Ease.ml/ci and Ease.ml/meter in Action: Towards Data Management for Statistical Generalization[*]

Cedric Renggli[*,†] Frances Ann Hubis[*,†]
Bojan Karlaš[†] Kevin Schawinski[§] Wentao Wu[‡] Ce Zhang[†]
† ETH Zurich ‡ Microsoft Research, Redmond § Modulos AG
{cedric.renggli, hubisf, bojan.karlas, ce.zhang}@inf.ethz.ch
wentao.wu@microsoft.com kevin.schawinski@modulos.ai

ABSTRACT

Developing machine learning (ML) applications is similar to developing traditional software — it is often an iterative process in which developers navigate within a rich space of *requirements*, *design decisions*, *implementations*, *empirical quality*, and *performance*. In traditional software development, software engineering is the field of study which provides principled guidelines for this iterative process. However, as of today, the counterpart of "software engineering for ML" is largely missing — developers of ML applications are left with powerful tools (e.g., TensorFlow and PyTorch) but little guidance regarding the development lifecycle itself.

In this paper, we view the management of ML development *lifecycles* from a *data management* perspective. We demonstrate two closely related systems, `ease.ml/ci` and `ease.ml/meter`, that provide *some* "principled guidelines" for ML application development: `ci` is a continuous integration engine for ML models and `meter` is a "profiler" for controlling *overfitting* of ML models. Both systems focus on managing the "statistical generalization power" of datasets used for assessing the quality of ML applications, namely, the *validation set* and the *test set*. By demonstrating these two systems we hope to spawn further discussions within our community on building this new type of data management systems for statistical generalization.

PVLDB Reference Format:
Cedric Renggli, Frances Ann Hubis, Bojan Karlas, Kevin Schawinski, Wentao Wu, Ce Zhang. Ease.ml/ci and Ease.ml/meter in Action: Towards Data Management for Statistical Generalization. *PVLDB*, 12(12): 1962-1965, 2019.
DOI: https://doi.org/10.14778/3352063.3352110

1. INTRODUCTION

Recent years have witnessed the rapid adaption of machine learning (ML) techniques to a diverse range of real-world applications. Part of this trend is enabled by the recent research on scalable ML training (e.g., [10]) and AutoML (e.g., [9, 11]), which significantly improve the *usability* of ML systems. Training a single ML model becomes *faster* and more *automatic*, and the portfolio of ML developers extends beyond ML experts. However, new challenges of usability arise, as developing ML applications, similar to developing traditional software, is an *iterative* process: ML systems should not only support the development of a single model, but also the *iterative process as a whole*.

(Motivating Examples) This work is motivated by our experience of supporting developers, most of whom are not ML experts, in building a range of ML applications [4, 7, 14, 15, 16, 17] over the last three years. Specifically, we are motivated by the struggles that our users face even after they have access to an AutoML system [11] with a scalable and efficient training engine [18] — when training an ML model becomes easier, users start to seek systematic guidelines, which, if followed step by step, can lead to the *right* model. This shift of focus is not dissimilar to what software engineering tries to provide for traditional software development. Typical questions that our users frequently asked include:

(Q1) Is my new model really better than my old model?

(Q2) How large does my validation set need to be? Is my current validation set still representative?

(Q3) How large does my test set need to be? Is my current test set still representative?

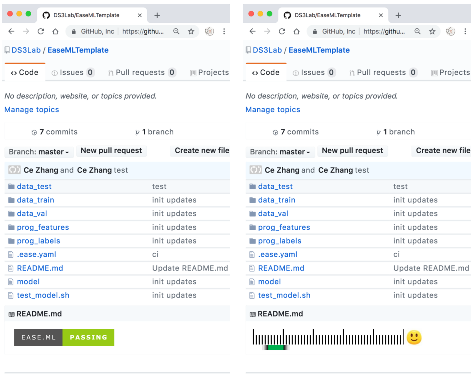
(Data Management for Statistical Generalization) These questions are essentially asking about the *statistical generalization* (ability) of a data set. Let M be a model, D_{test} be a test set, and let $\mathcal{D}_{test}$ be the (unknown) true underlying data distribution from which D_{test} is drawn. The questions posed above can be answered (see Section 2) if the system has access to a representative test score $S(M, D_{test})$ that is similar to the "true" score $S(M, \mathcal{D}_{test})$. The fundamental challenge is that the "statistical power" of D_{test} may fade once it has been used to provide an answer (i.e., D_{test} starts to *overfit*). As a result, any system we build to answer these three questions must be able to automatically *manage* the statistical power of a given dataset. This leads to a new type of data management system which, instead of managing the (relational) querying of the data, manages the statistical generalization of the data.[1]

(Summary of Demonstration Plans) In this proposal, we plan to demonstrate `ease.ml/ci` and `ease.ml/meter`, two closely related systems that we have built to manage the statistical power

[*] The first two authors contributed equally to this work.

