

# Enriching Machine Learning Model Metadata

Collecting performance metadata  
through automatic evaluation

Hendrik G. J. Kant



# Enriching Machine Learning Model Metadata

Collecting performance metadata through  
automatic evaluation

by

Hendrik G. J. Kant

to obtain the degree of Master of Science

at the Delft University of Technology,

to be defended publicly on Monday May 8th, 2023 at 4:00 PM.

Student number: 4024400  
Project duration: March 1, 2022 – May 8, 2023  
Thesis committee: Prof. dr. ir. A. Bozzon, TU Delft, thesis supervisor  
Dr. A. Katsifodimos, TU Delft, daily supervisor  
Dr. R. Hai, TU Delft, daily supervisor  
ir. Z. Li, TU Delft, daily co-supervisor  
Dr. ing. S. Proksch, TU Delft

An electronic version of this thesis is available at <http://repository.tudelft.nl/>.



# Preface

The completion of this thesis marks the ending of my studies at the TU Delft. This project has been one of the most challenging tasks I have faced in my life, which makes the satisfaction of having completed it all the greater.

Machine learning was not an integral part of my course load, and I was not familiar with the topic of model zoos before starting this project, but have found it an intriguing topic nonetheless. In all the projects I've done, during my studies as well as professionally, I have always tried to be mindful of the impact they have on actual people, and the real problems they try to solve. With model zoos being very end-user oriented applications, and still in their infancy, I am glad I was able to contribute to this topic in a meaningful way.

I would like to take this opportunity to thank Alessandro Bozzon, to whom I was a student and later assistant, and who was supervisor of the final project for my Bachelor's degree, which solidified him as my choice of supervisor for this project as well. In every capacity, your feedback has been invaluable and your enthusiasm infectious, and has motivated me throughout this final project. I also want to thank Ziyu Li, who has guided me to completion of this project with her feedback provided during our many meetings from start to finish, Asterios Katsifodimos and Rihan Hai for providing feedback during pivotal moments of this project, and Sebastian Proksch for serving as external committee member during the assessment of my work.

Furthermore, I would like to thank my supporting parents, who have encouraged but never pressured me during the many, many years it has taken me to complete my studies. Finally, I would like to thank my girlfriend, Asiye, who has kept me going during this project and was here for me, even when my mood was down and my mind pre-occupied.

Hendrik G. J. Kant  
Delft, April 2023



# Contents

<b>1</b>	<b>Introduction</b>	<b>1</b>
1.1	Limitations in current model zoos . . . . .	2
1.2	Prospects for model zoo improvements . . . . .	3
1.3	Potential use-cases. . . . .	4
1.3.1	UC1: Model comparison and selection. . . . .	4
1.3.2	UC2: Automated model selection for solving complex tasks . . . . .	4
1.4	Objective and research questions. . . . .	5
1.5	Contributions . . . . .	5
1.6	Outline . . . . .	5
<b>2</b>	<b>Related Work</b>	<b>7</b>
2.1	Model zoos . . . . .	7
2.2	Evaluation . . . . .	8
2.3	Model Cards. . . . .	9
2.4	Robustness . . . . .	9
<b>3</b>	<b>Framework Design</b>	<b>11</b>
3.1	Framework architecture . . . . .	11
3.1.1	Scraper . . . . .	12
3.1.2	Model Evaluation . . . . .	13
3.1.3	User Interface. . . . .	13
3.2	Metadata model. . . . .	14
3.2.1	Model metadata . . . . .	14
3.2.2	Dataset metadata. . . . .	15
3.2.3	Evaluation metadata . . . . .	17
3.2.4	Reference . . . . .	17
<b>4</b>	<b>Model evaluation</b>	<b>19</b>
4.1	Evaluation design goals . . . . .	19
4.2	Evaluation Subcomponents . . . . .	21
4.3	Evaluation Process . . . . .	23
<b>5</b>	<b>Analysis of collected performance metadata</b>	<b>27</b>
5.1	Performance metadata availability . . . . .	27
5.1.1	FiftyOne . . . . .	30
5.1.2	Hugging Face. . . . .	32

---

<b>6</b>	<b>User Interface</b>	<b>35</b>
6.1	Datasets . . . . .	35
6.1.1	Dataset information . . . . .	35
6.1.2	Scatterplot. . . . .	39
6.2	Models . . . . .	42
6.2.1	Browsing models . . . . .	42
6.2.2	Model information . . . . .	43
6.2.3	Performance results . . . . .	45
6.2.4	Variant comparison. . . . .	50
6.2.5	Model performance comparison . . . . .	50
<b>7</b>	<b>Conclusion and limitations</b>	<b>55</b>
7.1	Fulfillment of objective . . . . .	55
7.2	Limitations of provided solution . . . . .	56
7.3	Suggestions for future research . . . . .	57



*Dedicated to our cat Coco,  
with whom I couldn't spend enough time during her developing illness  
for most of this project*



# 1

## Introduction

As Machine Learning (ML) becomes more popular, more and more ML models are being created. In recent years, sharing these models has gained popularity, and has given rise to so called model zoos, repositories of ML models. There may be several reasons to share a ML model. A shared model can be re-used in multiple different applications. As the resources required, both time and computational, are increasing to create state-of-the-art models, significant savings can be had by re-using an existing model. Sharing a model also allows others to verify claims made about the model.

Several model zoos have gained popularity, such as PyTorch Hub<sup>1</sup>, TensorFlow Hub<sup>2</sup>, FiftyOne<sup>3</sup>, and — to the best of our knowledge — the largest to date, Hugging Face<sup>4</sup>, which contains over 160.000 models at the time of writing this report, having tripled in size since the start of this thesis. In these model zoos, users can find ready-to-use models, often categorized by task, training dataset, or ML framework. Users can browse and search for specific models, read provided information, and oftentimes start using the model quickly through an intuitive programmable interface. In some cases, datasets and information about them is also included, as well as tools for users to create models themselves.

Besides the models, metadata of the models is shared as well. Though there are varying definitions of metadata (Furner, 2020), for our purpose we will follow use the definition of "data about data" (Riley, 2017). In our context, the artifact that is described by the metadata is the ready-to-use model, and metadata of that model commonly includes information about who created it, how they did it, what the model does, and how well it does it. Metadata can take any form (e.g. textual, visual), but in current model zoos is often limited to textual information. The sharing

---

<sup>1</sup><https://pytorch.org/hub/>

<sup>2</sup><https://www.tensorflow.org/hub>

<sup>3</sup>[https://docs.voxel51.com/user\\_guide/model\\_zoo/index.html](https://docs.voxel51.com/user_guide/model_zoo/index.html)

<sup>4</sup><https://huggingface.co/>

of metadata allows other users to understand how that model works, thus providing a degree of transparency and explainability of the model in question. Until recent there was no clear standard of sharing a model’s metadata. Mitchell et al., 2019 sought to change this with a proposed to share a model’s metadata in so-called model-cards. This framework allows a standardized way to report all aspects related to the creation and use of the model, such as it’s training data and methods, performance evaluation results, and ethical considerations, both in textual and visual form. Most model zoos include a version of a model card, though the information included varies greatly, both between models in the same zoo, as well as what seems to be the standard for reporting between model zoos.

The proliferation of model zoos has made it uncomplicated to share ML models. However, in their current form, many model zoos provide little functionality beyond this. While it easy to find any model to fulfil a task, it remains difficult to find a model that fits specific needs. For users looking for a ready-to-use model, naturally information about the model that describes its behavior and performance is of great value. However, in current-day model zoos the parts of model cards that describe this aspect of a model — performance, limitations, ethical considerations — are neglected. Instead, they oftentimes provide merely enough information to determine if a model can be used for a given task, and a general description of a model, if that. Performance metadata, if provided, is limited to a handful of values, thus barely providing potential users with a sufficient understanding of the model’s performance, with other aspects such as limitations and ethical considerations receiving little to no attention at all.

In this thesis we focus on the availability of performance metadata in the model zoos’ model cards. Throughout this thesis, we describe a framework for a model evaluation pipeline, which overcomes the main problems surrounding performance metadata: the availability and richness of this data in current day model zoos. We also provide a reference benchmarking system built using our framework and extract rich performance metadata for 986 models from the Hugging Face and FiftyOne model zoos, and a new interface to view the metadata obtained with our benchmarking system.

## 1.1. Limitations in current model zoos

As mentioned, there are two main problems surrounding performance metadata in model cards: the availability and richness of this metadata. There are multiple aspects that contribute to these problems.

Though it varies between and within model zoos, metadata is often missing completely from the models’ model cards. Over the course of this thesis we have evaluated 1215 models from external model zoos (see Chapter 5 for our results). For over 25% of these models, no performance metadata was provided in the original model card.

If performance results are provided, they are, in most cases, limited to aggregated results. For example, for classification models the `precision` metric over a complete dataset is sometimes given, but not for each of the classes that can be classified by the model. If this is the case, they

provide only a limited understanding of a model’s performance, and hide possible performance disparities (Barocas et al., 2021).

An additional cause to these problems is the unstructured nature of performance reporting, for example by providing performance results in the model’s description, rather than a predefined format separately in the model card. This forces the user to scan a potentially long description for the desired information, and limits access to this information by third parties.

There is also a strong reliance on self-reported values, which come with inherent trust issues in the validity of the results, by requiring the creator of the model to provide these values when sharing the model in a model zoo. The creator might not want to spend the time to gather these values, or simply not know how, and thus omit them from the model card.

Each of these aspects also contributes to a lack of comparable performance metadata, thus preventing users from comparing models with each other, and being able to find the best performing model for their specific needs.

## 1.2. Prospects for model zoo improvements

As we have pointed out several shortcomings surrounding the performance metadata provided by current model zoos, we would like to use this section to describe what a model zoo that overcomes these problems might look like.

The first noticeable change for such an improved model zoo is observed when browsing for a model. Besides the options available in current model zoos to filter and search for models. As more performance metadata is present for models, new options will be available to filter models with more complex queries. These might include filters for evaluation on a certain dataset or the availability of a desired metric in the performance metadata. There might be more fine-grained search filters as well, such as for the training on, and evaluation of specific data classes (e.g. "show me models that were evaluated on pictures containing cats"), or attributes of the data objects used during classification (e.g. "show me models evaluated with images smaller than 50x50 pixels"). For these improvements over current model zoos, detailed information about the datasets, as well as the metrics used during evaluation need to be recorded in the evaluation process.

Further improvements would be observed in the model cards. Naturally, performance results would be provided in a structured manner, be disaggregated, comparable, and verified. As more metadata is available, model cards include visualizations to present the data in an attractive and informative way. Such visualizations will be interactive, with options to filter, sort, and zoom in on specific parts of the data as users please. Besides the presentation of metadata collected during evaluation, higher-level observations that describe a model’s behavior may be presented in the model card as well. For example, based on the evaluation results, it could be said about an image-classification model that it performs generally well on pictures of animals, or that a text-classification model errs on the more positive side overall.

As the performance metadata collected in our envisioned model zoo is comparable, it will include

a model comparison tool. A comparison can be made here for a handful of models, letting the user compare the models in fine detail, based on their (disaggregated) performance results, but also other model attributes, such as the speed and size of the model.

These improvements over current model zoos would enable new use-cases, of which we describe two in the following section.

### 1.3. Potential use-cases

The presence of rich metadata in model zoos would enable new use-cases. In this section we describe two use-cases that would be enabled by, and depend on, the presence of such performance metadata.

#### 1.3.1. UC1: Model comparison and selection

With the presence of increasingly rich performance metadata, users will be able to find an optimal model according to increasingly specific needs. For example, consider a user with the following classification need:

*“I have 10,000 black and white pictures and am interested in pictures containing bears. I want to minimize the false-negatives, and the process should be completed within one hour.”*

To obtain an optimal model to solve their problem, the user needs comparable performance metadata for each of the models that can solve this task. This metadata needs to be rich enough to provide the user with the information for the metric they are interested in, `recall`. The performance metrics also need to be disaggregated; the best performing model overall may not be best at classifying bears, which the user is particularly interested in. As there is a time constraint, the inference speed is of interest as well. As this particular metric depends on the hardware used during model evaluation, special care needs to be taken to ensure it remains comparable with other models. Finally, the user’s pictures are black-and-white, whereas most imaging datasets use color pictures. The performance of models trained on such datasets will be impacted by this difference, in varying degrees. Only by also evaluating models outside of their intended use can the performance metadata be rich enough to find an optimal model for the user’s need.

#### 1.3.2. UC2: Automated model selection for solving complex tasks

Each ML model performs a single task, yet more complex queries may exist that require a composition of models to answer. Furthermore, while each model has an average performance, they might perform better or worse in certain aspects in their domain. For example, a sentiment-analysis model — a model deriving mood from an input text — might be better at detecting anger than happiness, an object-detection model — a model identifying objects in images — might perform better at detecting chairs than airplanes. Tools build on top of a model zoo, that support complex queries such as

*Find pictures of the groom with the wedding cake*

requires the disaggregate performance information and limitations of each model to be able to create an optimal query execution path utilising one or more models.

## 1.4. Objective and research questions

The objective of this thesis is to find a means to provide performance metadata that is richer than that provided by current solutions. Our aim is to improve upon the current situation in a way that enables the use-cases described in Section 1.3. To do so, first the limitations and drawbacks of current solutions need to be identified, resulting in our first research question:

**RQ1:** What are the limitations of provided metadata in current model zoos?

After answering this question, we can move to devise a solution that overcomes the found limitations. Hence, our second research question:

**RQ2:** How can we provide rich and comparable performance metadata for machine learning models

## 1.5. Contributions

In the process of answering both research questions, we make several contributions to the field:

- We identified a need for richer performance metadata for machine learning models unaddressed in current solutions
- We devised a framework for a model evaluation pipeline, with which one can create a benchmarking system tailored to automate model performance evaluation for a specific set of machine learning models
- We delivered a reference implementation of a benchmarking system built using our framework, providing rich performance metadata for 986 models from the Hugging Face and FiftyOne model zoos
- We presented an interactive interface, showing many visualizations for both dataset and model metadata, with the latter uniquely enabled by our benchmarking system

The interactive interface is available at <https://www.metadataazoo.io>.

## 1.6. Outline

Chapter 2 gives an overview of existing related literature. Chapter 3 gives an overview of our framework for an evaluation pipeline devised during this thesis, with Chapter 4 describing in more detail the model evaluation component of our framework. In Chapter 5 we outline the results achieved with our implementation of a benchmarking system, built using our framework, where we compare the performance metadata collected during evaluation of 1215 models available in the Hugging Face and FiftyOne model zoos with the performance metadata provided by those model zoos. Chapter 6 describes and shows our user interface to view the metadata collected by our benchmarking system, with many visualizations made possible only by the collection of

this metadata. Finally, in Chapter 7 we provide some concluding remarks and options for further research into this field.



# 2

## Related Work

### 2.1. Model zoos

In recent years several model zoos have gained popularity. The Hugging Face Model Hub Wolf et al., 2020 (Hugging Face) is a combination of website and API through which users can explore and use A large number of models. Initially containing exclusively Transformers based NLP models and related datasets, through submission of new models by end-users it has steadily grown to include pre-trained<sup>1</sup>, fine-tuned models, and datasets for other tasks and libraries as well. While Hugging Face is — to the best of our knowledge — the largest to date, with over 160.000 models at the time of writing this thesis, many more exist, such as model zoos containing models created with a particular ML framework, such as PyTorch Hub<sup>2</sup> for PyTorch (Paszke et al., 2019) models and TensorFlow Hub<sup>3</sup> for models created with the TensorFlow (Abadi et al., 2016) framework, model zoos associated with software packages, such as the FiftyOne model zoo<sup>4</sup>, usable with the FiftyOne software package<sup>5</sup> and the Open Model Zoo for OpenVINO toolkit<sup>6</sup>, containing example models to be used in the OpenVINO toolkit<sup>7</sup>. While there is a variety of model zoos, many features are shared between popular model zoos. This includes, but is not limited to, the categorization of models (e.g. by task, framework, training dataset), browsing and searching models, a description of each model, access to datasets and their metadata, and APIs or software packages to start using the models.

---

<sup>1</sup>A partially trained model that can be used as starting point to fully train a model

<sup>2</sup><https://pytorch.org/blog/towards-reproducible-research-with-pytorch-hub/>

<sup>3</sup><https://www.tensorflow.org/hub>

<sup>4</sup>[https://docs.voxel51.com/user\\_guide/model\\_zoo/index.html](https://docs.voxel51.com/user_guide/model_zoo/index.html)

<sup>5</sup><https://docs.voxel51.com/index.html>

<sup>6</sup>[https://github.com/openvinotoolkit/open\\_model\\_zoo](https://github.com/openvinotoolkit/open_model_zoo)

<sup>7</sup><https://docs.openvino.ai/>

## 2.2. Evaluation

A machine learning model’s performance is evaluated based on task-specific metrics. For example, precision, recall, and accuracy tend to be used for classification tasks, BLEU Papineni et al., 2002 and metrics based on BLEU such as METEOR Banerjee and Lavie, 2005 and ROUGE Lin, 2004 for various natural-language-processing tasks such as machine-translation and summarization, and intersection-over-union for object-detection tasks.

