$$

For two other popular kernels – the squared exponential and the Matérn kernel – we can also get a bound that is sublinear in T, and

thus $R_T/T \to 0$ as $T \to \infty$ (Section 5.2 in [37]). We see that the regret of GREEDY is slightly better than the one of ROUNDROBIN.

5.4 A Hybrid Approach

One problem of Algorithm 2 is that it may enter a *freezing stage* and never step out. That is, after a certain time step (usually at the very end of running the algorithm), the candidate set of users will remain stable. The algorithm will stick to these users forever and therefore make no further progress if the optimal models for these users have been found. The reason for this phenomenon is that the empirical variance, though improved over the Gaussian variance in the UCB criterion, is still an estimated bound rather than the true gap between the observed and optimal model quality. When the observed model quality is close to the optimal quality, this estimated bound is no longer reliable as an indicator of the true gap. Consequently, the empirical variances for the users remain almost constant, which results in a stable candidate set.

To overcome this problem, we further propose a hybrid approach. When we notice that the candidate set remains unchanged and the overall regret does not drop for s steps, we know that the algorithm has entered the freezing stage. We then switch to the round-robin user-selection strategy so that the algorithm can leave the freezing stage and make progress on the other users. We used this hybrid approach in our experimental evaluation (Section 6), where we set $s = 10$. As we have just discussed, this hybrid approach observes the same regret bound as Algorithm 2 because using a round-robin user-selection strategy instead does not change the regret bound.

5.5 Discussion on Limitations

As the first attempt at resolving the multi-tenant model-selection problem, this paper has the following limitations. From the theoretical perspective, the regret bound is not yet theoretically optimal. From the practical perspective, the current framework only supports the case where the whole GPU pool is treated as a single device. With our current scale, this is not a problem as our deep learning subsystem still achieves significant speed up in our setup. (All machines are connected with InfiniBand, all communications are in low precision [43], and we tune learning rate following Goyal et al. [16].) However, as our service grows rapidly, this nice scalability will soon disappear. We will need to extend our current framework such that it is aware of multiple devices in the near future. Another limitation is that our analysis focuses on GP-UCB and it is not clear how to integrate other algorithms such as GP-EI [34] and GP-PI [25] into a multi-tenant framework. Last, we define the global satisfaction of all users as the sum of their regrets. It is not clear how to integrate hard rules such as the "each user's deadline" and design algorithms for other aggregation functions.

In addition, in the current version of `ease.ml` we assume that the accuracy of different applications are *comparable*: 1% accuracy loss in image classification is comparable to 1% accuracy loss in machine translation. This assumption is unlikely to be true in general: When the user pool becomes more heterogeneous, we need a better way to balance between different types of applications. Potential strategies could be to use weighted sum of accuracies or to partition all GPUs into multiple "subpools" to serve different applications independently. We leave investigating these possibilities as future work when we observe more heterogeneity from our users.

Moreover, so far we have also assumed that the jobs submitted by the same user are independent. In practice, it is possible that a user may submit similar jobs. By defining a similarity measure between jobs, one could further improve `ease.ml`. We leave this as another interesting direction for future exploration, as the problem of defining dataset similarity to predict the quality of machine learning models is itself an open problem.

6. EXPERIMENTS

We present the experimental results of `ease.ml` for the real and synthetic data sets. On the real data sets, we validate that `ease.ml` is able to provide better global user experiences on one real service we are providing to our users. We then use five synthetic data sets to better understand the multi-tenant model-selection subsystem inside `ease.ml`. We validate that each of the technical contributions involved in the design of our multi-tenant model-selection subsystem results in a significant speedup over strong baselines.

6.1 Data Sets

We now describe the seven data sets we used in the experiment. Figure 8 summarizes these data sets. Each data set used in our experiment contains a set of users and machine learning models. For each (user, model) pair, there are two measurements associated with it, namely (1) quality (accuracy) and (2) cost (execution time).

(Real Quality, Real Cost) The DEEPLEARNING data set was collected from the `ease.ml` log of 22 users running image classification tasks. Each user corresponds to a data set and `ease.ml` uses eight models for each data set: NIN, GoogLeNet, ResNet-50, AlexNet, BN-AlexNet, ResNet-18, VGG-16, and SqueezeNet. Each (user, model) pair is trained with an Adam optimizer [21]. The system automatically grid-searches the initial learning rate in $\{0.1, 0.01, 0.001, 0.0001\}$ and runs each setting for 100 epochs.