[1] A user of this new genre of data management system operates under a resource constraint (D_{test}) managed by the systems and wants to optimally use that resource for building ML applications.

Proceedings of the VLDB Endowment, Vol. 12, No. 12
ISSN 2150-8097.
DOI: https://doi.org/10.14778/3352063.3352110



DS3Lab / EaseMLTemplate

No description, website, or topics provided.
Manage topics

7 commits · 1 branch

Branch: master · New pull request · Create new file

Ce Zhang and Ce Zhang test

data_test	test
data_train	init updates
data_val	init updates
prog_features	init updates
prog_labels	init updates
.ease.yaml	ci
README.md	Update README.md
model	init updates
test_model.sh	init updates

README.md

DS3Lab / EaseMLTemplate

No description, website, or topics provided.
Manage topics

7 commits · 1 branch

Branch: master · New pull request · Create new file

Ce Zhang and Ce Zhang test

data_test	test
data_train	init updates
data_val	init updates
prog_features	init updates
prog_labels	init updates
.ease.yaml	ci
README.md	Update README.md
model	init updates
test_model.sh	init updates

README.md

Figure 1: User interfaces of `ci` (left) and `meter` (right).

of a test set to facilitate developer's understanding of the motivating questions from above. Specifically, we will focus on the following aspects in our demonstration:

1. A demonstration of multiple real-world *use cases* that motivate the development of both systems;
2. A demonstration of the *overfitting* problem if we do not take a data management view when building these systems;
3. A demonstration of the *functionality* of both systems using real development traces of multiple ML applications;
4. Interaction with the audience to *test* the system with synthetic development traces provided interactively;
5. Discussion with the audience about future *research directions* in data management for statistical generalization.

2. SYSTEM OVERVIEW

Building `ease.ml/ci` and `ease.ml/meter` is not trivial – the technical details are mainly developed and covered in our latest full research papers [8, 12]. In this section we provide a brief overview of both systems.

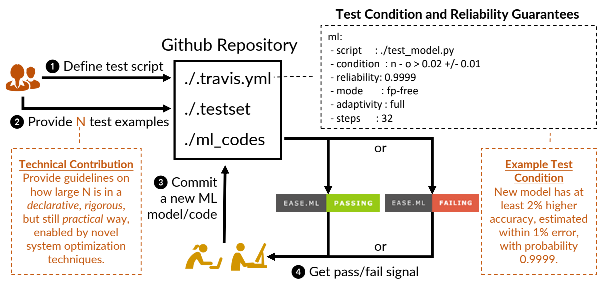
Figure 1 illustrates the interfaces of both systems. One can think about the common abstraction of both systems as "GitHub for ML" – developers commit a stream of ML models and get feedback signals for each model committed. All data sets (e.g., training, validation, and test) are managed by the systems — *developers* (who develop ML models) do not have access to the test set, while *labelers* (who provide test sets) have full access to the test set. Apart from providing signals to developers, the systems also asks labelers for new labels if required.[2] We next describe the interaction model of `ease.ml/ci` and `ease.ml/meter` in more detail.

2.1 Ease.ml/ci: Functionality

Figure 2 illustrates how users interact with `ease.ml/ci`. Similar to traditional systems for continuous integration, `ease.ml/ci` allows developers to define test conditions that specify *quality assertions* of ML models. Examples of such assertions include:

> *(A) The new version* n *of the ML model must be at least 2% more accurate than the old version* o*.*

Unlike traditional test assertions that raise a binary "pass/fail" signal, quality assertions in `ease.ml/ci` are *not* deterministic, due to the statistical nature of quantifying the model's power to generalize. Instead, we provide rigorous probabilistic guarantees for the



Test Condition and Reliability Guarantees

```
ml:
 - script     : ./test_model.py
 - condition  : n - o > 0.02 +/- 0.01
 - reliability: 0.9999
 - mode       : fp-free
 - adaptivity : full
 - steps      : 32
```

Technical Contribution
Provide guidelines on how large N is in a *declarative*, *rigorous*, but still *practical* way, enabled by novel system optimization techniques.

