Ease.ml in Action: Towards Multi-tenant Declarative Learning Services

Bojan Karlaš[†], Ji Liu[§], Wentao Wu[‡], Ce Zhang[†]
[†]ETH Zurich [§]University of Rochester [‡]Microsoft Research
{bojan.karlas, ce.zhang}@inf.ethz.ch, jliu@cs.rochester.edu,
wentao.wu@microsoft.com

ABSTRACT

We demonstrate `ease.ml`, a multi-tenant machine learning service we host at ETH Zurich for various research groups. Unlike existing machine learning services, `ease.ml` presents a novel architecture that supports multi-tenant, cost-aware model selection that optimizes for minimizing total regrets of all users. Moreover, it provides a novel user interface that enables *declarative* machine learning at a higher level: Users only need to specify the input/output schemata of their learning tasks and `ease.ml` can handle the rest. In this demonstration, we present the design principles of `ease.ml`, highlight the implementation of its key components, and showcase how `ease.ml` can help ease machine learning tasks that often perplex even experienced users.

PVLDB Reference Format:
Bojan Karlaš, Ji liu, Wentao Wu, Ce Zhang. Ease.ml in Action: Towards Multi-tenant Declarative Learning Services. *PVLDB*, 11(12): 2054-2057, 2018.
DOI: https://doi.org/10.14778/3229863.3236258

1. INTRODUCTION

The advance and wide application of machine learning technologies, especially the development of deep neural networks, have brought up new challenges. It has been demonstrated numerous times that a deliberately designed and well tuned machine learning model can achieve comparable quality as human beings for *some* tasks, such as when the convolutional neural networks are applied to image classification and when the recurrent neural networks are applied to speech recognition. Yet, it remains elusive in many situations to understand, for a given machine learning task, which are the suitable models and how to tune their parameters.

There has been substantial past and ongoing effort to make machine learning more *declarative*, in the sense to offload the burden from end users, most of whom are not machine learning experts, by automatically selecting and tuning appropriate models. Prominent examples include commercial machine learning platforms hosted by major cloud service providers such as Amazon Machine Learning, Microsoft Azure Machine Learning, and Google Cloud AutoML, as well as extensive work in the academia and open-source community (e.g., Auto-WEKA [12] and Spearmint [10]). Not surprisingly, hyperparameter tuning and model selection have been a focus in these works and systems given that they are the driving force of automated machine learning.

Proceedings of the VLDB Endowment, Vol. 11, No. 12
Copyright 2018 VLDB Endowment 2150-8097/18/08.
DOI: https://doi.org/10.14778/3229863.3236258

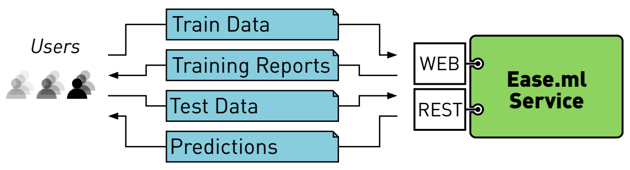


Figure 1: Interaction between users and `ease.ml`.

The vision and theoretical foundation of `ease.ml` is developed in our previous work [6, 13, 14]. In this demonstration we focus on presenting the end-to-end system to the audience. The goal of `ease.ml` is to provide declarative machine learning services to end users. The key difference of `ease.ml`, compared with existing systems, lies in that (1) it addresses the model selection problem from a multi-tenant, cost-aware perspective; and (2) it provides a more declarative user interface to let users express their learning tasks at a higher level of abstraction. Specifically, in `ease.ml` users only need to specify the input/output schemata and the system can automatically handle the rest, such as schema matching and model selection. We focused ourselves on the model selection mechanism employed in `ease.ml` and demonstrated its effectiveness in [6]. In this paper, we elaborate the design and implementation of `ease.ml` from a system building perspective, and showcase how `ease.ml` can ease machine learning tasks that are often challenging for non-expert users using concrete scenarios.