Using such metrics is not without limits. Accuracy is known to be a bad metric when using imbalanced datasets, and according to a study by Ferri et al., 2009, most classification metrics tend to measure different aspects of a model. Similarly, BLEU has been shown to have limitations (Callison-Burch et al., 2006, Reiter, 2018, Mathur et al., 2020).

It has been shown that problem exist in the way that evaluation results for ML models have been reported as well. Raff, 2019 attempted to reproduce results from 255 selected papers and found that 63% were reproducible. A similiary study by Blagec et al., 2020 using 3867 from the PapersWithCode<sup>8</sup> platform and found that a large majority of metrics used have properties that do no accurately describe the model’s performance. The used metrics themselves are also a problem. Marie et al., 2021 conducted a study of 769 papers related to machine-translation and found that nearly all of them (98,8%) still use this metric, despite numerous newer metrics being proposed and accepted as superior.

In order to simplify evaluation of models, several libraries have been created with such features. Some are framework-specific such as TorchMetrics (Detlefsen et al., 2022) for PyTorch or Keras<sup>9</sup> for TensorFlow. Other libraries exist that provide evaluation features such as Scikit-learn<sup>10</sup>, SciPy<sup>11</sup>, NLTK<sup>12</sup>, and StatsModels<sup>13</sup>. Furthermore there are standard implementations for individual metrics such as BARTScore (Yuan et al., 2021), SacreBleu (Post, 2018), or rouge\_score<sup>14</sup>. Some model zoos have included tools to evaluate their models as well. FiftyOne includes tools to evaluate models and then visualize the results, for example the bounding boxes of an object-detection task, through their software. Hugging Face released the evaluate library (von Werra et al., 2022) in an attempt to overcome the many different and incompatible metrics and methods of evaluation. Tied in to the Hugging Face Model Hub, it enables users to evaluate any supported combination of model and dataset from the Model Hub.

Finally, evaluation-as-a-service has gained traction, where evaluation of models is offloaded to a third-party (Yadav et al., 2019, Kiela et al., 2021, Ma et al., 2021, Liu et al., 2021, Coleman et al., 2017). Letting a designated party do the evaluation also enables holding challenges (Coleman et al., 2017, Russakovsky et al., 2015, Yadav et al., 2019), rather than comparing self-reported results. Centrally evaluating models has several advantages, such as increased reproducibility,

---

<sup>8</sup><https://paperswithcode.com/>

<sup>9</sup><https://keras.io/>

<sup>10</sup><https://scikit-learn.org/>

<sup>11</sup><https://scipy.org/>

<sup>12</sup><https://scipy.org/>

<sup>13</sup><https://www.statsmodels.org/>

<sup>14</sup><https://github.com/google-research/google-research/tree/master/rouge>

compatibility and trust regarding the results (Willis and Stodden, 2020), and move the fields towards a better approach to evaluation.

## 2.3. Model Cards

Mitchell et al., 2019 proposed a framework to standardize the metadata of models in so called model cards, similar to a different proposal to standardize dataset information (Gebru et al., 2021). Model card are intended to give any stakeholders information of the model in question in order to improve understanding of that model. the required information regarding the model, such as it's details, intended use, performance, and ethical considerations. Most model zoos include such a model card for each of their models, with varying levels of detail.

Crisan et al., 2022 build upon this to propose interactive model cards. Interactive model cards are extended with interactive elements, which let a user interact with a model by, for example, submitting own data to test the model or using interactive visualizations to explore the performance evaluation results.

As part of model cards, Mitchell et al., 2019 advocate for the reporting of disaggregated evaluation results. This is supported by other works, such as Barocas et al., 2021, recognizing that there may be a performance disparities in subgroups of datasets, for example for different races and genders in classification tasks (Buolamwini and Gebru, 2018), which are hidden by aggregated performance reporting.

Model zoos may provide a template for a model card to work with their own system. For example Hugging Face provides a template<sup>15</sup> which allows users to provide a description of the model, specify tags to categorize the model, or provide performance results that are shown separately in their user interface.

## 2.4. Robustness

Tangentially related to our project is the concept of robustness of a ML model. While there is no singular definition, it can broadly be defined as the model's retained performance when using corrupted or perturbed input. Several studies have been conducted regarding the impact of such input, showing that for certain types of models the performance takes a significant hit when using perturbed inputs (Akhtar and Mian, 2018, Dziugaite et al., 2016, Engstrom et al., 2017), noting that perturbations too small to notice are sometimes enough to trick the model. Others have also investigated how model robustness can be improved (Djulonga et al., 2021), and have created standardized benchmarks to benchmark robustness (Hendrycks and Dietterich, 2019).

For the purpose of this project, we note the importance of the robustness of a model in the context of completing a model card. As, in our case, the information is used by end-users in order to understand and compare models, we focus primarily on the use of perturbations that reflect real-world use-cases, rather than corrupted inputs.

---

<sup>15</sup><https://github.com/huggingface/hub-docs/blob/main/modelcard.md?plain=1>



# 3

## Framework Design

In this chapter we describe the architecture of our framework for a model evaluation pipeline. This framework can be used to implement a benchmarking system to evaluate models from any source, locally or external, which is able to automatically extract rich performance metadata for those models. The framework is designed such that a benchmarking system built with it, by design, overcomes the problems outlined in Section 1.1. As part of this thesis, we have implemented a benchmarking system using our framework, able to evaluate models from the Hugging Face and FiftyOne model zoos. In Chapter 5 we compare the performance metadata obtained for 1215 models by our benchmarking system, to the performance metadata already present in those external model zoos.

In Section 3.1 we describe the architecture of our framework. In short, the main components are (i) metadata storage, (ii) dataset storage, and (iii) a model evaluation component. For the needs of this thesis we have included a scraper to collect model metadata from external sources rather than adding our own, and a user interface to view the stored metadata, though how initial model metadata is obtained, and performance metadata is accessed, is not dictated by our framework. Finally, we outline our data model in Section 3.2.

### 3.1. Framework architecture

Our framework consists of a few main components, displayed in Figure 3.1. The metadata for each model and dataset (detailed in Section 3.2) are stored in metadata storage, with the files for datasets stored in a dataset storage. We obtain performance metadata for each model through a model evaluation component, described briefly in Section 3.1.2 and in greater detail in Chapter 4. For our benchmarking system we have created a scraper, described in Section 3.1.1, to obtain model metadata from external model zoos, and a user interface, described briefly in Section 3.1.3 and in greater detail in Chapter 6, to display the metadata obtained by the benchmarking system.

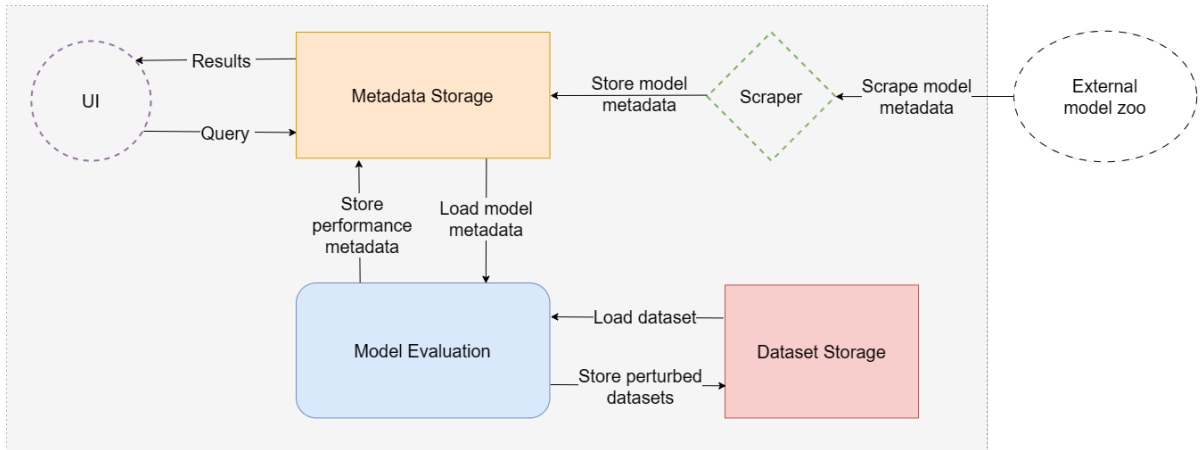


Figure 3.1: The subsystems of our model evaluation pipeline framework. Our implementation of a benchmarking system included the use of a scraper and user interface. Though not part of our framework, they have been included in the presented architecture to show the interactive between them and the components of our framework

Though not part of the framework itself, we have included a description of each of these systems in this section as well, and included them in Figure 3.1 to show the interaction between the scraper, the user interface, and the components of our framework.

### 3.1.1. Scraper

In this section we describe the scraper used to populate our metadata storage with metadata retrieved external sources. For our implementation of a benchmarking system, we have scraped model metadata from the Hugging Face and FiftyOne model zoos. For each source the metadata for unknown models is stored in our storage, and metadata for models that do not exist anymore in the external source is deleted. Initially, for each model only the model’s name and task are copied, and the origin of the model is noted. For the purpose of this thesis, our benchmarking system supports a subset of all scraped models for evaluation. In the interest of speeding up the scraping process, only for these models the complete available metadata — as far as relevant to, or required for evaluation — is obtained from the external source. See Section 5.1 for details on which models are included in this subset. For these models, any tags from the model’s external model card, containing information such as framework, training dataset supported languages, are retrieved. The training dataset is mapped to a dataset known in our system, if possible, and a reference is included in the model’s metadata. If available, the size of the model is recorded as well. If a reference to an external published paper present in the model’s description, it is extracted, stored, and added to the model’s metadata. The full textual description is not copied but rather retrieved when the model’s metadata is accessed. This is done to always retrieve the most current description, as they may change over time. The external metadata may be obtained through an API or scraped from a web page. As support for more models may be added in the future for our benchmarking system, the described metadata can also be added after the initial scrape. See Table 3.1 for an example of scraped metadata for a model.

We use the scraper also to obtain metadata for with datasets when possible. For each dataset, we

include the name, description, year, version, link to homepage, logo, and other basic information. If relevant for the dataset, we include a id-to-label mapping for classes represented in the dataset. Most datasets are divided in subsets, each used for a different task such as model training or evaluation. For each of these so-called splits we record the name, total number of objects in the split, and, for classification datasets, the distribution of the objects over each classifiable class. If applicable, up to 20 examples of data objects are stored, again per classifiable class for classification datasets. See Tables 3.2a to 3.2d for examples of scraped metadata for datasets.

### 3.1.2. Model Evaluation

Our framework addresses the problems described in Section 1.1 through the inclusion of a model evaluation component. The model evaluation component removes the need for self-reported performance metadata, or any user interaction besides the sharing of the model, as it allows continuing evaluation of all models represented in our metadata storage. This is made possible by, during the evaluation of a model, assessing which metadata is already stored for that model, and only obtaining metadata that is not present yet.

Through the use of re-usable components for metric calculation, the model evaluation component delivers comparable performance metadata, even between models from different sources. To obtain richer performance metadata, it enables evaluation using multiple dataset. We borrow the idea from model robustness to evaluate with perturbed versions of each dataset as well. The evaluation component defines standardized perturbation techniques, in order to evaluate a model's performance with input deviating from the training dataset, and providing the user with additional performance metadata. These dataset perturbations are handled automatically by the evaluation pipeline, creating and re-using perturbed datasets as needed.

For a more detailed description of the model evaluation component and evaluation process, see Chapter 4.

### 3.1.3. User Interface

The user interface included for this thesis allows users to explore and compare the rich metadata obtained by our benchmarking system. When viewing datasets, the user can view examples of data instances of this dataset, if they exist, information about the splits of the dataset, with information about the objects in it, a selection of models that are evaluated using this dataset is shown, and task-related information, such as the classes for datasets used to train classification models..

The user can browse models, and apply filters to search for a specific model. For each model we have created a model card: a combination of the description from the model card from the source model zoo, the scraped metadata, and several new interactive elements to visualize the model's performance results. This includes several diagrams allowing the user to view performance of a model, for a whole dataset, or subsections or perturbations thereof, task-specific representations such as a confusion matrix and misclassifications, and scatter plots to get a higher-level overview of the performance of a large number of models.

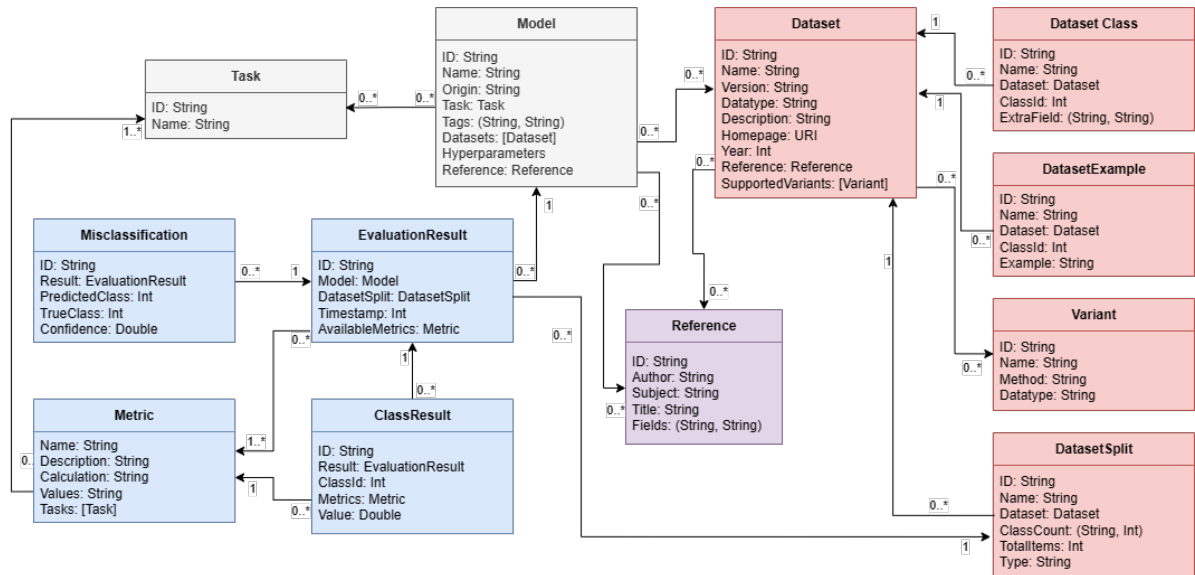


Figure 3.2: The metadata model. The color of each entity indicates which metadata it holds: grey describe models, red describe datasets, blue describe evaluation results, and the purple entity holds references to external resources.

The visualizations in our model card used to display the information regarding both datasets and models are interactive, allowing the user to filter, zoom, or disable parts of the data they are not interested in. For a look at the user interface and its various visualizations, please see Chapter 6.

## 3.2. Metadata model

In this section we describe the data model used to store model and dataset metadata. For our data model we took inspiration from other publications such as Schelter et al., 2017 and Li et al., 2022, but adapted the model to fit our needs. Figure 3.2 depicts our data model. The entities in the data model are color coded according to the information they relate to. They can be divided into metadata related to models (grey entities, described in Section 3.2.1), datasets (red, described in Section 3.2.2) and evaluation results (green, Section 3.2.3), with reference (purple, Section 3.2.4) being a standalone entity.

### 3.2.1. Model metadata

The entities containing metadata related to models are colored grey in our data model. The `Model` entity contains all information related to the model itself. This always contains a unique identifier, the name of the model, and the source of the model. Additionally, if known, it contains the task the model performs, a reference to the datasets it is trained on, and other information important to be able to find the appropriate model in the zoo. It has a reference to the `Task` entity, which contains only the name of a task. See Table 3.1 for an example of stored metadata for a `Model`.



Field	Value
<code>_id</code>	63a0abd915a79b5cfb175bcc
<code>name</code>	facebook/detr-resnet-101-dc5
<code>task</code>	object-detection
<code>datasets</code>	[6325c7632aa8981c9d3b411f]
<code>tags</code>	{dataset: [coco], library: [PyTorch], ...}
<code>files</code>	{pytorch.model.bin: 243011543, ...}
<code>config</code>	{backbone: resnet101, ...}

Table 3.1: An instance of the `Model` entity

### 3.2.2. Dataset metadata

Entities related to datasets (colored red) contain all information related to the stored datasets. The main entity is the `Dataset` itself. This entity describes a dataset which was used to train models, and can be used to evaluate models. It contains the name and version of the dataset, and the type of data in the dataset, for example images or text. If available, further information such as a description and a homepage URL can be stored here too. For each dataset we also record compatible datasets. A compatible dataset means that a model which is trained on this dataset, can also be evaluated using any compatible dataset. This allows us to expand the performance results collected for each model, which in turn gives the user more information. Finally, for each dataset we note the variants, if any, that can be created to use during evaluation. From these values our benchmarking system can automatically create the required variants during the evaluation process. Table 3.2a shows an example instance of the `Dataset` entity.

For many datasets, each object in it is labelled as a particular class of objects. For example, the CIFAR-10 image-classification dataset (Krizhevsky, Hinton, et al., 2009) assigns to each of its 60,000 images one of the ten classes. Similarly, the IMDB text classification dataset contains 50,000 movie reviews, each labeled as either a positive or negative review. This information is described by the `DatasetClass` entity. This entity contains a reference to the dataset it belongs to, and a numerical identifier and textual label of the class. Additionally it might contain extra information, such as a higher level label for the class. Table 3.2b shows an example instance of the `DatasetClass` entity.