The AZUREMLBENCH data set is from MLBENCH [26], our recent benchmark that compares 8 binary classifiers provided by Azure Machine Learning Studio over 17 Kaggle competitions. We recorded both quality and execution cost (including hyperparameter tuning) from Azure Machine Learning Studio. Details of the experimental setup can be found in [26].

To further understand the robustness of `ease.ml` and understand the behavior of each of our technical contributions, we used a set of synthetic data sets described as follows.

(Real Quality, Synthetic Cost) 179CLASSIFIER is a data set with real quality but synthetic costs. It contains 121 users and 179 models obtained from Delgado et al. [11] on benchmarking different machine- learning models on UCI data sets. We use each data set as a user. Because the original paper does not report the training time, we generate synthetic costs from the uniform distribution $\mathcal{U}(0, 1)$.

(Synthetic Quality, Synthetic Cost) We further generate a family of synthetic data sets with a synthetic data generator that generates synthetic quality and synthetic cost.

For N users and M models, the synthetic data generator contains a generative model for the quality $x_{i,j}$ of a model $j \in [K]$ for a user $i \in [N]$. We consider the following two factors that can affect $x_{i,j}$.
1. User baseline quality b_i: Different users have different degrees of difficulty associated with their tasks — we can achieve higher accuracy on some tasks than on others. For a user i, b_i then describes how difficult the corresponding task is. In other words, b_i characterizes the inherent difficulty of i and the final model quality $x_{i,j}$ is modeled as a fluctuation around b_i (for the model j). We simply draw the b_is from a normal distribution $\mathcal{N}(\mu_b, \sigma_b^2)$.
2. Model quality variation m_j: We use m_j to denote the fluctuation of model j over the baseline quality. The generative model for m_js needs to capture this model correlation. Specifically, for each model j, we first assign a "hidden feature" by drawing from $f(j) \sim \mathcal{U}(0, 1)$. Then, we define the covariance between two models j and j' as $\Sigma_M[j, j'] = \exp\left\{-\frac{(f(j)-f(j'))^2}{\sigma_M^2}\right\}$. We sample for each user i: $[m_1, ..., m_K] \sim \mathcal{N}(0, \Sigma_M)$.

Dataset	# Users	# Models	Quality	Cost	Total Time
DEEPLEARNING	22	8	Real	Real	496 hour
AZUREMLBENCH	17	8	Real	Real	322 hour
179CLASSIFIER	121	179	Real	Synthetic	-
SYN(0.01,0.1)	200	100	Synthetic	Synthetic	-
SYN(0.01,1.0)	200	100	Synthetic	Synthetic	-
SYN(0.5,0.1)	200	100	Synthetic	Synthetic	-
SYN(0.5,1.0)	200	100	Synthetic	Synthetic	-

Figure 8: Statistics of Datasets. "Total Time" is the total execution time of *all* models on the dataset.

We combine the above two factors and calculate the model quality as $x_{i,j} = b_i + \alpha \cdot m_j$. Each synthetic data set is specified by two hyperparameters: σ_M and α. σ_M captures the strength of the model correlation, and α captures the weight of the model correlation in the final quality. We vary these two parameters and generate the four data sets (SYN(σ_M, α)) shown in Figure 8. Note that the synthetic dataset generator does not model the scale of the datasets — the "execution time" of training each model is generated randomly. We instead use the two real datasets with real costs to understand the impact of execution time on `ease.ml`'s performance.

6.2 End-to-End Performance of ease.ml

We validate that `ease.ml` is able to achieve a better global user experience than an end-to-end system.

Competitor Strategies. We compare `ease.ml` with two strategies that most of our users were using on the DEEPLEARNING dataset before we provided them with `ease.ml`: (1) MOSTRECENT and (2) MOSTCITED. Both strategies use a round-robin scheduler to choose the next user to serve. Inside each user, it chooses the next model to train by choosing the networks with the most citations on Google Scholar or published most recently.

We also compare `ease.ml` with a version of Spearmint [34] that we adapted to the multi-tenant setting: SPEARMINT*. Each user runs her own SPEARMINT instance and SPEARMINT* schedules users in a round-robin fashion. We use the squared exponential kernel for the Gaussian process and the `EI Per Second` criterion to choose the next model for each user.

Protocol. We ran all three strategies, namely (1) `ease.ml`, (2) MOSTRECENT, and (3) MOSTCITED, on the data set DEEPLEARNING. On the other hand, we ran two strategies, namely (1) `ease.ml` and (2) SPEARMINT*, on the data set AZUREMLBENCH — we did not run MOSTRECENT and MOSTCITED because we did not have observational data on which strategy our users would use without `ease.ml`. We assume that all users enter the system at the same time and randomly sample ten users as a testing set and the rest of the users as a training set. For each strategy, we run it for 10% of the total runtime of all models. For `ease.ml`, all hyperparameters for GP-UCB are tuned by maximizing the log-marginal-likelihood as in scikit-learn.[1] We repeat the experiment 50 times.