Example Test Condition
New model has at least 2% higher accuracy, estimated within 1% error, with probability 0.9999.

Figure 2: User interaction with `ease.ml/ci`.

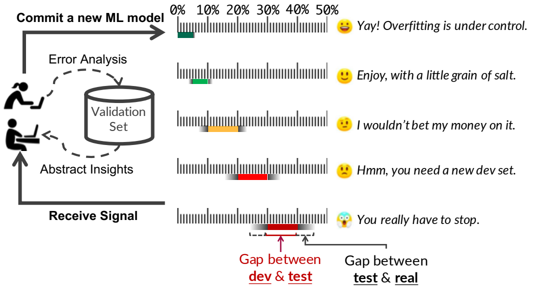


Figure 3: User interaction with `ease.ml/meter`.

assertions. Developers can specify the error tolerance of a test condition following an (ϵ, δ)-reliability paradigm: The test condition holds within the ϵ-error-bound with probability at least $1 - \delta$.

`ease.ml/ci` supports various forms of assertions based on a novel scripting language that enables user interactions in a *declarative* manner. Figure 2 also presents an example script, specifying the same assertion (A) but with error bound $\epsilon = 0.01$ and reliability (confidence) 0.9999 (i.e., $\delta = 0.0001$).

2.2 Ease.ml/meter: Functionality

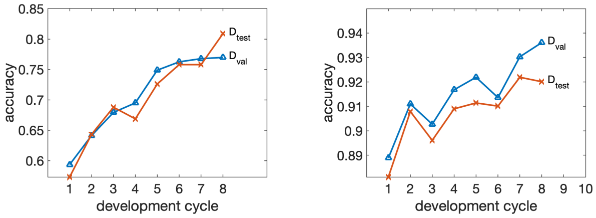
Figure 3 illustrates user interactions with `ease.ml/meter`. A human developer starts from an initial ML application H_0, and performs the following two steps in each development iteration t:

1. **Error Analysis.** The developer looks at the current validation set, and conducts error analysis on all errors currently being made by H_t;
2. **Development.** The developer then makes changes on the current ML application H_t, for example, by adding new features, adding more labeling functions, trying a different hyper-parameter setting, etc., to produce a new version of the ML application H_{t+1}.

Upon receiving the latest submission H_t, `ease.ml/meter` evaluates its accuracy over the (hidden) test set and returns an *overfitting signal* to the developer indicating the degree of overfitting of H_t.[3] Specifically, `ease.ml/meter` employs an overfitting *meter* for delivering overfitting signals to developers. The meter consists of m overfitting intervals $\{[\underline{r}_i, \bar{r}_i]\}_{i=1}^m$ that partitions the range $[0, 1]$ (e.g., $m = 5$ for the meter in Figure 3). Each range $R_i = [\underline{r}_i, \bar{r}_i]$ is further associated with an ϵ_i ($0 < \epsilon_i < 1$) that indicates the error bound of the estimated overfitting. Similar to the probabilistic guarantee (i.e., the (ϵ, δ)-reliability) provided by `ease.ml/ci` for test conditions, `ease.ml/meter` provides probabilistic guarantee for the overfitting signal it returns: The degree of overfitting is bounded by $[\underline{r}_i - \epsilon_i, \bar{r}_i + \epsilon_i]$ with probability (confidence) at least $1 - \delta$, if the meter returns signal i ($1 \leq i \leq m$).

[2] We assume that in both systems developers do not "collude" with labelers, e.g., labelers cannot send a copy of the test set to developers without having the systems know it.

[3] Following standard ML theory and practice, we define the degree of overfitting as the gap between training accuracy (computed using a validation set) and true accuracy (estimated using a test set).



(a) (SemEval 2018) (b) (SemEval 2019)

Figure 4: Real-world ML application development.

2.3 Core Technical Challenges and Solutions

The key (common) technical challenge faced by both systems, `ease.ml/ci` and `ease.ml/meter`, is to ensure probabilistic guarantees (for either test assertions in `ease.ml/ci` or overfitting signals in `ease.ml/meter`) via the (ϵ, δ)-reliability paradigm. A naive approach is to draw a new test set every time a new version of the ML application was submitted. Unfortunately, this "resampling" approach is often unfeasible in practice due to its reliance on a huge amount of *labeled* test data.