Most datasets are partitioned in disjoint subsets, called splits. Datasets are commonly divided in a training split to train models with, and a test and/or validation split to evaluate models with. The `DatasetSplit` entity contains information to describe any splits available for a dataset. This entity has a reference to the dataset it pertains to, the type of split (training, test, etc.), the number of objects in the split, and, if known, the division of objects per `DatasetClass` of the dataset. Table 3.2c shows an example instance of the `DatasetSplit` entity.

The `DatasetExample` contains descriptions of examples of the objects in the dataset. Besides a reference to the dataset itself, it contains the example and optionally a reference to a class of the dataset. The example might be a reference to a media file (for datasets of those types), or a text

snippet (for textual datasets). If a class is referenced, the example shows an object for the given class. If not, the example shows more classes, for example a collage of images for each class in the dataset. Table 3.2d show an example instance of the `DatasetExample` entity.

Finally, the `Variant` describes each of the available variants. Each variant is for a single data type, and it contains a description of the method of perturbation, so as to increase reproducibility of our results. Table 3.2e shows an example of the `Variant` entity.

Field	Value	Field	Value
<code>_id</code>	6325c78d2aa8981c9d3b48bb	<code>_id</code>	6325c7632aa8981c9d3b4124
<code>name</code>	imagenet	<code>dataset_id</code>	6325c7632aa8981c9d3b411f
<code>version</code>	2012	<code>name</code>	bicycle
<code>datatype</code>	IMAGE	<code>classId</code>	2
<code>description</code>	The ImageNet project is...	<code>extra_fields</code>	{superCategory: vehicle}
<code>homepage</code>	<a href="https://www.image-net.org/">https://www.image-net.org/</a>		
<code>year</code>	2012		
<code>reference</code>	6325c7752aa8981c9d3b449c		
<code>compatible_with</code>	[63a0cf8b22c94be4b5751187]		
<code>logo</code>	<a href="https://www.image-net.o...">https://www.image-net.o...</a>		
<code>supported_variants</code>	[greyscale, flipped, invert...		

(a) An instance of the `Dataset` entity(b) An instance of the `DatasetClass` entity

Field	Value
<code>_id</code>	6329b25b0f1513e306f0fe5a
<code>dataset_id</code>	6325c7892aa8981c9d3b488b
<code>split_type</code>	TRAIN
<code>total_items</code>	25000
<code>class_counts</code>	{1: 12500, 2: 12500}

(c) An instance of the `DatasetSplit` entity

Field	Value
<code>_id</code>	6325c78a2aa8981c9d3b4892
<code>dataset_id</code>	6325c7892aa8981c9d3b488b
<code>example</code>	"I rented I AM CURIOUS..."
<code>class_id</code>	0

(d) Instance of the `DatasetExample` entity for a specific class of the dataset

Field	Value
<code>_id</code>	63fb9aedb25eb0fc34626de9
<code>variant</code>	rotated90
<code>description</code>	The image is counter-clockwise rotated 90 degrees
<code>method</code>	<code>PIL.Image.transpose(Transpose.ROTATE_90)</code>
<code>datatype</code>	IMAGE

(e) An instance of the `Variant` entity

Table 3.2: Example instances of each entity related to dataset metadata

### 3.2.3. Evaluation metadata

Entities related to model evaluation are colored blue in the data model. The main entity is the `EvaluationResult`. It links together all aspects of a model's evaluation step. Besides references to both the model and dataset split, this entity contains a timestamp of evaluation, the metrics that have been calculated during processing of the results, and optionally a perturbation used during evaluation. Table 3.3a show an example instance of the `EvaluationResult` entity.

The `ClassResult` entity describes the actual performance results. Besides a reference to the `EvaluationResult` it pertains to, it holds a combination of a metric and its value. Additionally, a class id may be present if the given value indicates the model's performance for a specific class. If no class id is given, the value indicates the average for the whole dataset. Table 3.3b shows an example instance of the `ClassResult` entity.

The `Misclassification` describes classification errors made during model evaluation. Besides the confidence and object under consideration while making the mistake, it holds the predicted and true classes — only one of which is required. The nature of the mistake can be determined from the presence of the predicted and/or true classes. If the `PredictedClass` field is set, it was a false positive, meaning the model thought it was the predicted class, but it was not. If the `TrueClass` is set, it was a false negative, meaning it was that class, but the model thought otherwise. Note that in a classification setting there is always both a true class and predicted class given, but this is not required. Table 3.3c shows an example instance of the `Misclassification` entity.

The `Metric` describes the various metrics our system can calculate. For each `Metric`, we provide a simple description, and both the means of calculation and values that can occur. Each metric is linked to specific `Tasks` for which it can be used. Table 3.3d show an example of a `Metric` entity.

Finally, the `Runtime` describes the model's speed during evaluation. This entity holds a reference to an `EvaluationResult`, the total time it took to evaluate all objects in the dataset, and, to same needed conversions based on the desired representation, both the average time it took to evaluate an object of the dataset and the average number of objects evaluated per second. See table 3.3e for an example instance of the `Runtime` entity.

### 3.2.4. Reference

The `Reference` entity is another singular entity, used to store information pointing to a scientific publication. Both the `Model` and `Dataset` entities can hold a reference to a `Reference` entity. Table 3.4 shows an instance of the `Reference` entity.

Field	Value
<code>_id</code>	63a24e65fe72489ce66a946c
<code>timestamp</code>	1671581027
<code>model_id</code>	62bdfe83465b811023e311d9
<code>dataset_split</code>	6329b25b0f1513e306f0fe5b
<code>metrics</code>	[PRECISION, RECALL, ...]
<code>perturbation</code>	greyscale

(a) Instance of the EvaluationResult entity.

Field	Value
<code>_id</code>	63a22df80f95c6eaa292283c
<code>result_id</code>	63a22df80f95c6eaa2922838
<code>classId</code>	5
<code>metric</code>	F1SCORE
<code>value</code>	0.9876

(b) Instance of the ClassResult entity. If the `classId` field is omitted, the entity described average performance over all classes

Field	Value
<code>_id</code>	63a22dfb0f95c6eaa29228cb
<code>result_id</code>	63a22df80f95c6eaa2922838
<code>instance</code>	automobile_s_001582.png
<code>detected_class</code>	9
<code>true_class</code>	1
<code>confidence</code>	0.5762876868247986

(c) Instance of the Misclassification entity. Only one of `detected_class` and `true_class` is required

Field	Value
<code>_id</code>	63fb9aedb25eb0fc34626de6
<code>metric</code>	PRECISION
<code>calculation</code>	$(TP + FN) / TP$
<code>values</code>	Range between zero (min...
<code>tasks</code>	[text-classification, image...]

(d) Instance of the Metric entity

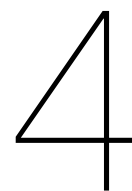
Field	Value
<code>_id</code>	63a22dfd0f95c6eaa2922936
<code>result_id</code>	63a22df80f95c6eaa2922838
<code>time_in_seconds</code>	87.92572036699858
<code>time_per_instance</code>	0.008792572036699857
<code>instances_per_second</code>	113.73236361624772

(e) Instance of the Runtime entity

Table 3.3: Example instances of each entity related to model evaluation metadata

Field	Value
<code>_id</code>	63a0cf8b22c94be4b5751186
<code>type</code>	inproceedings
<code>name</code>	wang2019learning
<code>author</code>	Wang, Haohan and Ge, Songwei and Lipton, Zachary and Xing, Eric P
<code>title</code>	Learning Robust Global Representations by Penalizing Local Predic...
<code>year</code>	2019
<code>booktitle</code>	Advances in Neural Information Processing Systems
<code>pages</code>	10506–10518

Table 3.4: Instance of the Reference entity



# Model evaluation

In this chapter we describe how models are evaluated in the model evaluation component of our framework. As the name suggests, this component performs the actual evaluation of the models, and obtains the performance metadata for them. It was designed such that it addresses the problems outlined in Section 1.1. In Section 4.1 we describe the goals set for the model evaluation component, which include ways to address the aforementioned problems, but should also allow for the creation of a maintainable and extensible benchmarking system. Then, Section 4.2 describes the subcomponents of our model evaluation component and how they work together to evaluate models. Finally, in Section 4.1 we detail the evaluation process itself.

## 4.1. Evaluation design goals

When designing our framework, several goals were kept in mind. We strived to complete the following goals with our model evaluation component, and thus the framework as a whole:

**DG1 Remove burden from user:** Remove burden to provide model performance metadata from the model creator

**DG2 Metadata for high degree of models:** Enable obtaining performance metadata for a high degree of models

**DG3 Comparable performance results:** Obtain comparable performance results between models, also from different sources

**DG4 Richer metadata:** Encourage obtaining richer metadata than provided by current model zoos

**DG5 Flexible design:** Use a flexible design to allow the evaluation of a growing number of models over time

**DG6 Expansion of metadata collection:** Enable the re-evaluation of any model to expand the collected metadata

The first four goals aim to overcome the shortcomings observed with performance metadata provided by existing model zoos, described in Section 1.1, and the latter two allowing for a more flexible benchmarking system being created when using our framework. We will briefly describe the purpose of each of these goals.

#### DG1: Removing burden from user

In current model zoos, the person writing the model card is expected to provide all metadata for a model card. This need for user action limits performance metadata sharing. For example, the person writing the model card may simply not want to spend the time required to evaluate their model, or not know how to perform the evaluation. By removing the burden from the user to provide the metadata, we remove one hurdle in the sharing of this metadata.

#### DG2: High degree of performance metadata

Availability of performance metadata varies within and across existing model zoos. The model evaluation component in our framework should be designed so, that it allows the creation of a benchmarking system that can obtain performance metadata for many, if not all, available models.

#### DG3: Comparable performance results

Performance metadata provided by current model zoos is often incomparable. This stems from self-reported, not-verified results, and the possible use of different metrics by different reporters, and is compounded by problems surrounding model performance reporting in the field of machine learning as a whole (see Section 2.2). By providing comparable results, users can compare models to find one that suits their specific needs.

#### DG4: Richer metadata

Model performance reporting is often limited to a handful of values, not always providing an aspiring user of that model enough information to determine if it fits their needs. The model evaluation component should encourage obtaining of richer performance metadata than provided by current solutions, thus providing more information of the model's performance, and allowing a more fine-grained comparison between models.

#### DG5: Flexible design

Model zoos are continuously growing with new types of models, possibly performing new tasks. Benchmarking systems built at one point in time would not be able to evaluate such new types of

models. The model evaluation component should be flexible enough to accommodate the expansion of a benchmarking system as needed.

#### DG6: Expansion of metadata collection

Over time, new datasets will become available to use during the evaluation of models, and new metrics to describe a model's performance, both providing more metadata. The design of the model evaluation component should anticipate the need for re-evaluation of a model, and the fact that metadata may exist which need not be obtained multiple times.

## 4.2. Evaluation Subcomponents

In this section we give an overview of the different subcomponents that make up the model evaluation component of our framework. There are five main subcomponents: (i) a single `Executor`, which is the entry point of the pipeline, accepts a model for evaluation, and directs the evaluation process, (ii) one or more user-defined `EvaluationModules`, where each `EvaluationModule` is able to execute a specific type of models, (iii) `EvaluationTasks`, generated in the evaluation process, each defining a small task to perform in the complete evaluation process for a model, (iv) one or more user-defined `Processors`, each providing a standardized way of calculating a predefined set of metrics, and (v) one or more user-defined `Perturbations`, standardized methods of data perturbation which, when applied to datasets, create what we call "dataset variants".

Our framework provides templates for the `EvaluationTask`, `Processor`, and `Perturbation` subcomponents. By implementing these subcomponents as needed, a benchmarking system can be created to evaluate any set of models, and provide the performance metadata desired. Following is a brief description of each component.

### **Executor**

The `Executor` directs the process of evaluating a model. To evaluate a given model, it obtains the available `EvaluationModules` that can execute this model. From the configuration of each `EvaluationModule`, the `Executor` determines what performance metadata can be obtained during the evaluation of that model, and what `EvaluationTasks` to perform to do so. The evaluation process is detailed in section 4.3.

### **EvaluationModule**

How a model is to be evaluated depends on its origin (e.g. which API to use to load the model), its task (i.e. which metrics to calculate), and, in some cases, which dataset was used to train the model. This last attribute is relevant for certain tasks, as it may restrict the responses a model may be able to give. For example, the responses of a classification model are limited to one of the defined classes of the training dataset.

When using our framework, one or more `EvaluationModules` should be defined by the creator of the resulting benchmarking system. Each `EvaluationModule` is required to specify a combination of model source and task, and a dataset, and shall implement a function that per-

forms model inference (i.e. it executes the model on some input) for models from that source performing that task, utilizing that dataset. For example, one `EvaluationModule` can specify the combination `{HuggingFace, text-classification, emotion}`, meaning it can execute text-classification models originating from Hugging Face, and will always use the emotion dataset when doing so. Another `EvaluationModule` could specify `{HuggingFace, text-classification, IMDB}`, meaning it can execute the same set of models, but using a different dataset. As mentioned, the training dataset of a model may be relevant during evaluation. For this reason, each `EvaluationModule` can judge if it can execute a given model based on the model's metadata and its own specified model origin, task, and dataset.

An observant reader will note that a varied set of models may result in many combinations of origin, task, and training dataset, and thus require many `EvaluationModules` to be able to evaluate all models. This is true, and by design. When creating a benchmarking system using our framework, each `EvaluationModule` is an uncomplicated component, which, once implemented, need rarely be updated, save to expand the metadata that is to be collected. This approach allows a benchmarking system created using our framework to start with support for evaluation of a small set of models, and expand afterwards by adding additional `EvaluationModules`. Through this approach, we have achieved design goal DG5, which helps to also help achieve design goal DG2.

To evaluate a model, one or more `EvaluationTasks` are to be completed. Through configuration options, each `EvaluationModule` lets the `Executor` know which `EvaluationTask` are required to fully evaluate a model or its subset. Each `EvaluationModule` configure 1. which `Processors` to use to process the model inference output, and optionally 2. a file location if the dataset is stored on disk, 3. which dataset variants are to be evaluated as well, and 4. a specific split to use during evaluation, if not a default one. From this information, one or more `EvaluationTasks` are created. When given a model and `EvaluationTask` by the `Executor`, the `EvaluationModule` performs the model inference according to the specifications of the `EvaluationTask` and returns the output to the `Executor` for further processing.

### **EvaluationTask**

The `EvaluationTask` contains the parameters to perform model inference. Each `EvaluationTask` defines (i) which dataset to use, (ii) the location of this dataset on disk, if stored on disk, (iii) which dataset variant to use, if any, and, (iv) the current and total number of `EvaluationTasks`, for logging purposes. During evaluation of a given model, an `EvaluationModule` executes the model according to parameters given by one or more `EvaluationTasks`, with each combination of model and `EvaluationTask` resulting in unique performance metadata.

### **Processor**

A `Processor` provides a standardized method of calculating performance metrics from the output of the model inference performed by the `EvaluationModule`. While some APIs from external sources provide means to calculate metrics, our framework only uses `Processors` to guaran-



tee comparability of performance metrics also between models from different sources, and thus satisfies design goal DG3. Several `Processors` are readily provided in our framework for the calculation of metrics of classification and object-detection tasks, with each calculating not only aggregated metrics, but also for each classifiable and detectable class, thus contributing to design goal DG4.

### **Perturbation**

Our framework allows definition of `Perturbations`, standardized methods of data perturbation to create dataset variants. These variants can be used in the evaluation process to evaluate a model using a perturbed version of a dataset. This provides additional performance metadata, particularly by describing the model's performance when provided with inputs that deviate from the training dataset — something that may happen during operation of the model. Any required dataset variants required during evaluation of a model need not be provided beforehand, but are created as part of the evaluation process. By supporting this extraction of additional performance metadata during the evaluation process, this feature contributes to design goal DG4 as well.