Metrics. We measure the performance of all strategies in two ways. For each of the 50 runs, we measure the average of accuracy loss among all users at a given time point. Accuracy loss for each user is defined as the gap between the best possible accuracy among all models and the best accuracy we trained for the user so far. We then measure the average across all 50 runs as the first performance measurement. Because we treat ourselves as a "service provider", we also care about the worst-case performance and thus we measure the worst-case accuracy loss across all 50 runs as another performance measure.

[1] `http://scikit-learn.org/stable/modules/gaussian_process.html#gaussian-process`

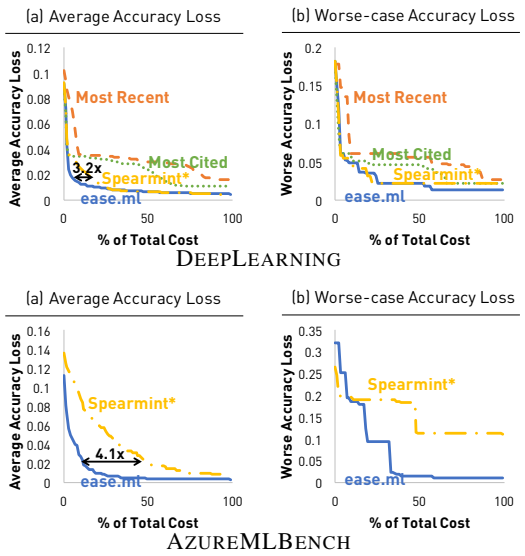

Figure 9: End-to-end performance of `ease.ml` on the two real datasets (DEEPLEARNING and AZUREMLBENCH) compared with SPEARMINT* and two strategies popularly used by our users without `ease.ml` (MOSTRECENT and MOSTCITED). Different users are scheduled with a round-robin scheduler for all three competitor strategies. Figure 11(c) presents the distributions of costs; model quality is shown in the arXiv version.

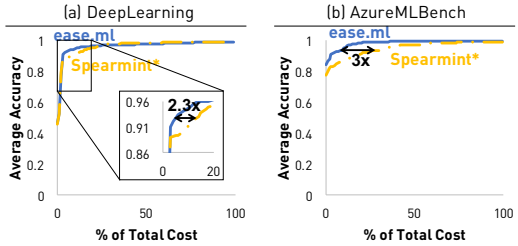

Figure 10: End-to-end performance measured with the average of *relative accuracy* normalized by the best model of each user.

Results. Figure 9 illustrates the result. On the DEEPLEARNING dataset, `ease.ml` outperforms the best of the two heuristics by up to 9.8× when comparing the average accuracy loss. The time spent on taking the average accuracy loss down from 0.1 to 0.02 of MOSTCITED is about 9.8 times of (i.e., 8.8 times longer than) that of `ease.ml`. In the worst case, `ease.ml` outperforms both competitors by up to 3.1× (details in Section 6.3).

Compared with SPEARMINT*, `ease.ml` outperforms significantly on both DEEPLEARNING and AZUREMLBENCH by up to 3.2× and 4.1× in terms of average accuracy loss. SPEARMINT* and `ease.ml` are different in two aspects: (1) For each individual user, SPEARMINT* uses cost-aware GP-EI while `ease.ml` uses cost-aware GP-UCB; and (2) `ease.ml` schedules multiple users in a different way than round robin. As we will see later in Section 6.3, even if we replace GP-EI in SPEARMINT* by GP-UCB for each user, we observe similar performance gap from `ease.ml`. The improvement of `ease.ml` over SPEARMINT* is therefore attributed to `ease.ml`'s multi-tenant scheduler.

The "accuracy loss" metric in Figure 9 is sensitive to the absolute accuracy of each user. In Figure 10 we further report the performance using the average of "relative accuracy" (i.e., the cur-