2. USER INTERFACE

As illustrated in Figure 1, a user interacts with the `ease.ml` service either through a REST API or a Web interface (which in turn acts as a facade for the REST API).

The user prepares a data set which consists of training examples given as input-output pairs. For example, both input and output can be tensors with *similar shapes*. A *schema* defines the shape and the size of each dimension of the input and output tensors. (We refer the readers to our full technical paper [6] for the details of the input/output representations and available schemata in `ease.ml` that cover cases that involve images, time series, natural language sentences, etc.)

In Figure 2, we illustrate the workflow of `ease.ml` and the interaction model between a user and the system. The major steps of this workflow are:

1. Training data upload: The user uploads her dataset. `ease.ml` will store the data set and perform automatic schema inference.

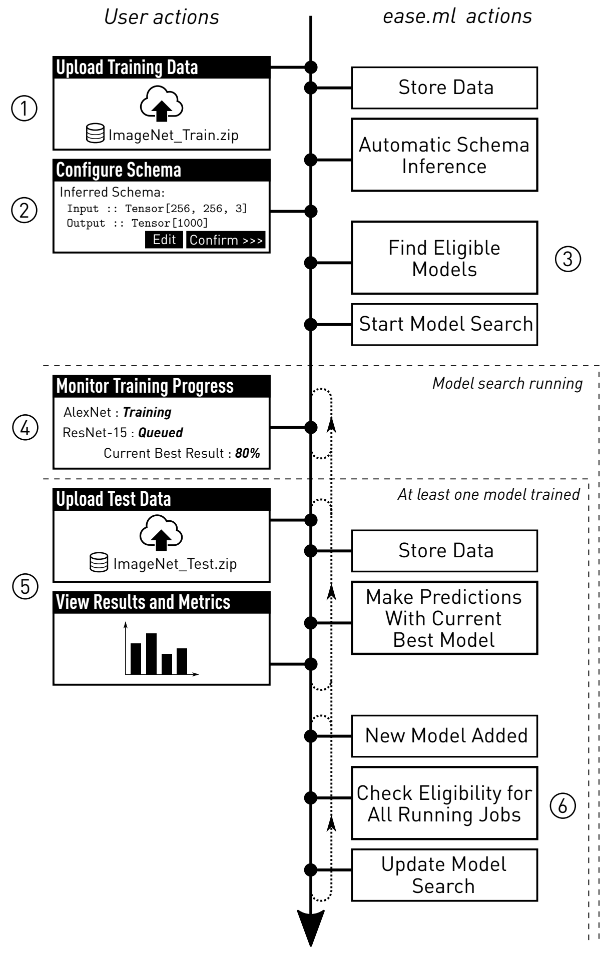


Figure 2: Workflow for model training in `ease.ml`.

2. Schema configuration: `ease.ml` returns the (automatically) inferred schema to the user for confirmation. The user can make changes if necessary. The confirmed schema is then returned to `ease.ml`. For most common machine learning tasks we saw in our applications, automatic schema inference is rather robust for common data formats. Based on the output schema of the dataset, `ease.ml` automatically produces a list of applicable objective functions allowing the user to choose one.

3. Model matching: `ease.ml` matches the data schema with its internal knowledge base, which contains all models that it is aware of. The result is a set of *candidate models* that have consistent schema; `ease.ml` then starts automatic model search within all candidate models.

4. Training progress monitoring: `ease.ml` provides a user interface that keeps getting updated automatically. From this interface, the user can monitor the training progress, the current best model, and details about all trained models through various reports that the system can produce on demand. `ease.ml` automatically deals with model selection and hyperparameter tuning using algorithms presented in our previous work [6, 13].

5. Model serving: When `ease.ml` finishes training the first model, it enables a model serving API which allows the user to get inference results given a new input tensor. The user can then develop downstream applications using this API. When `ease.ml` discovers a better model, the schema of the API stays the same (such that the user does not need to change her applications) but the prediction quality increases by switching to the better model automatically.