## **4.3. Evaluation Process**

In this section we describe the process of evaluating a model. To evaluate a given model, the `Executor` performs five steps:

- 1 Module selection:** The executor retrieves the `EvaluationModules` that can execute the given model
  
- 2 Task creation:** Creating `EvaluationTasks` based on the configuration of the retrieved `EvaluationModules`
  
- 3 Task filtering:** the `EvaluationTasks` that would not provide new metadata based on metadata present in storage are ignored
  
- 4 Model execution:** Executing the remaining `EvaluationTasks`
  
- 5 Results processing:** Processing and storing the results using `Processors` defined by the `EvaluationModule`

Figure 4.1 visualizes these steps. The combination of the first and third step allows a continuous evaluation cycle, where, at intervals, all models can be pushed through the evaluation pipeline for evaluation, regardless if they actually can be evaluated, or have been evaluated already. This means that, once a new model is shared, it will automatically be picked up in the next iteration of the cycle, and performance metadata is extracted as required. It also means that, if the performance metadata that should be extracted from a model changes, in a next iteration of the cycle this newly desired metadata, and only this metadata, is automatically extracted. Neither initial evaluation nor re-evaluation require actions from the model creator, thus completing design goal DG1. Next,

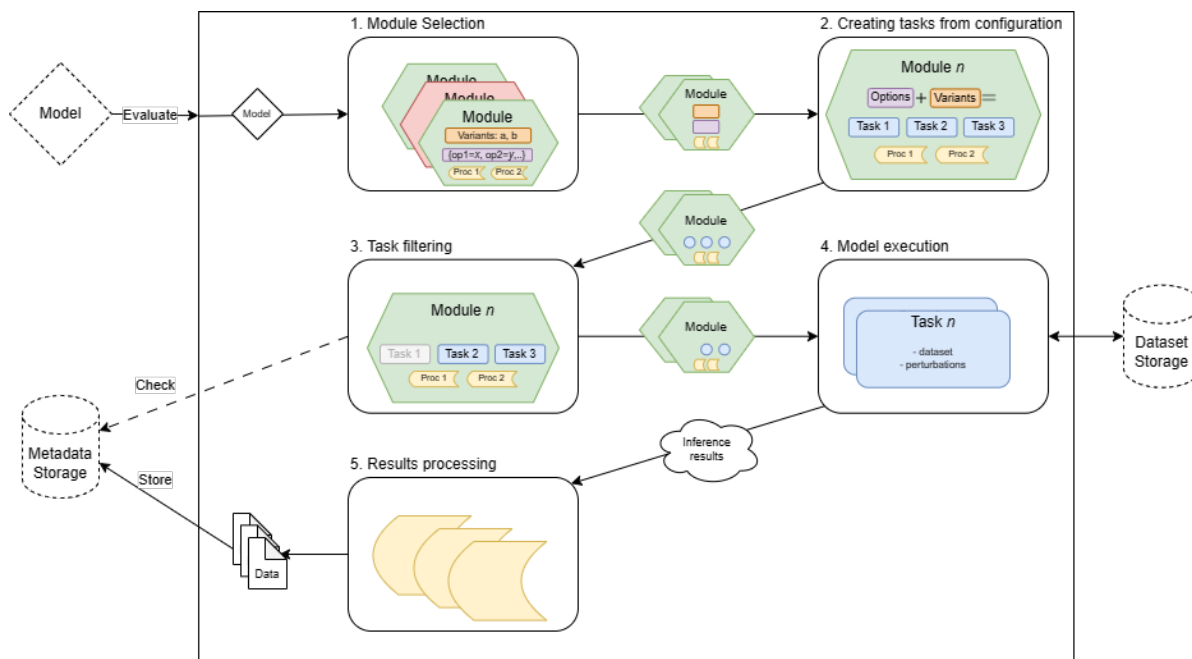


Figure 4.1: The process when evaluating a single model. Five steps are defined. 1. from all available `EvaluationModules` the ones able to evaluate the given model are selected. Then, for each `EvaluationModule`, 2. from the base step and configuration of the runner `EvaluationModule`, additional steps are created, 3. the `EvaluationTasks` that do not provide new metadata for this model are filtered out, 4. each remaining `EvaluationTask` is executed by the `EvaluationModule`, 5. the results are processed by each `Processor` defined by the `EvaluationModule` and stored

we describe each step.

## 1. Module selection

The first step the `Executor` performs is to select the `EvaluationModules` which can be used to execute the given model. If the model does not fall within the subset defined by any of the `EvaluationModules` present, this means it is not supported yet, and the evaluation process ends here. The subsequent steps are performed for each `EvaluationModule`.

## 2. Task creation

In the second step the `Executor` creates all `EvaluationTasks` to complete to evaluate the model with the `EvaluationModule`. It does so by first creating a base `EvaluationTask` to evaluate with the dataset defined by the `EvaluationModule`. From this base `EvaluationTask`, additional `EvaluationTasks` are created based on the configuration of the `EvaluationModule`, such as dataset variants to use during evaluation. Each `EvaluationTask` provides a unique set of metadata for each model.

## 3. Task filtering

Third, the `Executor` filters out the `EvaluationTasks` that would not result in new performance metadata. After filtering, if the model has been evaluated before, only new `EvaluationTasks` will be used during this evaluation, and no duplicate metadata will be collected. If no steps remain,

the evaluation process ends here. This step allows indiscriminate evaluation

#### **4. Model execution**

In the fourth step, each of the `EvaluationTasks` are executed. If the step specifies a dataset variant and source location on disk, the `Executor` ensures the dataset variant exists. If not, the variant is created in this step, and stored so that it can be re-used during the evaluation of another model in the future. To execute the `EvaluationTask`, it and the model are passed to the `EvaluationModule` from which the step was created, which performs model inference based on the parameters of the `EvaluationTask`. After the inference, the `EvaluationModule` returns information regarding the execution speed and the results of the inference itself.

#### **5. Results processing**

In the final step, after execution of each `EvaluationTask`, the results are processed by each `Processor` the `EvaluationModule` defined in its configuration, which in turn return standardized results. These processed results are then stored and accessible, for example through a user interface.



# 5

## Analysis of collected performance metadata

In this chapter we describe the results achieved by our implementation of a benchmarking system, built using our framework for an evaluation pipeline. In Section 5.1 we look at the availability and diversity of the performance metadata generated by our benchmarking system. We do this specifically for the subset of models our benchmarking system should be able to evaluate at time of writing this thesis. The user interface to view the obtained performance metadata is accessible at <https://www.metadatazoo.io>.

Some metadata, particularly inference speed, depend on hardware and software used during evaluation. For reproducibility of our results, we outline the hardware and software used in our benchmarking system in Tables 5.1a and 5.1b.

### 5.1. Performance metadata availability

In this section we compare the performance metadata provided by our benchmarking system to that provided by existing model zoos. We compare this metadata only for models that have already been passed through our benchmarking system. The `EvaluationModules` (see Section 4.2) in our system provides support to evaluate Hugging Face and FiftyOne models that perform one of three tasks, and were trained on one of thirteen datasets, resulting in a total of 1215 models evaluated. Table 5.2 shows the distribution of these models per selected task and training dataset, and per source the models were collected from.

Our benchmarking system was not able to evaluate all of these models. For 229 models (18.8%) evaluation failed, for one of three reasons: (i) incompatibility between libraries used in our benchmarking system and those used when creating the model, (ii) insufficient metadata available in the external source to load the model with their API, or (iii) use of unknown classification labels,

Hardware	
CPU	AMD EPYC 7413
GPU	NVidia A40
Memory	512GB
Platform	Version
Ubuntu	20.04.1 LTS
Docker	20.10.12
Software	Version
Language	Python 3.8.10
Database	MongoDB 4.4

(a) Execution environment of our benchmarking system

Dependencies	Version
pymongo	3.12.3
tensorflow	2.9.1
torch	1.10.2+cu113
torchvision	0.11.3+cu113
torchaudio	0.10.2+cu113
fiftyone	0.16.5
huggingface-hub	0.11.1
datasets	2.8.0
transformers	4.21.2
evaluate	0.2.2
pycocotools	2.0.6
Pillow	9.0.4

(b) Python dependencies used to implement our benchmarking system

Table 5.1: Hardware and version of software and important dependencies used in our benchmarking system

for example by using different capitalization (Sadness vs. sadness), using numbered labeling instead of human-readable names (LABEL\_0, LABEL\_1, ...vs. airplane, automobile, ...), using a differently worded label (Science/Technology vs. Sci/Tech, 5 star vs. 5), using unknown labels (e.g. Disgust when trained on the Emotion dataset), or using different language (悲哀 vs. sadness). For models that failed because of the third reason, inference results could not appropriately be judged to be correct or incorrect, and no metrics could be calculated. In some EvaluationModules the dataset metadata was manually extended to include these different labels in the label-to-class-id mapping, if this error was discovered and the differing labels could be identified, so that some models could be evaluated after all.

For the models that were evaluated, the collected performance metadata was consistent between models that perform the same task and were evaluated on the same dataset. This holds true between models from different external model zoos, due to the re-use of Processors by each EvaluationModule. Table 5.3 details the metrics collected for each task. As all evaluated object detection models were trained on the COCO dataset, the COCO metrics were calculated during evaluation for all of these models. These metrics were calculated for the dataset as a whole, but also for each individual class of objects to detect. For the two classification tasks, different metrics are used for the dataset average and per-class results, with the difference lying in true positives, false positives, and false negatives calculated for each classifiable class, and merely the number of correct and incorrect classifications recorded for the whole dataset. This is done because false positives/negatives may differ for each individual class, while for the whole dataset it counts as a single incorrect classification. For example, a classification model may classify a picture of a cat as a dog, counting as one false negative for the "cat" class, and one false positive for the "dog" class, but one "incorrect" classification for the dataset as a whole.

Task	Training Dataset	Evaluated Models	HF Models	FO Models
Object Detection	COCO	74	43	31
Image Classification	ImageNet	199	173	26
Image Classification	Beans	49	49	0
Image Classification	CIFAR-10	17	17	0
Image Classification	CIFAR-100	6	6	0
Image Classification	Food101	19	19	0
Image Classification	MNIST	7	7	0
Text Classification	Emotion	468	468	0
Text Classification	Ag-News	12	12	0
Text Classification	Banking77	14	14	0
Text Classification	IMDB	311	311	0
Text Classification	Rotten Tomatoes	14	14	0
Text Classification	Yelp Review	26	26	0
All tasks	All datasets	1215	1158	57

Table 5.2: Number of models evaluated by our benchmarking system by task and training dataset, showing the total number of models and for each of the external sources — Hugging Face (HF) and FiftyOne (FO).

Task	Collected metrics
Object Detection	COCO metrics <sup>a</sup>
Image Classification	True positive, False Positive, False Negative (per class) Correct, Incorrect (dataset) Precision, Recall, F1-Score, average and per class
Text Classification	True positive, False Positive, False Negative (per class) Correct, Incorrect (dataset average) Precision, Recall, F1-Score (dataset average and per class)

Table 5.3: The metrics collected during evaluation of a model of the specified tasks. As for each of these tasks the model is tasked with finding objects of a specific class, the collected metrics are calculated for each individual class, as well as the average over the dataset.

<sup>a</sup><https://cocodataset.org/#detection-eval>

Task	Training Dataset	Data points collected per evaluation
Object Detection	COCO	972 (FO models), 5832 (HF models)
Image Classification	ImageNet	7006 (FO models), 49042 (HF models)
Image Classification	Beans	27
Image Classification	CIFAR-10	76
Image Classification	CIFAR-100	706
Image Classification	Food101	713
Image Classification	MNIST	76
Text Classification	Emotion	48
Text Classification	Ag-News	41
Text Classification	Banking77	545
Text Classification	IMDB	40
Text Classification	Rotten Tomatoes	40
Text Classification	Yelp Review	41

Table 5.4: Number of data points collected during evaluation of a model of the given task and trained on the given dataset. For some combinations more data points are collected for Hugging Face (HF) models, as their evaluation include the use of additional datasets and dataset variants not used in evaluation of FiftyOne (FO) models

Table 5.4 shows how many performance data points were collected for each model, divided by task and training dataset. The numbers follow from the presented metrics collected. For example, for models evaluated with the CIFAR-10 dataset, 7 values are calculated for each of the 10 classes, and 6 aggregated values are calculated for the dataset average:  $6 + 7 * 10 = 76$  data points. For IMDB and rotten\_tomatoes models more data points are collected, as models trained on either of those datasets are evaluated using both, with evaluation using either dataset resulting in 20 data points. For COCO and ImageNet models, Hugging Face models are evaluated utilizing 5 dataset variants, resulting in more data points, and an additional dataset (ImageNet-sketch) was used to evaluate models trained with ImageNet.

Figures 5.1a and 5.1b show the fraction of evaluated models for which more than one dataset, or dataset variants were used during the evaluation. The use of additional datasets or variants increases the available performance metadata after evaluation.

### 5.1.1. FiftyOne

In the case of FiftyOne evaluation (see Figure 5.2b) failed for fourteen of the 57 models (24,5%). In all cases, this was due to a mismatch in libraries used by our benchmarking system (TensorFlow 2) and in the creation of the models (TensorFlow 1), meaning these models could not be executed. This could be remedied in future applications by setting up an alternative environment with different dependencies, to run a second instance of our benchmarking system.

FiftyOne does not provide much metadata for each model. Besides details such as the name, external source, size, tags, they state required packages and if the model has CPU or GPU support. No performance metadata are provided. This means that, for each of these models all the metadata



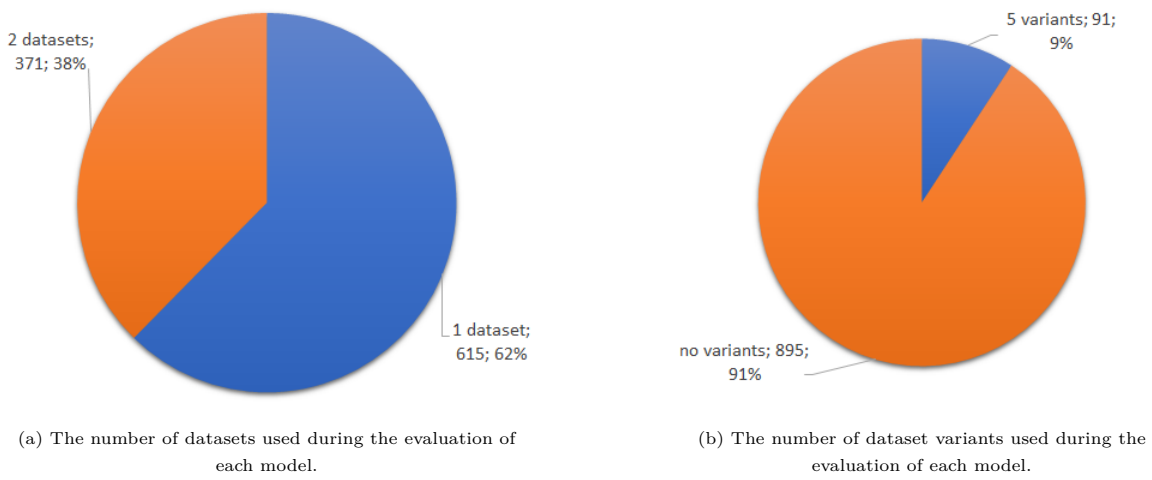


Figure 5.1: Fractions of evaluated models that use additional datasets or dataset variants, each resulting in additional performance metadata

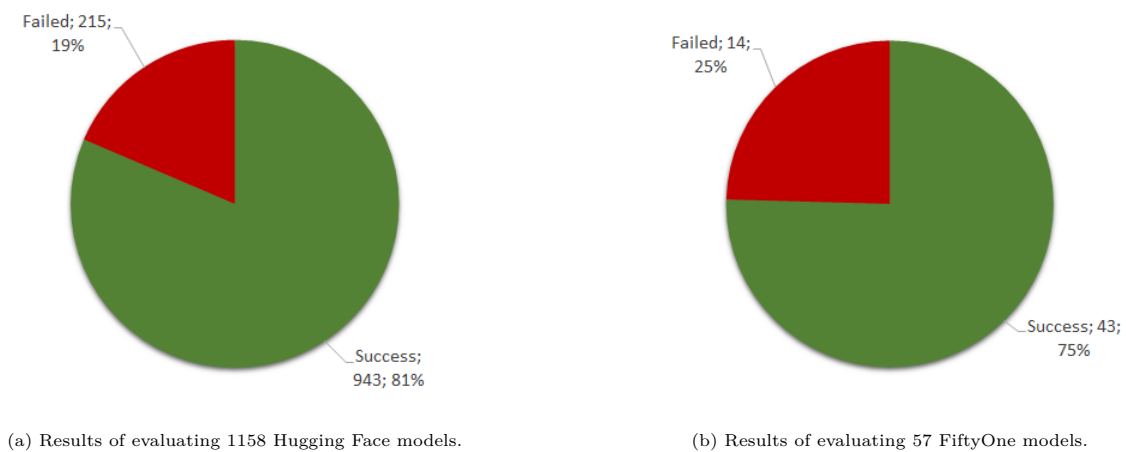


Figure 5.2: Fraction of evaluated models that succeeded and failed, per external source

collected by our benchmarking system is new to the model card of the model. Currently, for roughly half of the models in the FiftyOne model zoo, our benchmarking system expands the model card with new performance results, at no additional effort to the user. It is our believe that, by adding additional `EvaluationModules` and setting up alternative execution environments, for all FiftyOne models the model cards can be enriched with performance values through our benchmarking system.

### 5.1.2. Hugging Face

For Hugging Face models, some model cards do report performance data. This is done in textual form, but Hugging Face has also defined a structural way to present evaluation data<sup>1</sup>, which is shown in a separate section of the webpage of the model. However, this is not always used when reporting performance results. We attempted to categorize the manner performance results are reported on Hugging Face, for the 1158 models we also ran through our benchmarking system. Figure 5.3a shows the distribution of availability of performance metrics in either the description of the model card, or using Hugging Face’s format to report performance numbers. For 257 models (22,1%) of the model cards, performance results were not given. In another 98 cases (8,4%), they were given solely in the description of the model. For 10 model cards (0,9%), they were given only using Hugging Face’s format, and thus displayed separately in the model cards, and for the majority of models, 793 models (68,4%), they were given both in the model’s description as well as in Hugging Face’s model card format. Performance results that are shown separately are naturally preferable, as then the user does not need to scan the description for them.