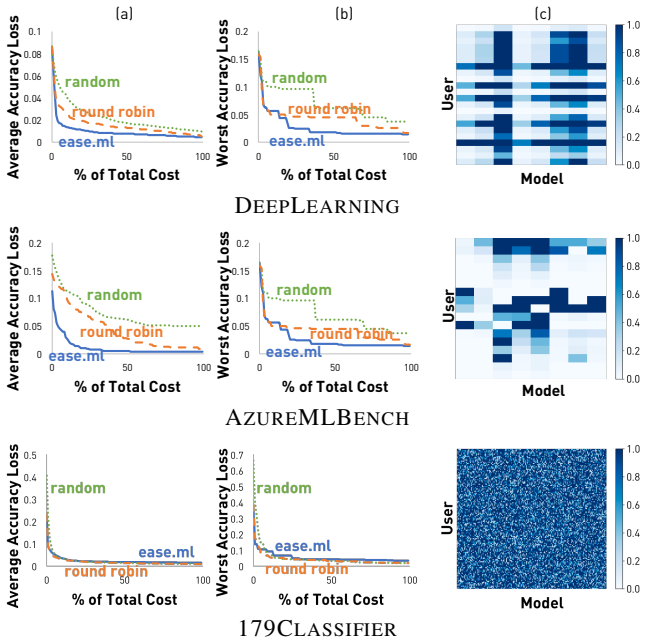

Figure 11: Performance of the cost-aware case.

rent best accuracy divided by the final best accuracy, for each user). We see that `ease.ml` still outperforms SPEARMINT* significantly on AZUREMLBENCH and DEEPLEARNING (3× and 2× faster to reach 95% of the final best accuracy) under this relative metric.

6.3 Multi-tenant Model Selection

We validate that the end-to-end performance improvement of `ease.ml` is brought about by the multi-tenant, cost-aware model-selection algorithm we propose in this paper. We follow a similar protocol as the end-to-end experiment with the following changes.

Stronger Competitors. We compare `ease.ml` with two even stronger baselines than MOSTCITED and MOSTRECENT — instead of using heuristics to choose the next models to run for each user, we use GP-UCB for each user and choose the next users to serve in a ROUNDROBIN and RANDOM way. In practice, we observe ROUNDROBIN outperforms both MOSTCITED and MOSTRECENT on DEEPLEARNING. Moreover, compared to SPEARMINT*, ROUNDROBIN uses the same multi-tenant scheduler while changing GP-EI to GP-UCB. We have observed similar performance of ROUNDROBIN and SPEARMINT* on our datasets.

Robustness to # Users. In addition to the end-to-end protocol, we also ran experiments with 50 users in the testing set for data sets with more than 100 users. The experiment result is similar to the ten-user case. Thus, we will only show the results for the ten users.

6.3.1 The Cost-oblivious, Multi-tenant Case

We first evaluate the multi-tenant setting when all systems are not cost-aware. In this case, the performance is measured as in # runs instead of cost (execution time). We run each system to allow it to train 50% of all available models.

Due to space limitation, we leave the detailed results and figures to the arXiv version of this paper and only summarize result here. On datasets with real quality we observe that the average and worst-case accuracy loss of `ease.ml` drops faster, up to 1.9×, compared to RANDOM and ROUNDROBIN. We also observe similar results for all four synthetic data sets. This shows that `ease.ml` reduces

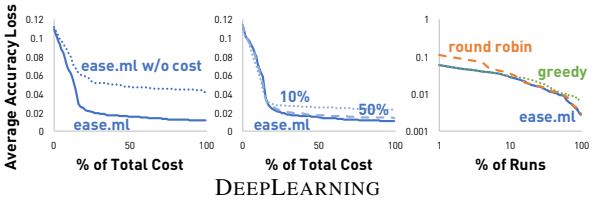

Figure 12: The impact of (left) cost-awareness; (middle) training data size (cost aware); and (right) hybrid execution.

the total regret of all users faster than the other two baselines. As we will see later, the large end-to-end improvement can only be achieved when different models have different execution costs, in which case prioritizing between users becomes more important.

Impact of Model Correlation. One important factor that has an impact on the performance of `ease.ml` is the strength of correlation among the quality of all models. Intuitively, the stronger the correlation, the better `ease.ml` would perform because its estimation on the quality of other models becomes more accurate. We study the impact of model correlation with the four synthetic data sets. As we increase σ_M from 0.01 to 0.5, the model correlation increases. As we reduce α from 1.0 to 0.1, the weight of the model correlation decreases, which implies that the impact of the model-irrelevant noise increases. We leave the figure to the arXiv version. We observed that the performance of the algorithms improves upon the stronger model correlation. This is understandable. When model correlation becomes stronger, it is easier to distinguish good models from bad ones. On the other hand, consider an extreme case when all the models are independent. In such circumstance, an evaluation of one model cannot gain information about the others, which therefore requires more explorations before the overall picture of model performance becomes clear. Meanwhile, dampening the impact of the model correlation has similar effects.

6.3.2 The Cost-aware, Multi-tenant Case

We now evaluate the multi-tenant setting when all systems are aware of the cost, the more realistic scenario that `ease.ml` is designed for. For DEEPLEARNING and AZUREMLBENCH, we use the real cost (execution time) of each model. For 179CLASSIFIER and the four synthetic data sets, we generate costs randomly.