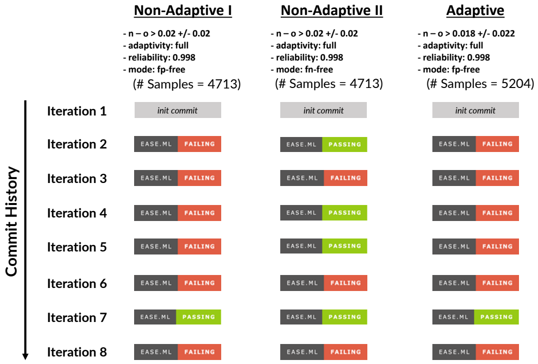
Specifically, to address the challenge in a generic setting, one has to deal with *dependencies* between subsequent submissions during the development lifecycle, due to the presence of *adaptive analysis*: The performance of the t^{th} model submission $f_t(\{X_i\})$ on the test set $\{X_i\}$ is reported back to the developer indirectly via a feedback/reporting function $g(\cdot)$. The developer can then take an *adaptive* action by picking the next submission $f_{t+1}(\{X_i\})$ based on the value returned by $g(\cdot)$. As was shown by recent work [5, 6], such kind of adaptive analysis can accelerate overfitting on the test set. As a result, to maintain the same level of statistical generalization power, one needs a larger test set (compared to the case where all submissions are independent). The technical contribution of both systems is a collection of novel statistical estimators that decreases the required test set size by two orders of magnitude — this reduction is critical for the viability of both systems. For more details, see Renggli et al. [12].

3. DEMONSTRATION SCENARIOS

We present details of the scenarios that we plan to demonstrate with `ease.ml/ci` and `ease.ml/meter`. We will use real-world development traces of two ML applications that we developed over the last two years. Each trace contains eight sequential learning models from the ML application construction. Figure 4 illustrates the evolution of validation/test accuracy of these models.

Trace 1: Relation Extraction. The first example trace comes from our submission to the "Semantic Relation Extraction and Classification in Scientific Papers" task in SemEval 2018 [13]. It is the top-ranked system for three out of four subtasks. This task aims to identify concepts from scientific documents and recognize semantic relationships that hold between the concepts. In particular, it requires semantic relation extraction and classification into six categories specific to scientific literature. The development history involves eight intermediate models before reaching the final system. Figure 4(a) plots the accuracy on the training set (using 5-fold cross validation) and the test set, respectively, in each step.

Trace 2: Emotion Detection. The second example comes from the development history of our submission to the "*EmoContext*" task in SemEval 2019. This task aims to detect emotions from text, leveraging contextual information, which is deemed challenging due to the lack of facial expressions and voice modulations.[4] It took developers eight iterations before delivering the final version. Changes in each individual step include adding word representations such as ELMo and GloVe, which lead to significant performance increase/drop. Figure 4(b) plots the accuracy on the validation set and the test set for in each step, respectively (assuming that the accuracy on the test set were reported to the user in every development step).



	Non-Adaptive I	Non-Adaptive II	Adaptive
	- n – o > 0.02 +/- 0.02 - adaptivity: full - reliability: 0.998 - mode: fp-free (# Samples = 4713)	- n – o > 0.02 +/- 0.02 - adaptivity: full - reliability: 0.998 - mode: fn-free (# Samples = 4713)	- n – o > 0.018 +/- 0.022 - adaptivity: full - reliability: 0.998 - mode: fp-free (# Samples = 5204)
Iteration 1	*init commit*	*init commit*	*init commit*
Iteration 2	EASE.ML FAILING	EASE.ML PASSING	EASE.ML FAILING
Iteration 3	EASE.ML FAILING	EASE.ML FAILING	EASE.ML FAILING
Iteration 4	EASE.ML FAILING	EASE.ML PASSING	EASE.ML FAILING
Iteration 5	EASE.ML FAILING	EASE.ML PASSING	EASE.ML FAILING
Iteration 6	EASE.ML FAILING	EASE.ML FAILING	EASE.ML FAILING
Iteration 7	EASE.ML PASSING	EASE.ML PASSING	EASE.ML PASSING
Iteration 8	EASE.ML FAILING	EASE.ML FAILING	EASE.ML FAILING

Figure 5: Continuous Integration Steps in `ease.ml/ci`.