Besides how often performance values are given, it is also relevant what exactly is given. Figure 5.3b shows for model cards that provide performance results in Hugging Face’s format, how many are self-reported, and how many are verified performance values. A large majority of performance values are self-reported, meaning the creator of the model provides them in the model card, and have not been verified to be accurate. Figure 5.3c shows for these same model cards how many values were given in total. Here it is visible that, even though a handful of model cards provide a few more values, for most models the performance is reported by just a single or two values. While this provides some value, users wanting to know a model’s performance on a specific metric, or under certain circumstances cannot get the answers they seek with such limited information. Of note is the one model who provides 23 values. However, this model (`morenolq/distilbert-base-cased-emotion`<sup>2</sup>) provides duplicate values for the same metric, even including a contradiction at that.

When we compare this to the results to be obtained by the benchmarking system built using our framework, there are two main take-aways. First, many Hugging Face models already included some performance results in their model cards. Using our benchmarking system we have added performance number to just 42 models, or 3,6% of the evaluated models. If we do not consider performance values that were reported in textual form only, this number still just grows to 140

<sup>1</sup><https://github.com/huggingface/hub-docs/blob/main/modelcard.md?plain=1>

<sup>2</sup><https://huggingface.co/morenolq/distilbert-base-cased-emotion>

models, or 12,1%. This number could be increased by updating our benchmarking system and the environment it runs in, to evaluate some of the models that could not be evaluated before, but for other it is not within our control. Nevertheless, a large share of Hugging Face models already include performance numbers.

It should be noted, that during inspection of the models themselves, many models were found of the same name. This may occur, as names are often a combination of a pre-trained model and the dataset it was finetuned on. What we consider less likely though, is a natural occurrence of 364 models names `distilbert-base-uncased-finetuned-emotion` and 146 models named `finetuning-sentiment-model-3000-samples`. Most — if not all — models of this name provide the same information in their model card and leave the same parts empty, as if using the same template. We find it plausible that these models were shared by students of an (online) course, and including performance values was part of the instructions. While we do not discount these models, they do significantly increase the share of models that include performance values in their model cards, both for Hugging Face and our benchmarking system, and a different selection of models to support in our benchmarking system could paint a significantly different picture.

The second main take-away, is that, even though many Hugging Face models provide some performance values, the performance metadata can be significantly enriched through the use of our benchmarking system. Where the vast majority of models provide just one or two values, our benchmarking system provides dozens to thousands of values. Some of these values come from evaluation with multiple datasets or dataset variants, which would be a new inclusion on Hugging Face, where evaluation values are based on original datasets only.

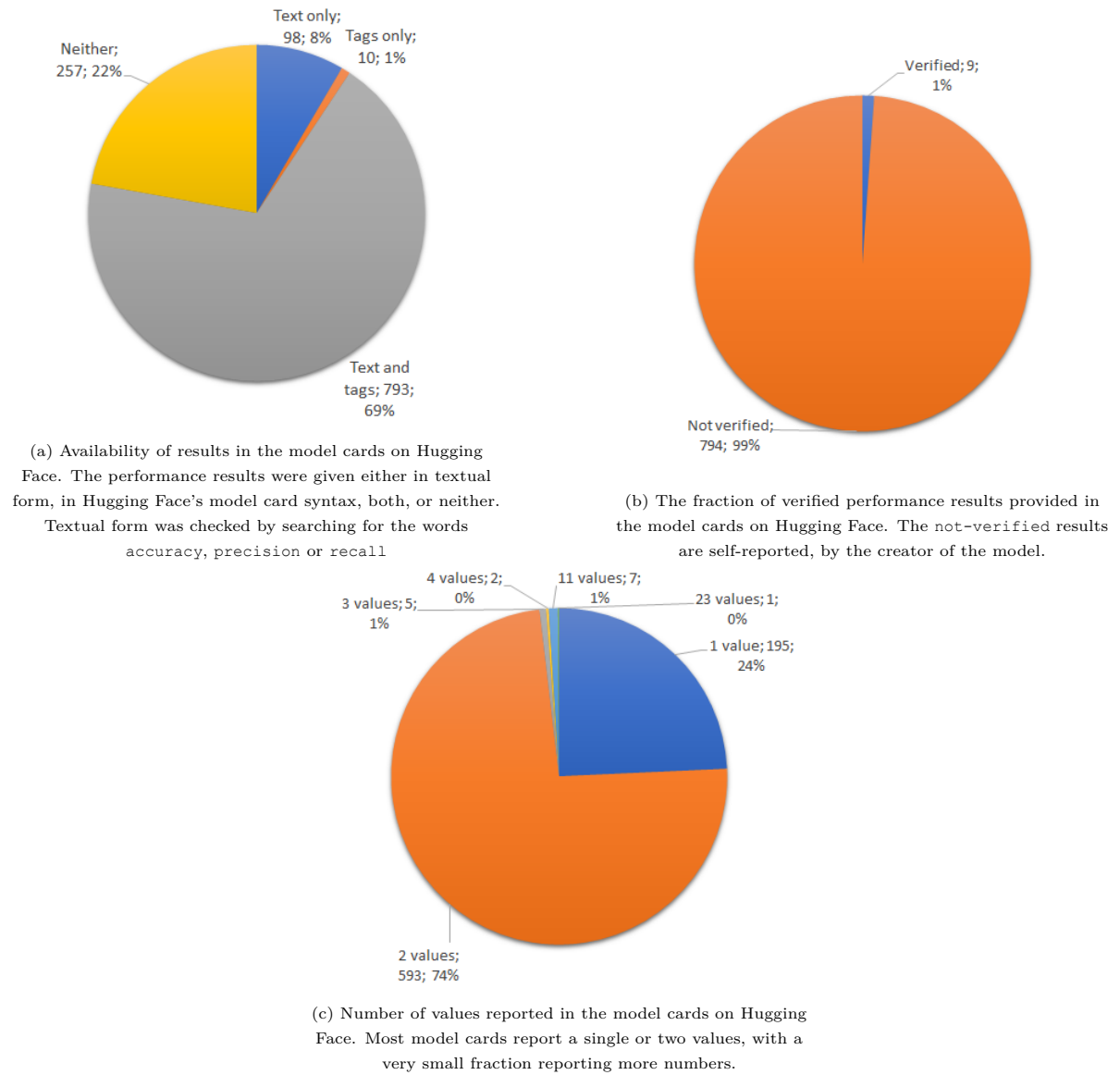


Figure 5.3: Statistics of the performance results provided by Hugging Face for the 1158 models also evaluated by our benchmarking system

# 6

## User Interface

This chapter will describe and show the interactive user interface (UI) created to visualize the performance metadata obtained by our implementation of a benchmarking system (also see Chapter 5). The main goal is to visualize the model performance metadata in a way that enables the use-cases as described in Section 1.3, though it does display other metadata of both models and datasets. The UI's web pages can be divided into two categories. The ones showing the information related to datasets are described in section 6.1, and those showing all information of the models in the zoo are described in section 6.2. The user interface is accessible at <https://www.metadatatoo.io/>.

### 6.1. Datasets

This section describes and shows the visualization of dataset metadata obtained by our benchmarking system. Though this aspect is not the primary focus of this thesis, the dataset used to train a model can provide a user with extra information about the model itself, such as biases in the training data and thus the model's performance, and examples of data used to train the model. The visualizations shown in this section are of the metadata of the COCO dataset<sup>1</sup>.

#### 6.1.1. Dataset information

For each dataset in our model zoo a page exists showing all its available information. As an example we look at the COCO dataset. The dataset page can be divided into four sections. At the top of the page we show the basic information for the dataset. For the COCO dataset, this is a short description, its logo. There is a link to the homepage of the dataset in case the user wants to get more information. Below that, two buttons are given allowing the user to view models trained using this dataset, or to view a scatterplot to compare all models already evaluated using

---

<sup>1</sup><https://www.metadatatoo.io/datasets/6325c7632aa8981c9d3b411f>

this dataset. Then, a reference, if available, can be shown should the user wish to reference the paper in which the dataset was introduced. If models trained on this dataset can be evaluated using other datasets as well, these datasets are listed as "compatible datasets". Finally, a list of perturbations that may be used during evaluation with this dataset are shown. See Figure 6.1 for a screenshot of COCO's information overview.

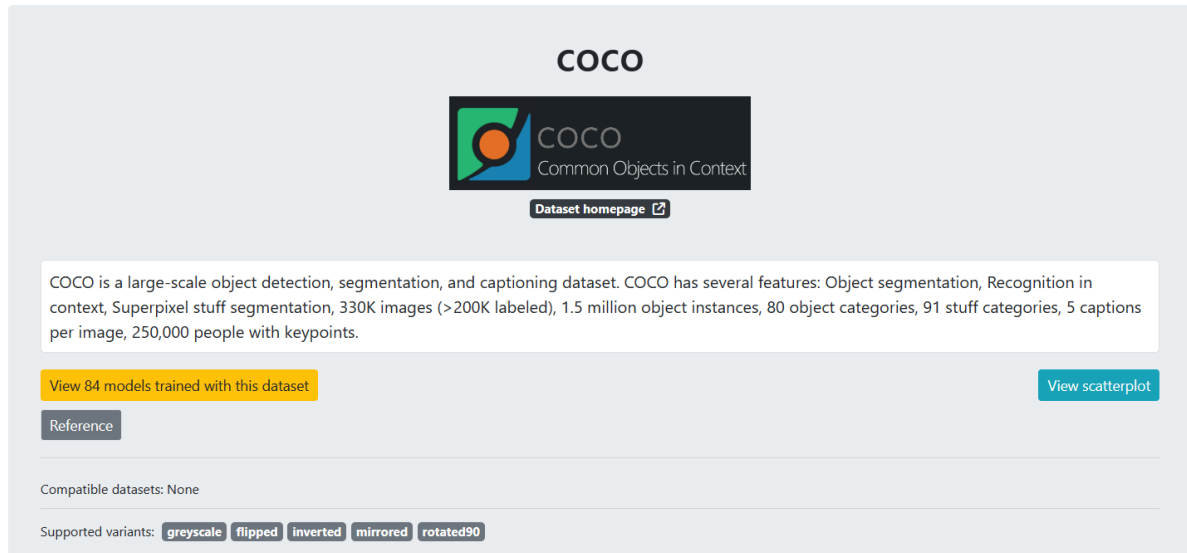


Figure 6.1: The basic information of the COCO dataset.

If possible, examples of objects in the dataset are shown. If applicable, such as for object detection datasets like COCO, we show examples of the dataset's classes, with up to 20 examples of each class. Figure 6.2 shows a screenshot of the classes of the COCO dataset. For each of its 80 classes, it shows the class id with the label. For each class, ten example images are shown containing at least one object of that class. Though for this dataset image examples are present for each class, other datasets might have just a single image, such as a collage showing examples of multiple classes, or text examples instead of images.

Further down the page, the user is presented with a summary of the models which have been evaluated using this dataset. Figure 6.3 shows this section for the COCO dataset. This summary is shown for each possible variant of the dataset, as the number may differ per variant. For each variant the total number of evaluated models is shown, and how many of these evaluated models were evaluated on this dataset. These numbers may be different, if model trained on a compatible dataset were evaluated using the shown dataset as well. For our COCO example, this is not the case. For each variant, the user may open a dialog showing all models evaluated using this variant. Here the user may enter a search term to search for a specific model, and directly view and compare the performance results of any evaluated model. Figure 6.4 shows this dialog for the greyscale variant of the COCO dataset.

Most datasets are subdivided in multiple subsets called splits. This is to reserve a subset of the data to, for example, test the model after training it on a different subset of the dataset. We show the different splits available for the dataset, and a few statistics for each of them. For our COCO

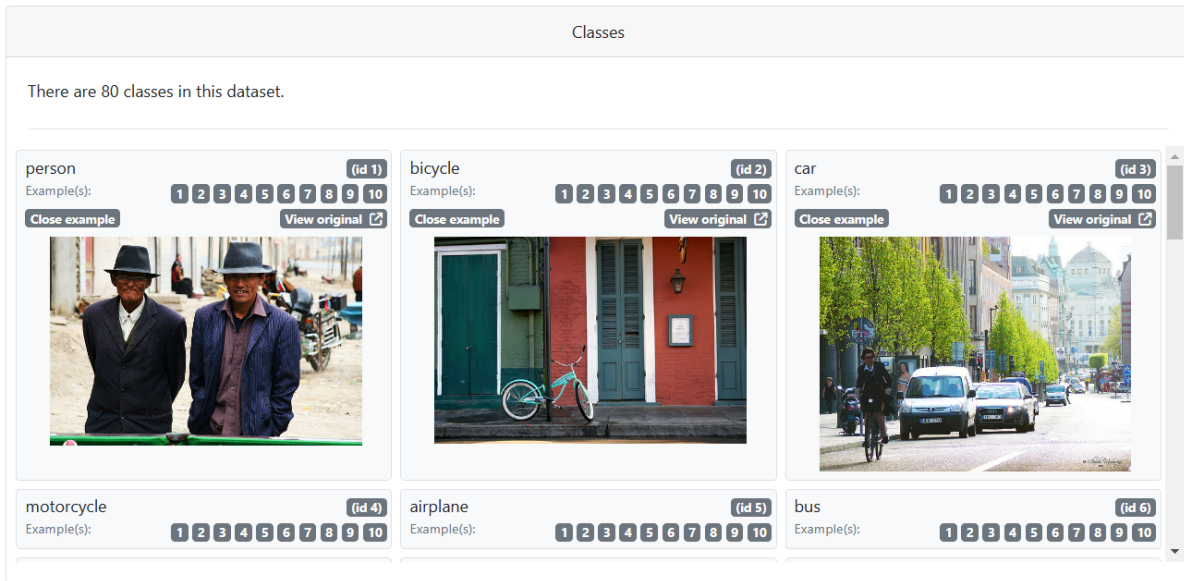


Figure 6.2: The available classes for the COCO dataset. Shown are examples for the first three classes

Evaluated models		
Supported variants: <b>greyscale</b> <b>flipped</b> <b>inverted</b> <b>mirrored</b> <b>rotated90</b>		
<b>Original Dataset</b> <a href="#">View evaluated models</a> <hr/> Models evaluated on dataset: <b>27</b> <hr/> Trained on this dataset: <b>27</b>	<b>Greyscale variant</b> <a href="#">View evaluated models</a> <hr/> Models evaluated with variant: <b>24</b> <hr/> Trained on this dataset: <b>24</b>	<b>Flipped variant</b> <a href="#">View evaluated models</a> <hr/> Models evaluated with variant: <b>24</b> <hr/> Trained on this dataset: <b>24</b>
<b>Inverted variant</b> <a href="#">View evaluated models</a> <hr/> Models evaluated with variant: <b>24</b> <hr/> Trained on this dataset: <b>24</b>	<b>Mirrored variant</b> <a href="#">View evaluated models</a> <hr/> Models evaluated with variant: <b>24</b> <hr/> Trained on this dataset: <b>24</b>	<b>Rotated90 variant</b> <a href="#">View evaluated models</a> <hr/> Models evaluated with variant: <b>24</b> <hr/> Trained on this dataset: <b>24</b>

Figure 6.3: A summary of the models evaluated with the COCO dataset



Models evaluated with the greyscale variant

Search:

centernet-hg104-1024-coco-tf2	View	Compare
centernet-hg104-512-coco-tf2	View	Compare
centernet-resnet101-v1-fpn-512-coco-tf2	View	Compare
centernet-resnet50-v1-fpn-512-coco-tf2	View	Compare
centernet-resnet50-v2-512-coco-tf2	View	Compare
efficientdet-d0-512-coco-tf2	View	Compare
efficientdet-d1-640-coco-tf2	View	Compare
efficientdet-d2-768-coco-tf2	View	Compare
efficientdet-d3-896-coco-tf2	View	Compare
efficientdet-d4-1024-coco-tf2	View	Compare

Previous 1 2 3 Next

Figure 6.4: An overview of models evaluated with the greyscale variant of the COCO dataset.