6. Knowledge base maintenance: When a new model is made available by other machine learning researchers (e.g., a new neural network architecture for image classification), `ease.ml` maintenance staff integrates this new model into the knowledge base. All user applications with consistent schema get pushed into the execution queue for model selection. If the new model provides a better accuracy, the model used by the corresponding model serving API gets automatically updated. In this way, users' applications always stay up to date without any intervention.

Remarks. For advanced users, our system offers more features: (1) Customizing model selection and tuning by introducing arbitrary constraints; (2) Downloading a runnable (Docker) image of a trained model; and (3) Uploading custom models and objective functions. Expert users might have background knowledge about some models that obviously cannot work. In this case, `ease.ml` also provides the possibility for ad hoc model filtering.

3. DESIGN AND IMPLEMENTATION

`ease.ml` is designed to serve multiple users at the same time with limited, shared resources. As a result, the technical challenge that `ease.ml` needs to solve is a new multi-tenant model selection scenario that was not addressed in previous work. Although there have been lots of systems on model selection (see [7] for a survey), most of them focus on the single-tenant setting. Moving to the multi-tenant setting introduces new challenges, as the system has to balance the resource demands from different users in terms of the expected potential improvements they could achieve if their models were trained. We have developed efficient techniques and we refer the readers to our full technical paper [6, 13]. However, our focus there was on the algorithmic side — we studied the theoretical properties of our proposed algorithms and compared them experimentally with baseline approaches. In this demonstration, we focus on the system design and implementation.

3.1 Design Principles

We design `ease.ml` with the following principles:

- **Multi-tenancy**: In the current design, we aim at supporting both a single machine and a moderate-size cluster with around 100 GPUs. Moreover, we hope to be able to serve both a single user and a user base up to 100 users.

- **Efficiency and fairness**: `ease.ml` should be able to prioritize among users according to the ratio of the expected model quality gain over one unit of cost.

- **Extensibility**: Adding new models and objective functions should be possible even while `ease.ml` is online. When new models are added to `ease.ml`, it should transparently improve users' model quality whenever possible.

3.2 System Architecture

We design `ease.ml` in a *model-centric* fashion: Instead of aiming at building a general Bayesian optimization framework like Spearmint [10], `ease.ml` is aware of the specific type of workload it supports, i.e., selection of machine learning models. This allows us to have a design that is tailored to machine learning. Figure 3 illustrates the lifecycle of machine learning models, and Figure 4 shows the system architecture.

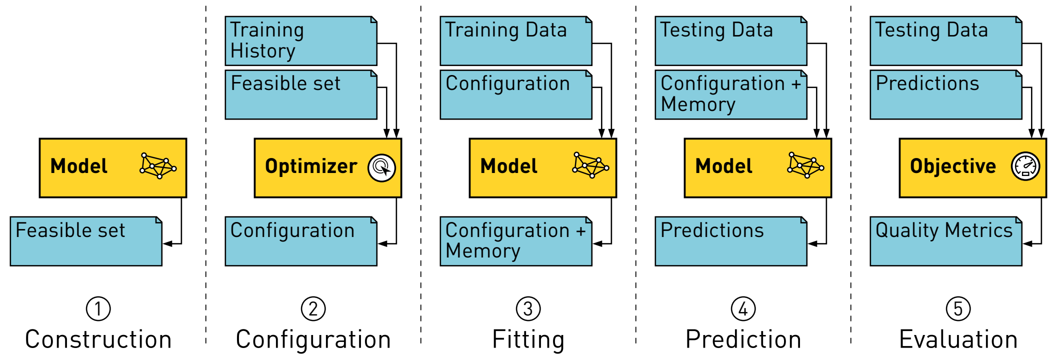


Figure 3: The lifecycle of a model that consists of five stages: construction, configuration, fitting, prediction, and evaluation.