3.1 Scenario 1: *Without*-Data Management

We will start by highlighting the importance of data management in ML application development lifecycles. We plan to showcase a couple of scenarios, where developers, intentionally or not, overfitted to the test set. In particular, we focus on demonstrating the following aspects: (1) If there is no data management system and the developers have full access to the test set, overfitting on the test set occurs easily in continuous development; (2) Even if we control its *direct* access, overfitting on the test set may still occur easily if the developers have access to the *test result*. We hope that this part would convince the audience of the motivation behind `ease.ml/ci` and `ease.ml/meter`.

3.2 Scenario 2: Ease.ml/ci

We then showcase the functionality of `ease.ml/ci` by using trace data from real-world ML application development activities. Figure 5 presents some example traces.

The development history in Figure 5 involves eight individual steps and three different scripts expressing test assertions. The first two assertions test whether the new model is better than the previous one by at least 2% (in terms of accuracy) in a *non-adaptive* setting, i.e., by assuming the models are independent. `fp-free` and `fn-free` further represent different modes that `ease.ml/ci` is operating on, enforcing elimination of *false positives* and *false negatives*, respectively. Meanwhile, the third assertion further mimics the scenario where developers can get feedback in each commit without false negatives. All these three scripting queries can be supported (in a rigorous manner) with about 5.5K test samples.

3.3 Scenario 3: Ease.ml/meter

We then showcase the functionality of `ease.ml/meter` by using the same trace data. Figure 6 illustrates the signals that the developer would get during the development process, using different modes of `ease.ml/meter`. By putting together the quality characteristics of both traces (Figure 4) and the signals from `ease.ml/meter` (Figure 6) we will guide the audience through different instances of overfitting behavior and elaborate how the

[4] `https://competitions.codalab.org/competitions/19790`

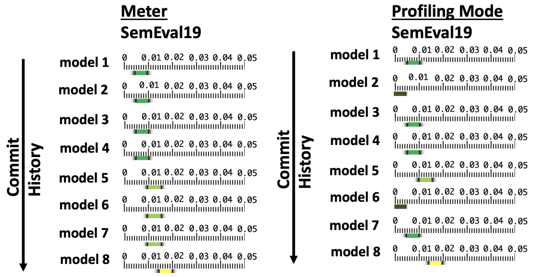
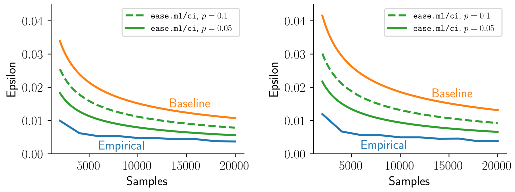


Figure 6: An illustration of using `ease.ml/meter` in Real-world ML application development (SemEval 2019).



(a) $1 - \delta = 0.99$

(b) $1 - \delta = 0.999$

Figure 7: Comparison of Sample Size Estimators.

signals from `ease.ml/meter` can support developers during the development process.

3.4 Scenario 4: Technical Demonstration

In this part, we highlight the technical differences between baseline approaches and `ci/meter` in terms of sample complexity. As an example, Figure 7 compares the estimated error with the empirical error in `ease.ml/ci`, for a model with accuracy around 98%. We refer the readers to Renggli et al. [12] for more details.

3.5 Scenario 5: Interactions with Audience

In this part, we would like to invite our audience to interact with the systems, optionally using their own synthetic development traces. We plan to provide some data generator to assist the audience to specify their trace data. We can then illustrate how `ease.ml/ci` and `ease.ml/meter` would work on these customized instances. For example, we can rerun the test conditions designed for the demonstration of `ease.ml/ci`, or showcase overfitting signals reported by `ease.ml/meter`, on these customized trace data. Also, by interactively adopting the configurations for both `ease.ml/ci` and `ease.ml/meter`, we show how this directly affects the number of required labeled samples the user needs to provide. We believe that this kind of personalized engagement provides an incentive to reflect upon current ML practices and has the potential to initiate in-depth conversations.

4. RELATED WORK

AutoML Systems. There is a great deal of recent work on developing automatic machine learning (AutoML) systems that provide "declarative" ML services to alleviate development efforts.