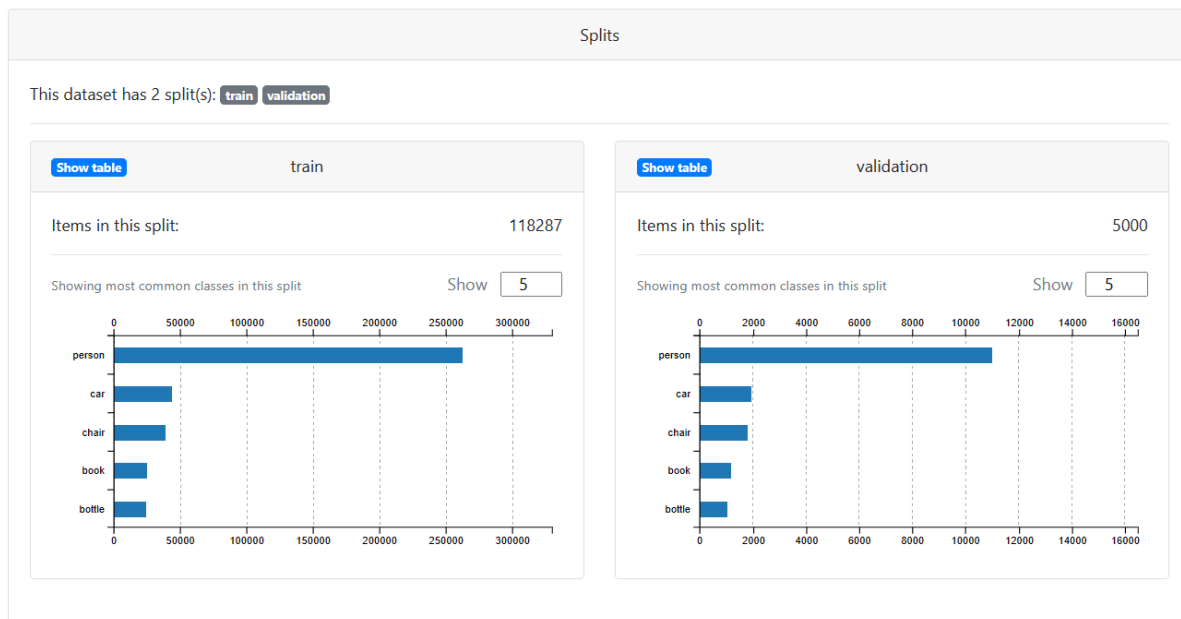


Figure 6.5: The splits of the COCO dataset.

dataset there are 2 splits: `train` and `validation`, as shown in Figure 6.5. The total number of objects in each split is given, 118287 and 5000 respectively. For the COCO dataset we also know the division of the dataset objects per class of the dataset, so we can display this in a diagram as well. This diagram is limited to just the five most common classes as there might be many, depending on the dataset, but can be extended by the user to show more.

### 6.1.2. Scatterplot

Once many models have been evaluated using the same dataset, one might be interested to compare all of these models directly to gain a high-level understanding of the performance of different models. For this purpose our user interface includes a scatterplot, in which a user can compare all models evaluated on a specific dataset, and the user can specify the attributes they want to compare, such as any of the performance metrics collected during evaluation, runtime, or model size. If many models have been evaluated, certain areas of the plot can become too crowded to easily make out the minor differences between models. In this case, the user can zoom in and pan around the plot to get a better view. Clicking a dot in the plot brings the user to the evaluation results of the model it represents. Figure 6.6 shows all models for the COCO dataset, with the models' score for the `COCO_AP_50_95_ALL`<sup>2</sup> metric size compared to the speed of the evaluation.

If so desired, the values presented in the scatterplot can also be viewed in tabular form, with options to sort the models on either of the selected attributes, and a field to search for specific models. Figure 6.7 shows the table displaying the same information shown in the scatterplot shown in Figure 6.6.

<sup>2</sup><https://cocodataset.org/#detection-eval>

Show models for

Show  on x-axis, against  on y-axis, for class

Plotting 27 models, COCO\_AP\_50\_95\_ALL versus RUNTIME, for class Bicycle. Scroll to zoom.  
Click a value in the plot to view the model's results.

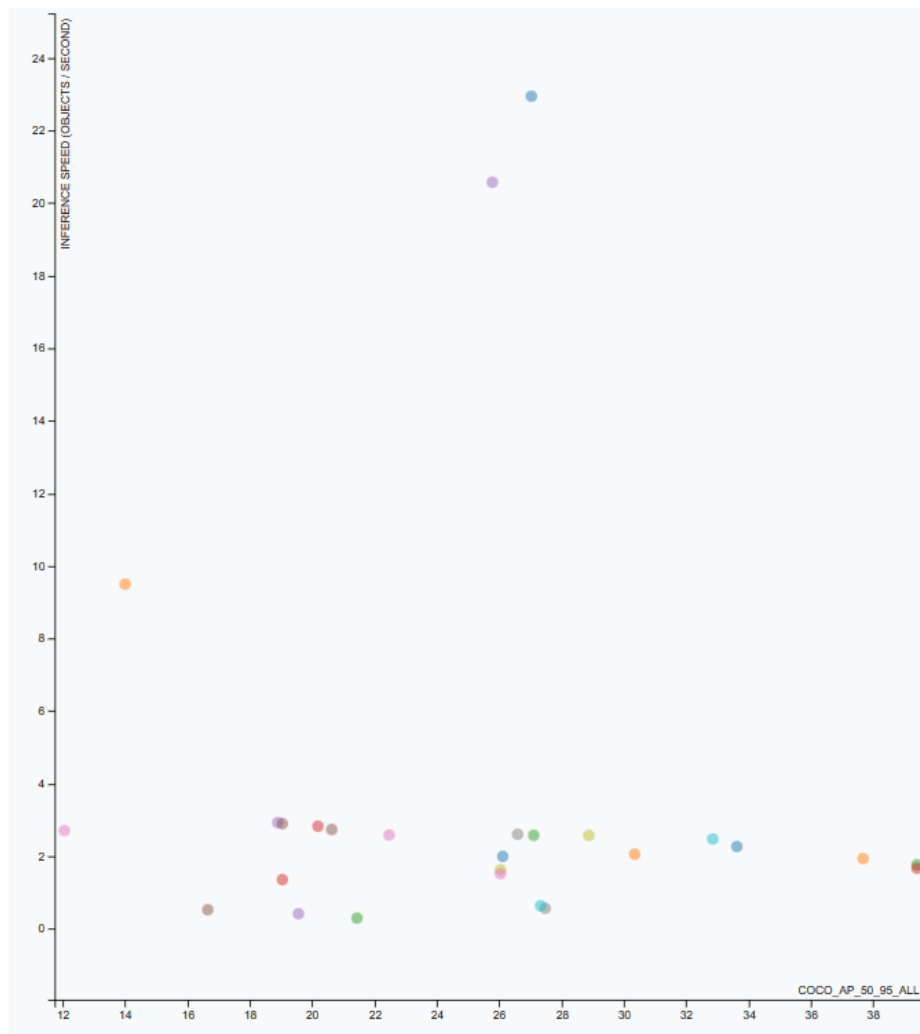


Figure 6.6: A scatterplot comparing all models evaluated on the COCO dataset. The plot shows the model's performance on the COCO\_AP\_50\_95\_ALL metric versus total duration of the evaluation.

Show models for object-detection ▼

Show COCO\_AP\_50\_95\_ALL ▼ on x-axis, against RUNTIME ▼ on y-axis, for class bicycle ▼

[View](#)

[View plot](#)

---

Show 10 entries Search:

Model	COCO_AP_50_95_ALL	RUNTIME
centernet-hg104-1024-coco-tf2	30,33	2,06
centernet-hg104-512-coco-tf2	27,10	2,58
centernet-resnet101-v1-fpn-512-coco-tf2	20,16	2,83
centernet-resnet50-v1-fpn-512-coco-tf2	18,90	2,93
centernet-resnet50-v2-512-coco-tf2	19,05	2,90
efficientdet-d0-512-coco-tf2	22,47	2,59
efficientdet-d1-640-coco-tf2	26,58	2,61
efficientdet-d2-768-coco-tf2	28,87	2,58
efficientdet-d3-896-coco-tf2	32,83	2,48
efficientdet-d4-1024-coco-tf2	33,62	2,27

Showing 1 to 10 of 27 entries

[Previous](#)
[1](#)
[2](#)
[3](#)
[Next](#)

Figure 6.7: A table to show the same information as the scatterplot shown in Figure 6.6

The screenshot shows a web interface for browsing models. On the left, there are several filter sections: 'Name contains:' with a text input; 'Model came from:' with a dropdown menu set to 'fiftyone'; 'Task is:' with a dropdown menu set to 'Any'; 'Trained on dataset:' with a dropdown menu set to 'Any'; 'Evaluated on dataset:' with a dropdown menu set to 'Any'; 'Metrics available:' with a dropdown menu set to 'Any'; and 'Sort:' with 'Name' and 'Asc' selected. A 'Submit' button is at the bottom of the filters. At the top right, there are navigation buttons: 'Go home', '← previous', 'Page 1 / 6', and 'next →'. The main content is a table with the following columns: Model, Origin, Task, Dataset, and Metrics available. The table lists several models, including alexnet-imagenet-torch, centernet-hg104-1024-coco-tf2, centernet-hg104-512-coco-tf2, centernet-mobilenet-v2-fpn-512-coco-tf2, centernet-resnet101-v1-fpn-512-coco-tf2, centernet-resnet50-v1-fpn-512-coco-tf2, centernet-resnet50-v2-512-coco-tf2, deeplabv3-cityscapes-1f, deeplabv3-mnv2-cityscapes-1f, and deeplabv3-resnet101-coco-torch. Each row shows the model name, its origin (FiftyOne), the task it performs (e.g., image-segmentation, object-detection, unknown), the dataset it was trained on (e.g., imagenet, coco), and whether performance metrics are available (Yes/No) along with 'View' and 'Compare' buttons.

Model	Origin	Task	Dataset	Metrics available
alexnet-imagenet-torch	FiftyOne	image-segmentation	imagenet View	No
centernet-hg104-1024-coco-tf2	FiftyOne	object-detection	coco View	Yes View Compare
centernet-hg104-512-coco-tf2	FiftyOne	object-detection	coco View	Yes View Compare
centernet-mobilenet-v2-fpn-512-coco-tf2	FiftyOne	unknown		No
centernet-resnet101-v1-fpn-512-coco-tf2	FiftyOne	object-detection	coco View	Yes View Compare
centernet-resnet50-v1-fpn-512-coco-tf2	FiftyOne	object-detection	coco View	Yes View Compare
centernet-resnet50-v2-512-coco-tf2	FiftyOne	object-detection	coco View	Yes View Compare
deeplabv3-cityscapes-1f	FiftyOne	image-segmentation		No
deeplabv3-mnv2-cityscapes-1f	FiftyOne	image-segmentation		No
deeplabv3-resnet101-coco-torch	FiftyOne	image-segmentation	coco View	No

Figure 6.8: The view to browse models, with several options to filter and sort this list.

## 6.2. Models

This section will describe and show the visualizations used to display the metadata for models. The visualizations shown in these sections are made possible by the ability of our benchmarking system to extract rich performance metadata during model evaluation. This rich metadata allows for several visualizations, each highlighting a different aspect of the model's performance. The visualizations can display a model's performance after evaluation, including task-specific visualizations such as a confusion matrix and inspection of errors made during evaluation. They also allow inspection of a model's behavior when presented with perturbed data, by comparing performance during evaluation of a dataset variant with performance of the original dataset. Finally, two or more models can be compared head-to-head, in which the performance of multiple models is compared directly. Each of these visualizations is interactive, allowing the user to select data to view, remove metrics they are not interested in, and sort the presented data in a way that interests them.

### 6.2.1. Browsing models

First, the user is presented with a list of all models, shown in Figure 6.8. Here, the user can browse and search for a specific model. The shown list of models highlights for each model the origin of the model, the task it performs, the dataset it was trained on, and whether performance metadata is available to view, with links to directly view and compare. Several filtering options are available to search for a specific model, such as filtering by model origin, task, the dataset the model was trained and/or evaluated, and whether performance metadata is available, and options are given to sort the resulting list of models.



Figure 6.9: The basic info for the facebook/regnet-x-120 model

### 6.2.2. Model information

For each model the user can view its information and any available performance metadata through a few interactive visualizations. First showing on the page for each model is its basic info. Figure 6.9 shows a screenshot of the basic info for the facebook/regnet-x-120<sup>3</sup> model. First, there is the name of the model, and a link to the external source if there is one; Hugging Face for this model. For the shown model, there are links back to the list of all models, with a filter to either the origin, task, dataset, or availability of metrics pre-applied. If there are evaluation results — for this model for the ImageNet and ImageNet-Sketch datasets — the user can also view or compare the performance results. Below this, the scraped tags will also be shown here, if there are any. Finally, the model's size is shown, and a reference which can be used in scientific papers, if it exists. For Hugging Face models such as facebook/regnet-x-120 there are often files available for multiple frameworks. In our example the model can be used with either the PyTorch or TensorFlow, thus showing the size for either configuration.

Below the basic info of the model is some information about each of the datasets used for training or evaluation of the model. Figure 6.10 shows here the information of the ImageNet dataset for the facebook/regnet-x-120 model. If the model has been evaluated on this dataset, a summary of this can be made visible to show a summary and the total duration of the evaluation. This diagram, like most throughout our model zoo, are interactive. The user can disable and enable the metrics they want to see, and hovering the mouse cursor over the metrics shows a popup presenting the data in a concise way. If the model has been evaluated on other datasets, they are shown after the training dataset, using the same summary of performance.

Finally, for each model any available hyperparameters, and any textual description from the original source is shown, with a link to the external source. Figures 6.12 and 6.13 show the hyperparameters and a portion of this description for the facebook/regnet-x-120 model.

<sup>3</sup><https://www.metadatatoo.io/models/63a0b0d515a79b5cfb17a0c5>

Trained with
Evaluated with
The IMAGENET dataset
View imagenet dataset

The ImageNet project is a large visual database designed for use in visual object recognition software research. More than 14 million images have been hand-annotated by the project to indicate what objects are pictured and in at least one million of the images, bounding boxes are also provided. ImageNet contains more than 20,000 categories, with a typical category, such as "balloon" or "strawberry", consisting of several hundred images. The database of annotations of third-party image URLs is freely available directly from ImageNet, though the actual images are not owned by ImageNet.

Show evaluation summary
View evaluation results
Compare results

Figure 6.10: The dataset information for the facebook/regnet-x-120 model.

## Evaluation summary ✕

The diagrams below shows the overall performance of this model on this dataset. You can view the evaluation results for more in-depth results, or compare it with other models.

Total objects in dataset	Total time in seconds	Time per object	Objects per second
100000	3720.17488	0.0744	13.44023

Metric	Value
RECALL	0.795
F1SCORE	0.795
PRECISION	0.795

View evaluation results
Compare results

Figure 6.11: A summary of the evaluation results for the evaluation of the facebook/regnet-x-120 model with the ImageNet dataset. The mouse cursor is hovered over the diagram, opening a popup with the information of the group average.

Hyperparameters	
Parameter	Value
architectures	[RegNetForImageClassification]
depths	[2, 5, 11, 1]
downsample_in_first_stage	true
embedding_size	32
groups_width	112
hidden_act	relu
hidden_sizes	[224, 448, 896, 2240]
layer_type	x
model_type	regnet
num_channels	3
torch_dtype	float32
transformers_version	4.18.0.dev0

Figure 6.12: Display of the hyperparameters for the facebook/regnet-x-120 model.

Original Readme [View on huggingface](#)

### RegNet

RegNet model trained on imagenet-1k. It was introduced in the paper [Designing Network Design Spaces](#) and first released in [this repository](#). Disclaimer: The team releasing RegNet did not write a model card for this model so this model card has been written by the Hugging Face team.

#### Model description

The authors design search spaces to perform Neural Architecture Search (NAS). They first start from a high dimensional search space and iteratively reduce the search space by empirically applying constraints based on the best-performing models sampled by the current search space.

$n, 1, 1$                        $w_4, r/32, r/32$                        $w_i, r_i, r_i$   
 head                              stage 4                              block  $d_i$

Figure 6.13: A snippet of the shown description for the facebook/regnet-x-120 model.

## facebook/regnet-x-120

Results for evaluation on *imagenet*

Evaluated variations: [greyscale](#) [inverted](#) [rotated90](#) [mirrored](#) [flipped](#)

Task: image-classification    Origin: Hugging Face    Metrics: yes    Dataset: imagenet

[View imagenet dataset](#)                      [Compare results](#)                      [View on Hugging Face](#)

Used metrics: [PRECISION](#) [RECALL](#) [F1SCORE](#)

[View imagenet](#)                      The *imagenet* dataset

ImageNet image classification models are evaluated using the [validation split](#), which consists of 50.000 images, with exactly 50 images for each of the 1000 classes.

Figure 6.14: Info for the evaluation of the facebook/regnet-x-120 model on the ImageNet dataset.

### 6.2.3. Performance results

If the model has been evaluated, the results of each evaluation can be viewed. As was seen in basic information for our example model facebook/regnet-x-120, it has been evaluated on the ImageNet and ImageNet-Sketch datasets. When viewing the results of an evaluation, the user is first shown again some basic information of the evaluation. The information consists of two parts. First, some information about the evaluation itself, such as which dataset was used for this particular evaluation, which other variants have been evaluated, and which metrics were collected. There are also quick links to the external source of the model, and to compare the model's performance results against other models evaluated with the same dataset. The second part of this evaluation information is regarding the dataset used. Here the user can find a quick description of the dataset and split used for this evaluation, with links to both of them for more information. An example is shown in Figure 6.14<sup>4</sup>; this evaluation was performed on the `test` split of the ImageNet dataset, which contains 10.000 images evenly distributed over all 1.000 classes.

After the basic information, the user is first presented with the inference speed of the model. For each evaluation the total number of objects evaluated is recorded and the total time needed to

<sup>4</sup><https://www.metadatatoo.io/models/63a0b0d515a79b5cfb17a0c5/results/63b6e821f93ee77589ca804b>

Evaluation runtime			
Total objects in dataset	Total time in seconds	Time per object	Objects per second
100000	3720.17488	0.0744	13.44023

Figure 6.15: The speed and summary of characteristics for the evaluation of the facebook/regnet-x-120 model on the ImageNet dataset.