Results. Figure 11 shows the result. The first two columns show the average accuracy loss and worst-case accuracy loss, respectively. The third column shows the cost distribution of the underlying data set. All data sets share the same quality distribution as the cost-oblivious case, as can be read in the arXiv version.

The relative performance of the three algorithms is similar to the cost-oblivious case. However, the improvement of `ease.ml` becomes more significant. This is because different costs magnify the differences between different users, and thus the multi-tenant algorithm in `ease.ml` becomes more useful.

Impact of Cost-Awareness. We validate the impact of using a cost-aware model-selection algorithm for each user. We conduct a lesion study by disabling the cost-aware component in `ease.ml` (set $c_{i,j} = 1$ for GP-UCB). Figure 12(left) shows the result on the DEEPLEARNING data set. We see that considering the execution cost of the model significantly improves the performance of the system. Combining the data distribution in the ArXiv version and the cost distribution in Figure 11, we see that this improvement is reasonable — models exist that are significantly faster on certain tasks and have a quality that is only a little bit worse than the best slower model. By integrating cost into our algorithm, `ease.ml` is able to automatically balance between execution time and the increase in quality of the global user experience.

Impact of the Size of Training Data. We validate that by providing ease.ml as a service available to multiple users and collecting the logs from all these users. ease.ml is able to use this information to help other users in the system. The design of the algorithm in ease.ml achieves this by calculating the kernel of the Guassian Process from the training set — in other words, the performance of a model on other users' data sets defines the similarity (correlation) between models. To validate the impact of this kernel, we decrease the amount of training data made available to the kernel (10%, 50%, 100%) and compare their performance. From Figure 12(middle) we see that having more models to calculate a kernel significantly improves the performance of ease.ml. On the other hand, we also observe the phenomenon of "diminishing return" — that is, using 50% of the training data results in similar performance as using 100% of the training data.

Impact of Hybrid Execution. We now validate the impact of the hybrid approach in ease.ml. We disable the hybrid component and use each of the two strategies for comparison: (1) GREEDY and (2) ROUNDROBIN. Figure 12(right) shows the result on the 179CLASSIFIER data set for the cost-oblivious case.

We observe that, while GREEDY outperforms ROUNDROBIN at the beginning, there is a crossover point as the algorithms proceed where ROUNDROBIN becomes superior to GREEDY. Switching to ROUNDROBIN after the crossover point makes the hybrid execution strategy the best among the three algorithms. The reason that the crossover point exists is because of the quality of the GP estimator: As the execution proceeds, it is more and more important for GP to have a good estimation of the quality to choose the next user to serve. When all users' current accuracy is high enough, the modeling error of treating the model-selection problem as a Gaussian Process starts to become ineligible. Thus, ROUNDROBIN works better for the second half by serving users fairly.

7. RELATED WORK

From the system perspective, ease.ml is closely related to the AutoML systems recently built by the database and machine learning communities. Examples include Spark TuPAQ [35], Auto-WEKA [22, 40], Google Vizier [14], Spearmint [33], GPyOpt [17], and Auto scikit-learn [13]. Most of these systems are built on state-of-the-art automatic model selection and hyperparameter tuning algorithms such as GP-UCB [37], GP-PI [25], GP-EI [34]. See Luo [28] for an overview. Compared to these systems, ease.ml is an AutoML system based on GP-UCB. However, the focus is on the multi-tenant case in which multiple users share the same infrastructure running machine learning workloads. To our best knowledge, this is different from all existing model-selection systems.

Multi-task Model Selection and Bayesian Optimization. The most relevant line of research is *multi-task model selection* optimized for multiple concurrent single-tenant model selection tasks. ease.ml contains a new multi-task model-selection algorithm.

Compared to previous work, ease.ml focuses on a different multi-tenant setting. For example, Swersky et al. [38] proposed different algorithms for two scenarios. Other than the cross validation application we discussed before, it also consider cases when a cheaper classifier and an expensive classifier coexist. Swersky et al. try to query the cheaper classifier to get information for the expensive one — this is different from our objective as their algorithm requires a "primary user" while in ease.ml all users need to be treated equally. Similar observations hold for previous work by Bardenet et al. [4] and Hutter et al. [19]. We designed ease.ml to optimize for an objective that we generalized from our experience in serving our collaborators and also designed a novel algorithm.