In a typical AutoML system, users only need to upload their datasets and provide high-level specifications of their machine learning tasks (e.g., schemata of inputs/outputs, task categories such as binary classification/multi-class classification/regression, loss functions to be minimized, etc.), and the system can take over the rest via automated pipeline execution (e.g., training/validating/testing), infrastructure support (e.g., resource allocation, job scheduling), and performance-critical functionality such as model selection and hyperparameter tuning. Examples of AutoML services include systems built by major cloud service providers, such as Amazon [1], Microsoft [3], and Google [2], as well as systems developed by academic institutions, such as the Northstar system developed at MIT [9] and our own recent effort on the `ease.ml` service [11]. In this paper, we focus on the struggles that our users have even after they have access to these AutoML systems, and hope to provide more systematic (and automatic) guidelines to help users use these powerful AutoML systems better.

Adaptive Analysis. Recent theoretic work [6] has revealed how adaptive analysis increases the risk of false conclusions reached by statistical methods (known as "false discovery" in the literature).

Intuitively, adaptive analysis makes it more likely to draw conclusions tied to the specific data set used in a statistical study, rather than conclusions that can be generalized to the underlying distribution that governs data generation. Our work in this paper can be viewed as an application of this theoretical observation to a novel, important scenario emerging from the field of ML application development lifecycle management and quality control.

5. CONCLUSION

We have demonstrated `ease.ml/ci` and `ease.ml/meter`, two systems for managing statistical generalization of test data sets in continuous ML application development activities. We view our current work as a first step towards data management in lifecycle control and quality insurance of ML applications, an emerging yet promising area that has not been paid enough attention to by the database community. We hope that our work offers some insights into this fertile ground and provides interesting scenarios that could inspire future research.

Acknowledgements CZ and the DS3Lab gratefully acknowledge the support from the Swiss National Science Foundation (Project Number 200021_184628).

6. REFERENCES

[1] Amazon Sage Maker. `aws.amazon.com/blogs/aws/sagemaker-automatic-model-tuning`.

[2] Google cloud AutoML. `cloud.google.com/automl`.

[3] Microsoft Azure Machine Learning. azure.microsoft.com/en-us/blog/announcing-automated-ml-capability-in-azure-machine-learning.

[4] S. Ackermann et al. Using transfer learning to detect galaxy mergers. *MNRAS*, 2018.

[5] A. Blum and M. Hardt. The Ladder: A reliable leaderboard for machine learning competitions. In *ICML*, 2015.

[6] C. Dwork et al. Preserving statistical validity in adaptive data analysis. In *STOC*, pages 117–126, 2015.

[7] I. Girardi et al. Patient risk assessment and warning symptom detection using deep attention-based neural networks. *LOUHI*, 2018.

[8] F. A. Hubis et al. Ease.ml/meter: Quantitative overfitting management for human-in-the-loop ml application development. *CoRR*, abs/1906.00299, 2019.

[9] T. Kraska. Northstar: An interactive data science system. *PVLDB*, 11(12):2150–2164, 2018.

[10] M. Li et al. Scaling distributed machine learning with the parameter server. In *OSDI*, pages 583–598, 2014.

[11] T. Li et al. Ease.ml: Towards multi-tenant resource sharing for machine learning workloads. *PVLDB*, 11(5):607–620, 2018.

[12] C. Renggli et al. Continuous integration of machine learning models with ease. ml/ci: Towards a rigorous yet practical treatment. *SysML*, 2019.

[13] J. Rotsztejn et al. ETH-DS3Lab at semeval-2018 task 7: Effectively combining recurrent and convolutional neural networks for relation classification and extraction. In *SemEval@NAACL-HLT*, 2018.

[14] K. Schawinski et al. Generative adversarial networks recover features in astrophysical images of galaxies beyond the deconvolution limit. *MNRAS*, 2017.

[15] K. Schawinski et al. Exploring galaxy evolution with generative models. *Astronomy & Astrophysics*, 2018.

[16] D. Stark et al. PSFGAN: a generative adversarial network system for separating quasar point sources and host galaxy light. *MNRAS*, 2018.

[17] M. Su et al. Generative adversarial networks as a tool to recover structural information from cryo-electron microscopy data. *BioRxiv*, 2018.

[18] H. Zhang et al. ZipML: Training linear models with end-to-end low precision, and a little bit of deep learning. In *ICML*, 2017.