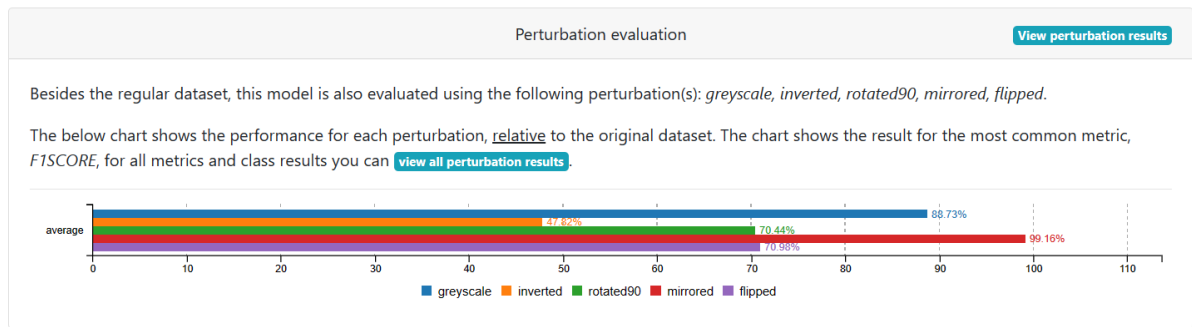


Figure 6.16: The summary of evaluation of the perturbations evaluation of the ImageNet dataset for the facebook/regnet-x-120.

evaluate those objects, as well as both the number of objects per second and the time it takes on average to evaluate a single object. The inference speed for the facebook/regnet-120-x model is shown in Figure 6.15.

Next, if the model's evaluation included the evaluation using dataset variants, a summary is shown. For our example facebook/regnet-x-120 model this is shown in Figure 6.16. First, a list of evaluated variants is given. For our model these are the *rotated90, flipped, mirrored, greyscale, and inverted* variants. For this chart a single metric is selected, one that gives an indication of performance of a model for the given evaluated task. Also, the chart shows just the overall average result of the perturbation evaluation, and the user is directed to view the full comparison of variant results for a more detailed look.

For certain tasks there might be specific visualisations that are appropriate for the task. In our running example, the evaluated task is *image-classification*. This is a multi-class classification task, for which a confusion matrix is often used to show results. In this matrix, it can be shown for each class how often mistakenly the model thought it was one of the other classes. Figure 6.17 shows this matrix for the evaluation of the facebook/regnet-x-120 model on the ImageNet dataset. The number of correct classifications for each class are on the diagonal of the matrix, with the other numbers, in red, show the number of times the model made the same mistake. Initially, the classes related to the 3 most common misclassifications during the evaluation are shown. The user can then select the specific classes they are interested in to view in the matrix.

From the confusion matrix, the user inspect for exactly which objects the model made the mistakes.



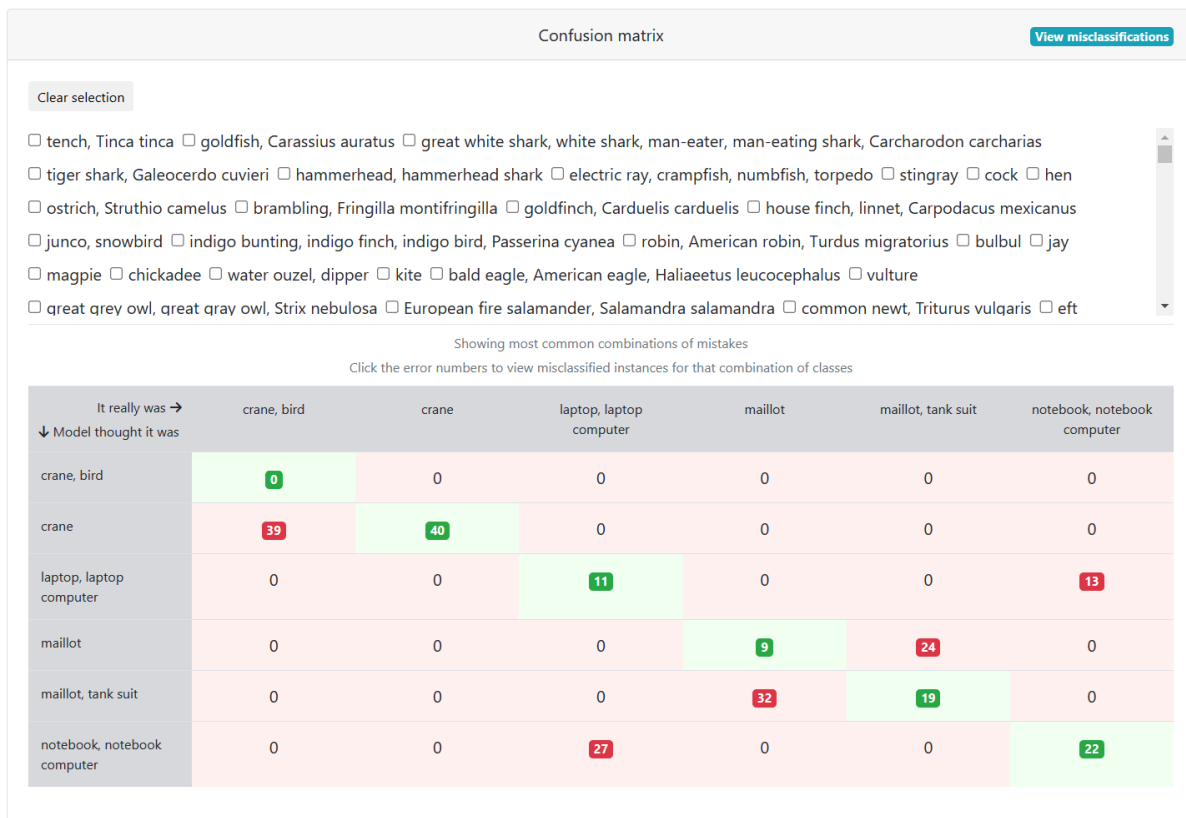


Figure 6.17: The confusion matrix with a selection of combinations of classes for the evaluation of the facebook/regnet-x-120 model on the ImageNet dataset.

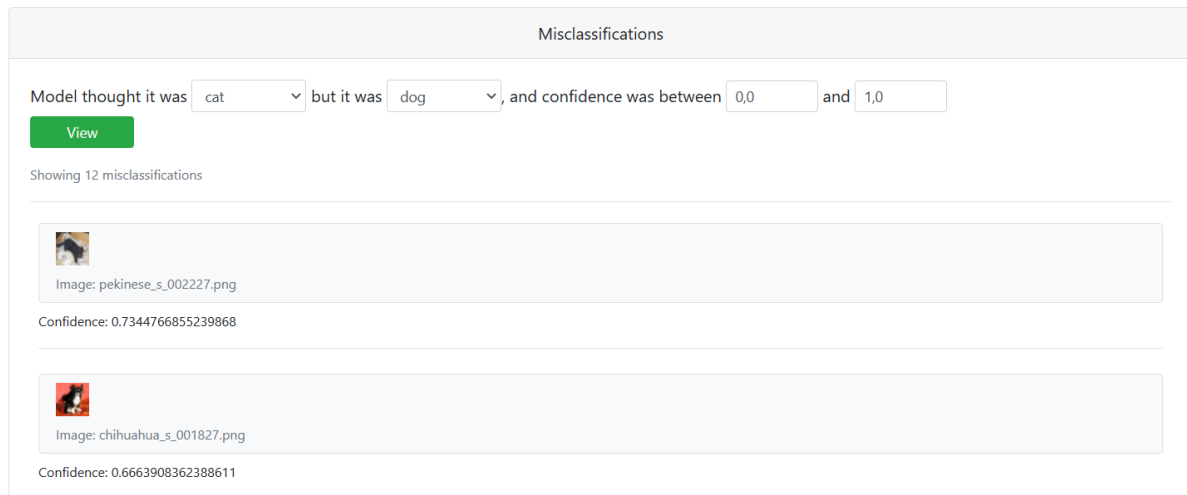


Figure 6.18: The misclassifications for the glopez/cifar-10 when evaluating using the CIFAR-10 dataset. Shown are instance where the model classified the image as 'cat', while in reality it was 'dog'.

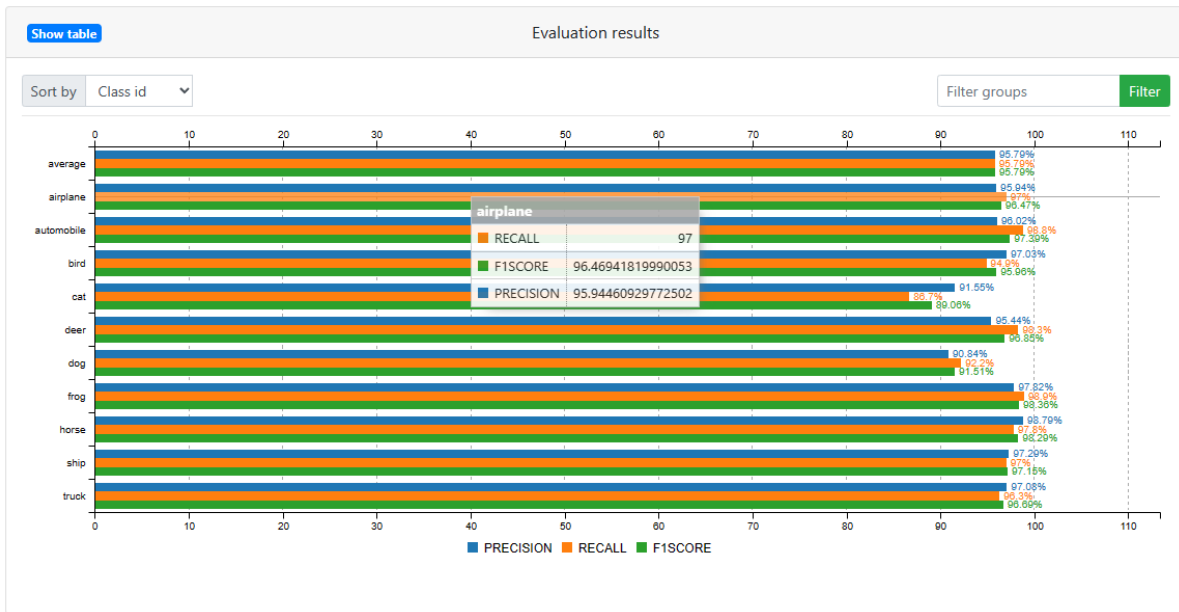
Figure 6.18 shows the misclassifications for another model, glopez/cifar-10<sup>56</sup>, evaluated on the CIFAR-10 dataset. Shown are 2 of the 12 images where the model thought the image was of a cat, but it was actually of a dog. The instances are sorted by confidence, ascending. Those instances are generally more interesting, as they were where the model confidently made errors, rather than where the model is "guessing". The user can select any other combination of classes, and filter the instances based on confidence if they are interested in a different range.

Finally, for each evaluation the evaluation results are shown. This is done in detail, in order to let the user get a good understanding of the model's performance. For multi-class classification tasks such as image-classification, this means that the metrics are calculated for each class separately, as well as average for the whole dataset. Figure 6.19a shows the diagram with these results for the evaluation of glopez/cifar-10 with the CIFAR-10 dataset. The metrics shown depend on the evaluated task. For multi-class classification tasks, they are the metrics of precision, recall, and f1-score. Again, when the mouse cursor is hovered over the diagram, a popup legend shows the user which value corresponds to which metric. The diagram is initially sorted according to the id of each class, but the classes can be sorted based on each of the metrics available. The user can also filter the shown classes, and if the user is not interested in one of the metrics, they can hide it. Figure 6.19b shows the same chart with a filter and sorting option applied, and one of the metrics hidden from the chart.

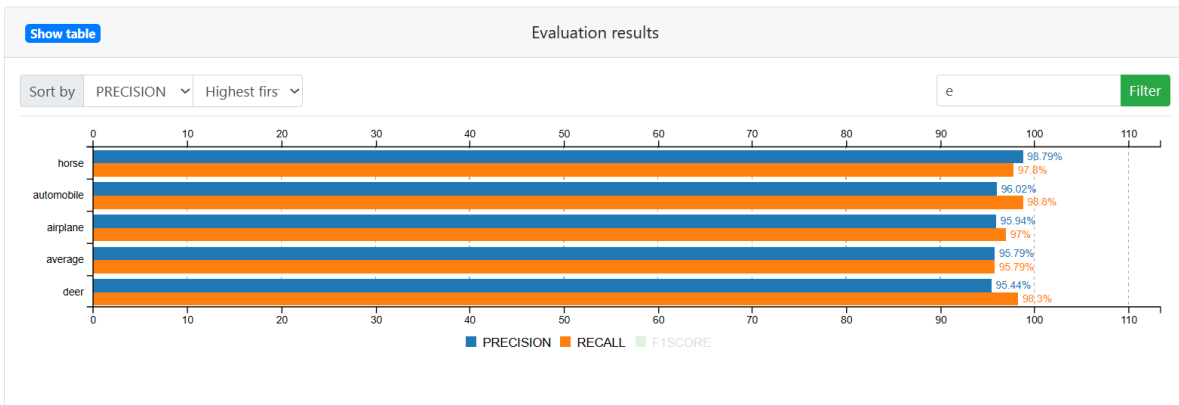
Besides a chart, the user can also opt to view the results in a table. The table can be sorted on any of the metrics, and can be filtered to only show selected classes matching the given input. Figure 6.20 shows the table corresponding to the diagram in Figure 6.19a.

<sup>5</sup><https://www.metadatasoo.io/models/633c34b646bfec446fc8a024>

<sup>6</sup><https://www.metadatasoo.io/models/633c34b646bfec446fc8a024/results/63a22ec70f95c6eaa2922a74/misclassifications>

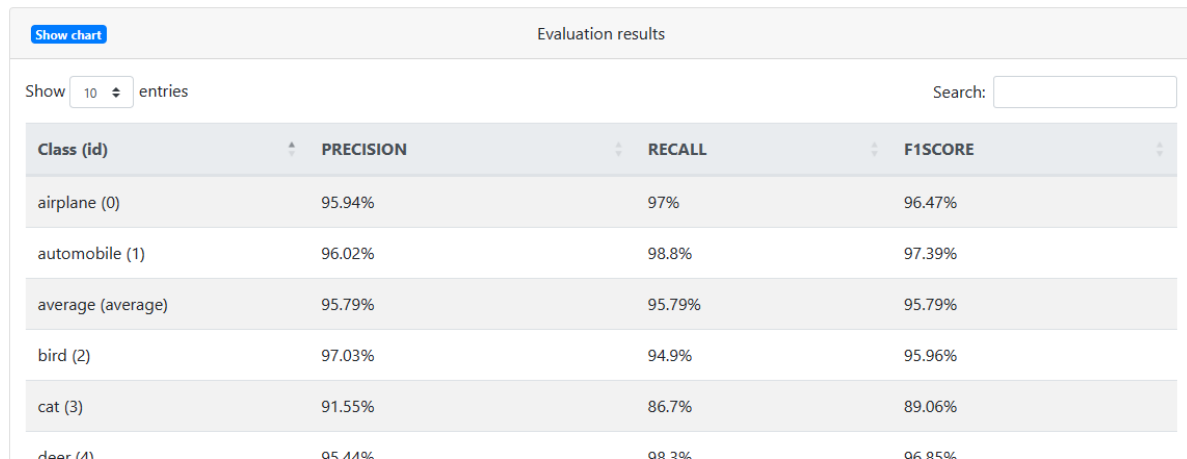


(a) The initial view of the chart showing performance results. The mouse cursor is hovered over the chart to show a popup with a small legend and the values of the metrics for the class.



(b) The same chart with a filter to only show classes containing "e", with the classes sorted on the achieved PRECISION, and the F1-Score metric hidden

Figure 6.19: The chart showing the metrics for each of the CIFAR-10 classes for the evaluation of the glopez/cifar-10 model on the CIFAR-10 dataset.



Class (id)	PRECISION	RECALL	F1SCORE
airplane (0)	95.94%	97%	96.47%
automobile (1)	96.02%	98.8%	97.39%
average (average)	95.79%	95.79%	95.79%
bird (2)	97.03%	94.9%	95.96%
cat (3)	91.55%	86.7%	89.06%
deer (4)	95.11%	98.2%	96.85%

Figure 6.20: The table showing the metrics for each of the ImageNet classes for the evaluation of the glopez/cifar-10 model on the CIFAR-10 dataset.