Multi-task Gaussian Process. There has been work done in extending the Gaussian Process to the multi-task case. One technical example is the *intrinsic model of coregionalization* [15] that decomposes a kernel with a Kronecker product. Most multi-task model-selection algorithms use some version of a multi-task Gaussian Process as the underlying estimator. ease.ml uses a simple multi-task Gaussian Process estimator in which we do not consider the dependencies between users. One future direction will be to further integrate user correlations into ease.ml. Another related direction is parallel Gaussian Process in which multiple processes are being evaluated instead of just one [12]. The key here is to balance the diversity of multiple samples, and integrating similar techniques to extend ease.ml's resource model from a single device to multiple devices will be the subject of future work.

Multi-tenant Clouds and Resource Management. The focus of multi-tenancy on shared infrastructure is not new — in fact, it has been one of the classic topics intensively studied by the database community. Examples include sharing buffer pools [30] and CPUs [9] for multi-tenant relational "databases-as-a-service" and sharing semi-structured data sources [5]. ease.ml is inspired by such work but focuses on multi-tenancy for machine learning.

Declarative Machine Learning Clouds. There has been intensive study of declarative machine learning systems [1, 2, 7, 18, 27, 29, 36]. In ease.ml we study how to integrate automatic model selection into a high-level abstraction and hope our result can be integrated into other systems in the future. Another related trend is the so-called "machine learning cloud." Examples include the Azure ML Studio and Amazon ML. These services provide a high-level interface and often support automatic model selection and hyperparameter tuning. ease.ml is designed with the goal of lowering the operating cost for these services by enabling multiple users sharing the same underlying infrastructure.

8. CONCLUSION

In this paper, we presented ease.ml, a declarative machine learning system that automatically manages an infrastructure shared by multiple users. We focused on studying one of the key technical problems in ease.ml, i.e., multi-tenant model-selection. We gave the first formulation and proposed a solution that extends the well-known GP-UCB algorithm into the multi-tenant, cost-aware setting with theoretical guarantees. Our experimental evaluation substantiates the effectiveness of ease.ml, which significantly outperforms popular heuristic approaches currently used in practice.

The multi-tenant model selection framework in ease.ml is motivated by our experience with supporting ease.ml users, but it applicability goes beyond ease.ml— we hope this framework can also help other service providers that manage much larger machine learning infrastructures, such as Azure Machine Learning Studio and Amazon Machine Learning, to reduce their operating costs. Moreover, the techniques used in our theoretical analysis may have their own interest and can be applied to a broader scope beyond machine learning, in particular applications that can be modeled using multi-armed bandits.

(Acknowledgement) CZ and the DS3Lab gratefully acknowledge the support from the Swiss National Science Foundation NRP 75 407540_167266, IBM Zurich, Mercedes-Benz Research & Development North America, Oracle Labs, Swisscom, Zurich Insurance, Chinese Scholarship Council, and the Department of Computer Science at ETH Zurich, the GPU donation from NVIDIA Corporation, the cloud computation resources from Microsoft Azure for Research award program.

9. REFERENCES

[1] M. Abadi, A. Agarwal, P. Barham, E. Brevdo, Z. Chen, C. Citro, G. S. Corrado, A. Davis, J. Dean, M. Devin, S. Ghemawat, I. Goodfellow, A. Harp, G. Irving, M. Isard, Y. Jia, R. Jozefowicz, L. Kaiser, M. Kudlur, J. Levenberg, D. Mane, R. Monga, S. Moore, D. Murray, C. Olah, M. Schuster, J. Shlens, B. Steiner, I. Sutskever, K. Talwar, P. Tucker, V. Vanhoucke, V. Vasudevan, F. Viegas, O. Vinyals, P. Warden, M. Wattenberg, M. Wicke, Y. Yu, and X. Zheng. TensorFlow: Large-Scale Machine Learning on Heterogeneous Distributed Systems. *ArXiv*, mar 2016.

[2] A. Alexandrov, A. Kunft, A. Katsifodimos, F. Schüler, L. Thamsen, O. Kao, T. Herb, and V. Markl. Implicit Parallelism through Deep Language Embedding. In *Proceedings of the 2015 ACM SIGMOD International Conference on Management of Data - SIGMOD '15*, pages 47–61, New York, New York, USA, 2015. ACM Press.

[3] P. Bailis, K. Olukotun, C. Re, and M. Zaharia. Infrastructure for Usable Machine Learning: The Stanford DAWN Project. *arXiv*, may 2017.

[4] R. Bardenet, M. Brendel, B. Kégl, and M. Sebag. Collaborative hyperparameter tuning. In *ICML*, pages II–199. JMLR.org, 2013.