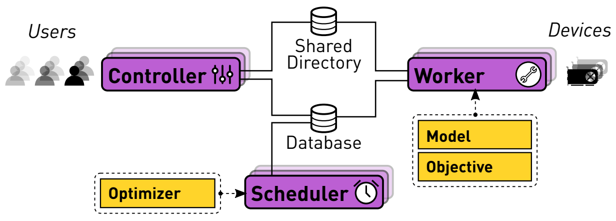


Figure 4: The system architecture in a distributed setting.

Implementation Details. We embed all *modules* (i.e. models, optimizers and objectives) inside Docker images in order to encapsulate their various dependencies. This means that all of them can be implemented in any language and using any framework. The module images are executed as stateless processes. The complete state of the system is stored in a centralized database (such as MongoDB) complemented by a shared working directory on a file system that is mounted to all instances. We split the state information as follows:

1. **Database**: Contains metadata about users, data sets, modules, jobs and tasks.
2. **Shared Directory**: It contains all data sets that users have uploaded, as well as all predictions produced by models. Apart from that, all Docker images with models, optimizers and objectives are kept here in TAR format. Finally the shared directory keeps all model parameters generated during training (i.e. model *memory*) along with logs and potential debug output.

To facilitate fault tolerance, processes do not communicate directly, but only indirectly through (persistent) message passing via databases. A process that wants to send a message will write it to an appropriate database record, and a process that is expecting a message queries the corresponding database instance.

We define three types of processes based on their workloads:

1. **Controller**: It provides an interface between users and internal data stores. It will initiate training and prediction by creating corresponding jobs and tasks respectively. Prediction tasks can immediately be picked up by workers, while training jobs need to be first processed by a **Scheduler**.
2. **Scheduler**: It picks up training jobs and runs the *Optimizer* program to suggest next training tasks to be performed by a **Worker**. The *Optimizer* program implements techniques described in our previous work [6, 13].
3. **Worker**: Each worker has a dedicated device that it uses for efficiently running the model code. It picks up tasks from the database and runs them.

3.3 Limitations and Future Work

One of the main shortcomings of our system is the model optimization method that we use, which is described in [13]. The problem is that it uses Gaussian processes for modeling the objective function of the unknown model space. The implementations of Gaussian processes rely on inverting a square $n \times n$ matrix, where n is the number of experiments performed. Since our system is always running, n is unbounded, which means that optimization could be more computationally expensive for longer experiments. How to speed up this process for long runs is an interesting future direction. We expect that updating the optimization algorithm will not change the architecture of the system.

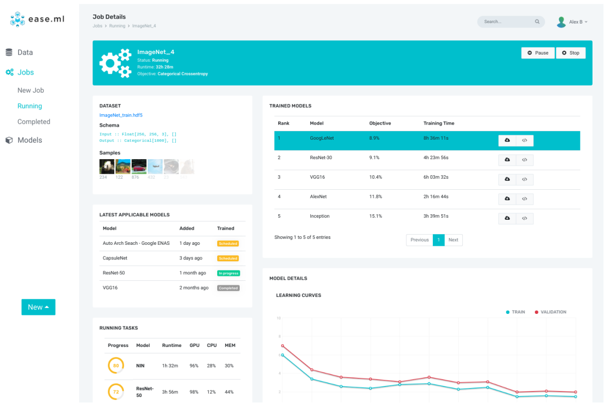
Other directions for future work include various improvements over the system itself: (1) **Early stopping** — we can stop early for long-running model training sessions predicted not promising (using techniques similar to the Halving strategy [5]); (2) **Enrichment** — we can add more functionalities to the pipeline, such as model ensemble, automated data pre- and post-processing, etc.; (3) **Efficient Model Management** — we can integrate recent research by the database community, e.g., ModelHub [8], to efficiently store and manage all training histories; (4) **Model Compression** — one requirement we see from many of our users is to automatically compress and deploy trained models to embedded systems such as embedded GPU, FPGA, and other DSPs, and we hope to integrate such functionality into `ease.ml` in the future.