[5] K. Bellare, C. Curino, A. Machanavajihala, P. Mika, M. Rahurkar, and A. Sane. WOO: a scalable and multi-tenant platform for continuous knowledge base synthesis. *PVLDB*, 6(11):1114–1125, 2013.

[6] C. Binnig, A. Fekete, A. Nandi, Association for Computing Machinery, C. ACM-Sigmod International Conference on Management of Data (2016 : San Francisco, and C. ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems (2016 : San Francisco. *Proceedings of the Workshop on Human-In-the-Loop Data Analytics*. ACM, 2016.

[7] M. Boehm, A. C. Surve, S. Tatikonda, M. W. Dusenberry, D. Eriksson, A. V. Evfimievski, F. M. Manshadi, N. Pansare, B. Reinwald, F. R. Reiss, and P. Sen. SystemML: Declarative Machine Learning on Spark. *PVLDB*, 9(13):1425–1436, 2016.

[8] S. Bubeck and N. Cesa-Bianchi. Regret Analysis of Stochastic and Nonstochastic Multi-armed Bandit Problems. *Foundations and Trends in Machine Learning*, 5(1):1–122, 2012.

[9] S. Das, V. R. Narasayya, F. Li, and M. Syamala. CPU sharing techniques for performance isolation in multi-tenant relational database-as-a-service. *PVLDB*, 7(1):37–48, 2013.

[10] David Shue, Michael J. Freedman, and Anees Shaikh. Performance Isolation and Fairness for Multi-Tenant Cloud Storage — USENIX. In *OSDI*, 2012.

[11] M. F. Delgado, E. Cernadas, S. Barro, and D. G. Amorim. Do we need hundreds of classifiers to solve real world classification problems? *Journal of Machine Learning Research*, 15(1):3133–3181, 2014.

[12] T. Desautels, A. Krause, and J. W. Burdick. Parallelizing Exploration-Exploitation Tradeoffs in Gaussian Process Bandit Optimization. *Journal of Machine Learning Research*, 15:4053–4103, 2014.

[13] M. Feurer, A. Klein, K. Eggensperger, J. Springenberg, M. Blum, and F. Hutter. Efficient and Robust Automated Machine Learning. In *NIPS*, pages 2962–2970, 2015.

[14] D. Golovin, B. Solnik, S. Moitra, G. Kochanski, J. E. Karro, and D. Sculley. Google Vizier: A Service for Black-Box Optimization. In *KDD*, 2017.

[15] P. Goovaerts. *Geostatistics for natural resources evaluation*. Oxford University Press, 1997.

[16] P. Goyal, P. Dollár, R. Girshick, P. Noordhuis, L. Wesolowski, A. Kyrola, A. Tulloch, Y. Jia, and K. He. Accurate, Large Minibatch SGD: Training ImageNet in 1 Hour. *ArXiv e-prints*, June 2017.

[17] GPyOpt. {GPyOpt}: A Bayesian Optimization framework in python. \url{http://github.com/SheffieldML/GPyOpt}, 2016.

[18] J. M. Hellerstein, C. Ré, F. Schoppmann, D. Z. Wang, E. Fratkin, A. Gorajek, K. S. Ng, C. Welton, X. Feng, K. Li, and A. Kumar. The MADlib analytics library: Or MAD skills, the SQL. *PVLDB*, 5(12):1700–1711, 2012.

[19] F. Hutter, H. H. Hoos, and K. Leyton-Brown. Sequential Model-Based Optimization for General Algorithm Configuration. In *LION*, pages 507–523. Springer-Verlag, 2011.

[20] Jonathan Mace, Peter Bodik, Rodrigo Fonseca, and Madanlal Musuvathi. Retro: Targeted Resource Management in Multi-tenant Distributed Systems — USENIX. In *NSDI*, 2015.

[21] D. P. Kingma and J. Ba. Adam: A Method for Stochastic Optimization. *ICLR*, dec 2014.

[22] L. Kotthoff, C. Thornton, H. H. Hoos, F. Hutter, and K. Leyton-Brown. Auto-WEKA 2.0: Automatic model selection and hyperparameter optimization in WEKA. *Journal of Machine Learning Research*, 18(25):1–5, 2017.

[23] R. Krebs, S. Spinner, N. Ahmed, and S. Kounev. Resource Usage Control in Multi-tenant Applications. In *2014 14th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing*, pages 122–131. IEEE, may 2014.

[24] S. Krishnan and E. Wu. PALM: Machine Learning Explanations For Iterative Debugging. In *Proceedings of the 2nd Workshop on Human-In-the-Loop Data Analytics - HILDA'17*, pages 1–6, New York, New York, USA, 2017. ACM Press.