4. DEMONSTRATION SCENARIOS

User Experience. We will start by walking through the various scenarios of user interactions with `ease.ml` (depicted in Figures 1 and 2) and showcasing major features of the system.

Specifically, we will upload an example data set (prepared before the demonstration) to the service. Our data parser will scrape the data set and try to automatically infer the schema, after which we will check the inferred schema and do potential modifications. The system will then display the model search space. We will explain the models and show how to exclude some models. The model search job can start after we confirm the model space.

As model training proceeds, we will show the dashboard containing all running model search jobs along with various metrics. As soon as one model is trained for a job, it becomes possible to make predictions, either through the Web interface (Figure 5) or through the REST API. We show how both approaches for predictions work and we will show the quality metrics of those predictions.



Job Details

Jobs › Running › ImageNet_4

ImageNet_4

Status: Running

Runtime: 32h 28m

Objective: Categorical Crossentropy

DATASET

ImageNet_train.hdf5

Schema

```
Input :: Float[256, 256, 3], []
Output :: Categorical[1000], []
```

Samples

TRAINED MODELS

Rank	Model	Objective	Training Time
1	GoogLeNet	8.9%	8h 36m 11s
2	ResNet-30	9.1%	4h 23m 56s
3	VGG16	10.4%	6h 03m 32s
4	AlexNet	11.8%	2h 16m 44s
5	Inception	15.1%	3h 39m 51s

Showing 1 to 5 of 5 entries

LATEST APPLICABLE MODELS

Model	Added	Trained
Auto Arch Seach - Google ENAS	1 day ago	Scheduled
CapsuleNet	3 days ago	Scheduled
ResNet-50	1 month ago	In progress
VGG16	2 months ago	Completed

RUNNING TASKS

Progress	Model	Runtime	GPU	CPU	MEM
80	NIN	1h 32m	96%	28%	30%
72	ResNet-50	3h 56m	98%	12%	44%

MODEL DETAILS

LEARNING CURVES

Figure 5: Web interface of `ease.ml`.

Finally, we will wait for the model search to converge, essentially finding the best model possible for the given data set.[1] The convergence state indicator will be shown in the dashboard next to the job. Then we will introduce a new model to the system that will be applicable to the same data set. We will then see how the system picks this up and continues the model search while notifying the user through the dashboard that the model search has continued.

Audience participation: We also plan to prepare several other data sets. After our walkthrough, we will invite the audience to try our service using these backup data sets.

Multi-user Scheduling Algorithm. In this part, we will show step by step visualizations of an example model search job. We will visualize the state of all processes along with internal dynamics of the scheduling algorithm. We will perform this exercise both for a single-user and a multi-user scenario. The goal is to show how the scheduling policy is affected by having multiple users competing for a limited amount of computational resources.

Audience participation: We would like to encourage the audience to examine the intermediate scheduling steps using our backup data sets and the visualization interface. Hopefully they will get a deeper understanding of how the multi-tenant scheduler underlying `ease.ml` works by trying out by themselves.

Applications. `ease.ml` has been used to enable more than ten scientific applications developed by our users at both ETH Zurich and other institutes. In the last demonstration scenario, we showcase the power of the simple interface provided by `ease.ml` by demonstrating an array of real-world applications, ranging from astrophysics, biology, proteomics, and meteorology, built by our users using `ease.ml`.

5. RELATED WORK

There is a growing number of online machine learning services. Some prominent examples include: Amazon Machine Learning on AWS, bogml.com, datarobot.com, Google Cloud Auto ML, Microsoft Azure Machine Learning, rapidminer.com, prediction.io, and skytree.net. They all contain various features to facilitate the work of data scientists. Most of them offer some sort of automated model search and/or tuning capabilities.

Among published work which aims at automated model selection and tuning, we have Auto-WEKA [4, 12], Auto-sklearn [1], Spark TuPAQ [11], and Google Vizier [2]. Spearmint [10] and GPyOpt [3] focus only on hyperparameter tuning. All of them focus on a single-user scenario without offering any explicitly enforced resource sharing policies.

All mentioned systems use some form of black-box optimization for their model selection and hyperparameter tuning. Current successful approaches mostly rely on Bayesian optimization (see [9] for a review). One of the focus points of our system is optimal resource allocation in a multi-user setting. Therefore, we use a variant of the Gaussian Process based approach combined with the Expected Improvement acquisition function, but modified for the multi-user, multi-device scenario [13], with proven theoretical guarantees for speed of convergence.

[1]For the purpose of our demonstration, we will deliberately use datasets with relatively small sizes so that the convergences are fast.

6. CONCLUSION

We have demonstrated `ease.ml`, a multi-tenant service that aims for efficient resource allocation and improved user experience in declarative machine learning. Compared with other systems, `ease.ml` allows users to express their learning tasks at a more abstract level by focusing on specifying the input/output schemata. We have also highlighted the key principles underlying the design and implementation of `ease.ml` that enable automated model selection and hyperparameter tuning behind the scene.

Acknowledgements. CZ and the DS3Lab gratefully acknowledge the support from Mercedes-Benz Research & Development North America, Oracle Labs, Swisscom, Zurich Insurance, Chinese Scholarship Council, and the Department of Computer Science at ETH Zurich.

7. REFERENCES

[1] M. Feurer, A. Klein, K. Eggensperger, J. Springenberg, M. Blum, and F. Hutter. Efficient and robust automated machine learning. In *Proc. NIPS*, pages 2962–2970, 2015.

[2] D. Golovin, B. Solnik, S. Moitra, G. Kochanski, J. Karro, and D. Sculley. Google vizier: A service for black-box optimization. In *KDD*, pages 1487–1495. ACM, 2017.

[3] GPyOpt. GPyOpt: A bayesian optimization framework in python. http://github.com/SheffieldML/GPyOpt, 2016.

[4] L. Kotthoff, C. Thornton, H. H. Hoos, F. Hutter, K. Leyton-Brown, et al. Auto-weka 2.0: Automatic model selection and hyperparameter optimization in weka. *JMLR*, 18:5, 2017.

[5] L. Li, K. Jamieson, G. DeSalvo, A. Rostamizadeh, and A. Talwalkar. Hyperband: A novel bandit-based approach to hyperparameter optimization. *arXiv preprint arXiv:1603.06560*, 2016.

[6] T. Li, J. Zhong, J. Liu, W. Wu, and C. Zhang. Ease.ml: Towards multi-tenant resource sharing for machine learning workloads. *PVLDB*, 11(5):607–620, 2018.

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

[8] H. Miao, A. Li, L. S. Davis, and A. Deshpande. Modelhub: Deep learning lifecycle management. In *ICDE*, pages 1393–1394, April 2017.

[9] B. Shahriari, K. Swersky, Z. Wang, R. P. Adams, and N. De Freitas. Taking the human out of the loop: A review of bayesian optimization. *Proc. IEEE*, 104(1):148–175, 2016.

[10] J. Snoek, H. Larochelle, and R. P. Adams. Practical bayesian optimization of machine learning algorithms. In *NIPS*, pages 2951–2959, 2012.

[11] 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 *Proc. SoCC*, pages 368–380. ACM, 2015.

[12] C. Thornton, F. Hutter, H. H. Hoos, and K. Leyton-Brown. Auto-weka: Combined selection and hyperparameter optimization of classification algorithms. In *KDD*, pages 847–855. ACM, 2013.

[13] C. Yu, C. Zhang, J. Zhong, B. Karlas, and J. Liu. Multi-device, multi-tenant model selection with gp-ei. 2018. under review.

[14] C. Zhang, W. Wu, and T. Li. An overreaction to the broken machine learning abstraction: The ease.ml vision. In *HILDA*, pages 3:1–3:6, 2017.