[25] H. J. Kushner. A New Method of Locating the Maximum Point of an Arbitrary Multipeak Curve in the Presence of Noise. *Journal of Basic Engineering*, 86(1), 1964.

[26] Y. Liu, H. Zhang, L. Zeng, W. Wu, and C. Zhang. MLBench: How Good Are Machine Learning Clouds for Binary Classification Tasks on Structured Data? *ArXiv e-prints*.

[27] Y. Low, D. Bickson, J. Gonzalez, C. Guestrin, A. Kyrola, and J. M. Hellerstein. Distributed GraphLab. *PVLDB*, 5(8):716–727, 2012.

[28] G. Luo. A review of automatic selection methods for machine learning algorithms and hyper-parameter values. *NetMAHIB*, 5(1):18, 2016.

[29] X. Meng, J. Bradley, B. Yavuz, E. Sparks, S. Venkataraman, D. Liu, J. Freeman, D. Tsai, M. Amde, S. Owen, D. Xin, R. Xin, M. J. Franklin, R. Zadeh, M. Zaharia, and A. Talwalkar. *MLlib: machine learning in apache spark*, volume 17. MIT Press, 2016.

[30] V. Narasayya, I. Menache, M. Singh, F. Li, M. Syamala, and S. Chaudhuri. Sharing buffer pool memory in multi-tenant relational database-as-a-service. *PVLDB*, 8(7):726–737, 2015.

[31] J. Pennington, R. Socher, and C. D. Manning. Glove: Global vectors for word representation. In *EMNLP*, pages 1532–1543, 2014.

[32] K. Schawinski, C. Zhang, H. Zhang, L. Fowler, and G. K. Santhanam. Generative Adversarial Networks recover

features in astrophysical images of galaxies beyond the deconvolution limit. *Monthly Notices of the Royal Astronomical Society: Letters*, 120(1):slx008, jan 2017.

[33] J. Snoek, H. Larochelle, and R. P. Adams. Practical Bayesian Optimization of Machine Learning Algorithms. In *NIPS*, pages 2951–2959, 2012.

[34] J. Snoek, H. Larochelle, and R. P. Adams. Practical bayesian optimization of machine learning algorithms. pages 2960–2968, 2012.

[35] E. R. Sparks, A. Talwalkar, D. Haas, M. J. Franklin, M. I. Jordan, and T. Kraska. Automating model search for large scale machine learning. In *Proceedings of the Sixth ACM Symposium on Cloud Computing - SoCC '15*, pages 368–380, New York, New York, USA, 2015. ACM Press.

[36] E. R. Sparks, S. Venkataraman, T. Kaftan, M. J. Franklin, and B. Recht. KeystoneML: Optimizing Pipelines for Large-Scale Advanced Analytics. *ICDE*, 2017.

[37] N. Srinivas, A. Krause, S. Kakade, and M. W. Seeger. Gaussian process optimization in the bandit setting: No regret and experimental design. In *ICML*, pages 1015–1022, 2010.

[38] K. Swersky, J. Snoek, and R. P. Adams. Multi-Task Bayesian Optimization. In *NIPS*, pages 2004–2012, 2013.

[39] P. Tamagnini, J. Krause, A. Dasgupta, and E. Bertini. Interpreting Black-Box Classifiers Using Instance-Level Visual Explanations. In *Proceedings of the 2nd Workshop on Human-In-the-Loop Data Analytics - HILDA'17*, pages 1–6, New York, New York, USA, 2017. ACM Press.

[40] C. Thornton, F. Hutter, H. H. Hoos, and K. Leyton-Brown. Auto-WEKA. In *Proceedings of the 19th ACM SIGKDD international conference on Knowledge discovery and data mining - KDD '13*, page 847, New York, New York, USA, 2013. ACM Press.

[41] P. Varma, D. Iter, C. De Sa, and C. Ré. Flipper: A Systematic Approach to Debugging Training Sets. In *Proceedings of the 2nd Workshop on Human-In-the-Loop Data Analytics - HILDA'17*, pages 1–5, New York, New York, USA, 2017. ACM Press.

[42] C. Zhang, W. Wu, and T. Li. An Overreaction to the Broken Machine Learning Abstraction: The ease.ml Vision. In *Proceedings of the 2nd Workshop on Human-In-the-Loop Data Analytics - HILDA'17*, pages 1–6, New York, New York, USA, 2017. ACM Press.

[43] H. Zhang, K. Kara, J. Li, D. Alistarh, J. Liu, and C. Zhang. ZipML: An End-to-end Bitwise Framework for Dense Generalized Linear Models. *ICML*, 2017.