### 6.2.4. Variant comparison

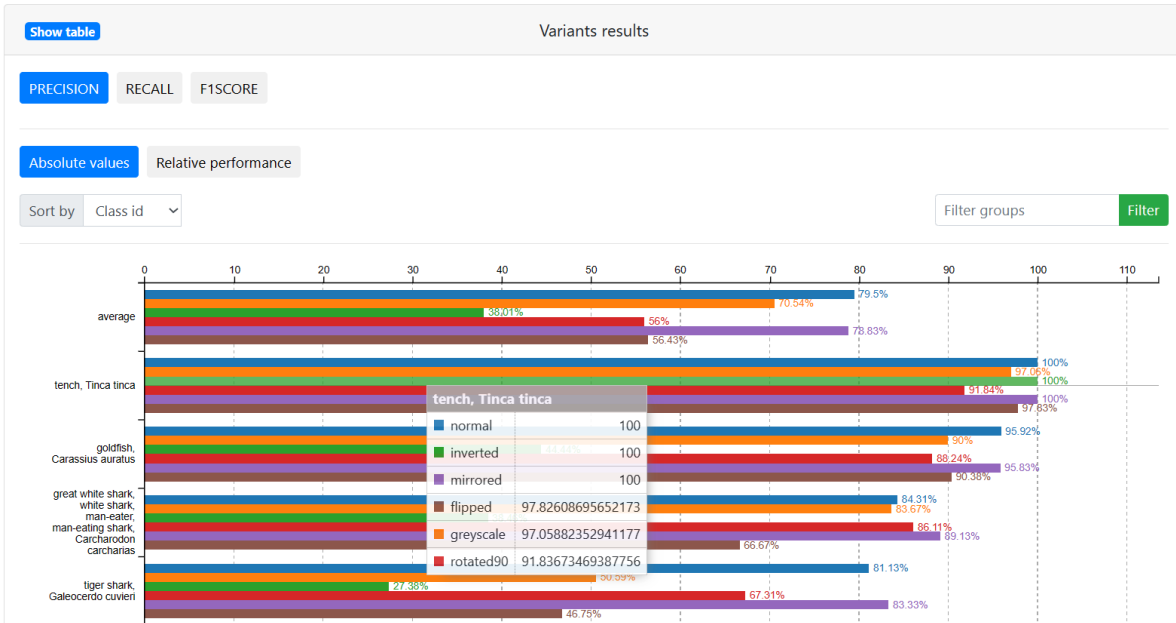
If the model has been evaluated with variants of datasets as well, the user can view the results for each variant, and compare against results obtained from the original dataset. As was seen in Figure 6.16, our example model facebook/regnet-x-120 has been evaluated with five variants of the ImageNet dataset. From the evaluation results for any variant, the comparison of all variants can be accessed. Here, a diagram is shown which is structured similar to the one that shows the results for the evaluation with the original dataset. For the comparison of variations for the facebook/regnet-x-120 model, this means that the comparison is made for each class of the evaluated ImageNet dataset, as partially shown in Figure 6.21a. However, as there are multiple results compared, rather than showing the calculated metrics for each class, the diagram shows the result for each variation for a single selected metric.

This diagram has similar options to customize the view. The user can again apply a filter, sort the classes according to performance of a single variant, hide the results of the variant they are not interested in, and view a table with these results. Furthermore, the user can select the metric they are interested in, and the diagram can show either the absolute values of each evaluation, or the results of the evaluated variations relative to those of the original dataset. The user can use the latter option to quickly get an insight of how each variation impacts the model's performance.

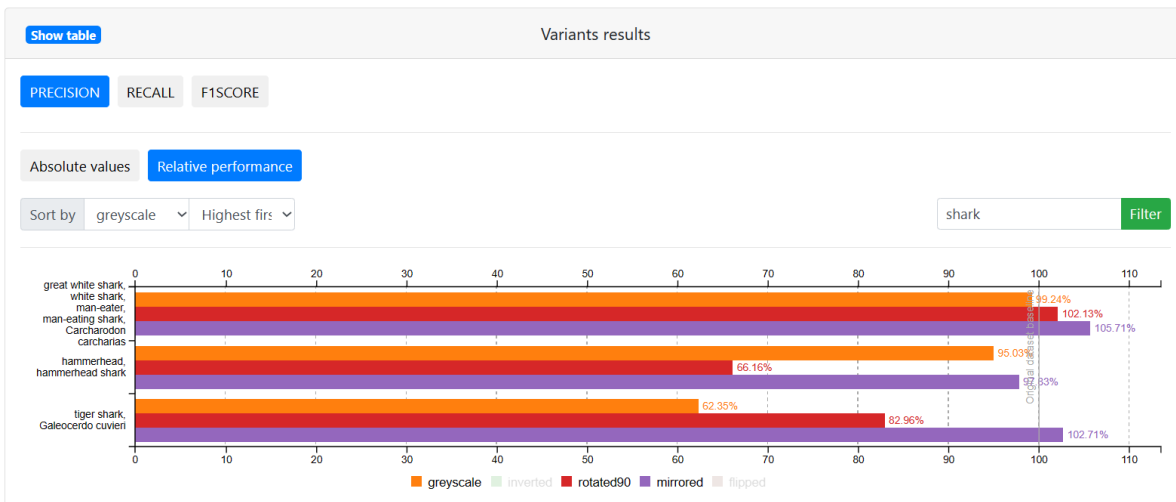
In Figure 6.21b the diagram shows the results of the evaluations of the variations relative to those of evaluation of the original dataset. At a glance the impact of each variation becomes clear. While every variation affects the overall performance negatively, the effect is different for each class, with the model even performing better for some variants for certain classes.

### 6.2.5. Model performance comparison

Besides comparing all evaluated models for a single dataset, as described in Section 6.1.2, a user can two or more models head-to-head as well. Upon reaching the model comparison tool, first



(a) Initial view showing the real values for each class, for the shown metric



(b) The view showing performance of each variant relative to the original dataset, thus clearly indicating how much each variant impacts the model's performance. The filter shark is applied, thus showing only classes containing this word, the classes are sorted by performance of the greyscale variant, and the results of multiple variants are hidden from the chart.

Figure 6.21: A view of the comparison of results achieved by evaluation of multiple variants of the same dataset for the facebook/regnet-x-120 model

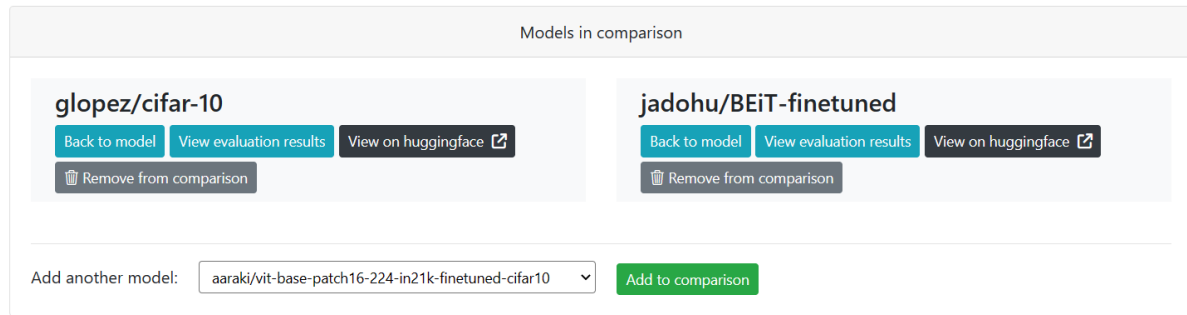


Figure 6.22: Selection of models to compare. Two CIFAR-10 image-classification models have been selected, with the option to add more models that have been evaluated with CIFAR-10.

some models need to be added to the comparison. Figure 6.22 shows this for a comparison<sup>7</sup> of image-classification models evaluated on the CIFAR-10 dataset. In this example, the `glopez/cifar-10` and `jadohu/BEiT-finetuned`<sup>8</sup> models have been selected for comparison. More models can be added, provided they have been evaluated with this dataset. From here, the user can quickly view each model's evaluation results, or go to the external source of the model.

After models have been added, two diagrams are used to compare the models. First the inference speed for all models is shown in a diagram. Figure 6.23 show this for our two selected models. The second diagram compares the evaluation results for each of the selected models. Since CIFAR-10 is used for image-classification, like ImageNet, during evaluation all metrics are calculated for each of the classes supported by the dataset. During comparison, all models will therefore also compared for each class as well. Figure 6.24 shows the comparison of the results of the two selected models. By now this diagram looks familiar. Just like the comparison of evaluated variants of a dataset, again the user must select a metric to compare. Then for each of the classes of the CIFAR-10 dataset the results for each model, for the selected metric, are shown. When hovering over the results of a class, a popup will be shown with the legend and values for each model. In our example, the results for the recall metric are shown. The classes of the diagram can again be sorted, and for our comparison we have sorted them based on the performance of the `glopez/cifar-10` model, in descending order.

From this example comparison, one can see that the `glopez/cifar-10` model evaluates images faster, on average, than the `jadohu/BEiT-finetuned` model. However, the performance is worse anywhere from one to ten percentage points, depending on the class. If one wants to use one of these specific models, the presented information lets them choose the right model for their needs.

<sup>7</sup><https://www.metadatasoo.io/results/compare/?id=63a0b62f1c242bc07af546ee,633c34b646bfec446fc8a024&dataset=6325c78d2aa8981c9d3b48bb>

<sup>8</sup><https://www.metadatasoo.io/models/63a0b62f1c242bc07af546ee>

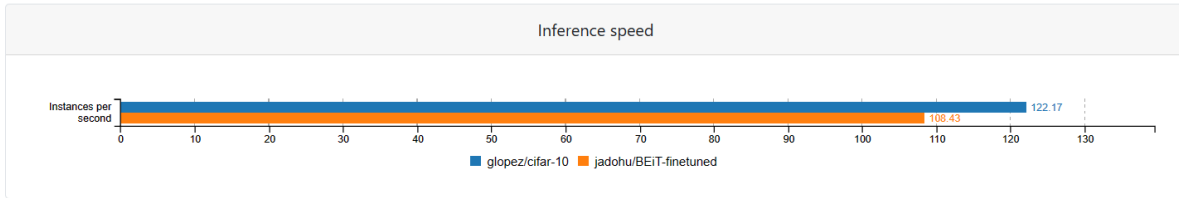


Figure 6.23: A comparison of the inference speed of two image-classification models evaluated on the CIFAR-10 dataset.

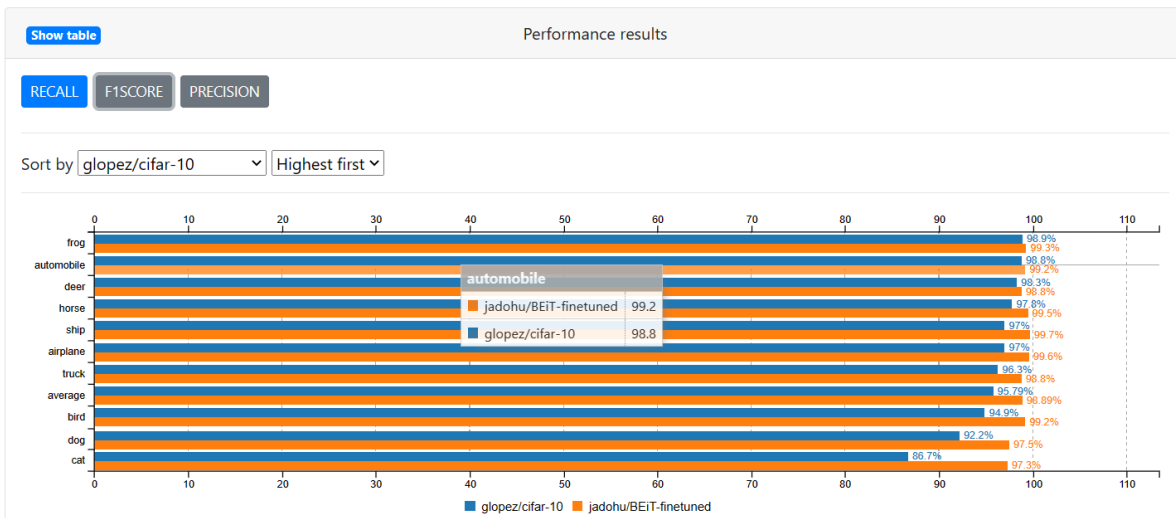


Figure 6.24: A comparison of the results two image-classification models evaluated on the CIFAR-10 dataset. The classes are sorted by the performance of the glopez/cifar-10 model, in descending order. The mouse cursor is hovered over the values for the automobile class, showing a popup with the legend and values for this class.







# Conclusion and limitations

As the sharing of machine learning model is becoming more popular, the number of model zoos is steadily growing. However, the current generation of model zoos provide little functionality beyond the sharing of models, in part due to a lack of performance metadata available for the models within them. In this work, we have provided a framework to enable and encourage the obtaining of rich performance metadata for machine learning models through the automatic evaluation of those models.

Furthermore, we have presented a benchmarking system built using our framework and used this system to attempt the evaluation of 1215 models from the Hugging Face and FiftyOne model zoos. Our results show an increase of the number of models for which performance metadata is present, with room to increase this number further. Our results also show a vast improvement in the richness of this metadata, providing disaggregated performance results, for evaluation using multiple datasets, and perturbations thereof, where current model zoos are often limited to a mere handful of performance results obtained by evaluating with a single dataset.

## 7.1. Fulfillment of objective

As part of this thesis we have posed two research questions. Our first research question was related to the problems currently existing surrounding the limitations of the provided performance metadata in current model zoos:

**RQ1:** What are the limitations of provided metadata in current model zoos?

We have answered this question by observing limitations of the performance metadata provided by current model zoos — a lack of performance results, the unstructured nature of reported performance results, a tendency to use only aggregated values, and the incomparability of the provided results — and in the process of obtaining these results — a reliance on self-reported

results. In addition to this, through literature survey we noted problems that plague performance reporting in the field of machine learning as a whole, such as non-reproducibility of reported results, and a tendency to use outdated or inappropriate metrics during evaluation of a model.

Our second research question was aimed to provide a solution for the found limitations in existing model zoos:

**RQ2:** How can we provide rich and comparable performance metadata for machine learning models

We have answered this question by providing a framework from which a benchmarking system can be created that overcomes the posed limitations. We have shown the validity of our solution by creating our own benchmarking system, and providing rich performance metadata for 1158 models from existing model zoos. For many of these models, no performance metadata was present until now, and for all others we have vastly increased the richness of the available metadata. Our benchmarking system can provide performance metadata for a high degree of models it evaluates, and the performance results provided are comparable, disaggregate, structured, and does not rely on users' input beyond the point of sharing a model.

## 7.2. Limitations of provided solution

Though we have fulfilled the objective of this thesis, some limitations of our framework can be observed as well. While our framework provides the tools to create a benchmarking system to evaluate models and obtain rich metadata, as any tool the end results depend on the wielder of the tool. It is entirely possible to our framework to evaluate models, and obtain nothing but aggregated results. However, as the calculation of metrics need only be defined once, and can be re-used during evaluation of models that perform many different tasks, we feel it regardless encourages the proper use of our tool.

Second, as exemplified by implemented benchmarking system, to run models that have differing requirements of the execution environment, multiple benchmarking system running in different environments are required. While not completely obstructing the evaluation of different types of models, this is an inelegant way to achieve evaluation of a varied set of models, and can not be avoided by the current version of our framework.

Lastly, and perhaps most importantly, our framework requires the training dataset of a model to be known to evaluate a model. This places a requirement on the metadata of the model for evaluation, that does not exist for users that merely want to use that model. This is particularly wry because the training dataset is sometimes mentioned in the name or description of the model, and would be easy to ascertain by a user. For some tasks the training dataset is not required to determine how a model should be evaluated. For example, the output of a text-summarization model — a model that shortens a long text while retaining the important parts of the input text — solely depends on the input, and not on the training dataset. For such cases it may be possible to relax this requirement. For tasks where knowledge of the training dataset is required, such as classification tasks, future research might provide simple ways to deduce it from the behavior of

the model itself.

### 7.3. Suggestions for future research

While our framework is an improvement over the current state-of-the-art solutions for providing performance metadata, improvements are possible, of which we will describe two.

First, a major drawback is the reliance on knowledge of the training dataset of a model in the evaluation process. This is not always known, which means a model cannot be evaluated, even though users are able to use it. Future research may be done to find ways of efficiently determining the training dataset, either from other metadata of the model, or through execution of the model itself.

Second, while our framework encourages providing rich performance metadata, it still merely provides numbers. Future research may be done to find ways of making higher level statements about a model's performance or biases that can be derived from the provided metadata. For example, for an image-classification model it might be said then that it performs poor on the classification of animals — not a specific class, but animals in general. In another example, for a summarization model an observation could be made that it tends to leave out important people in its summarizations of long texts, and these people are most often women. The recognition of such biases is important for the use of models, and while this information is available in the performance metadata, it is complex to uncover.



# Bibliography

- Abadi, M., Barham, P., Chen, J., Chen, Z., Davis, A., Dean, J., Devin, M., Ghemawat, S., Irving, G., Isard, M., et al. (2016). Tensorflow: A system for large-scale machine learning. *Osd*, 16(2016), 265–283.
- Akhtar, N., & Mian, A. (2018). Threat of adversarial attacks on deep learning in computer vision: A survey. *Ieee Access*, 6, 14410–14430.
- Banerjee, S., & Lavie, A. (2005). Meteor: An automatic metric for mt evaluation with improved correlation with human judgments. *Proceedings of the acl workshop on intrinsic and extrinsic evaluation measures for machine translation and/or summarization*, 65–72.
- Barocas, S., Guo, A., Kamar, E., Krones, J., Morris, M. R., Vaughan, J. W., Wadsworth, W. D., & Wallach, H. (2021). Designing disaggregated evaluations of ai systems: Choices, considerations, and tradeoffs. *Proceedings of the 2021 AAAI/ACM Conference on AI, Ethics, and Society*, 368–378.
- Blagec, K., Dorffner, G., Moradi, M., & Samwald, M. (2020). A critical analysis of metrics used for measuring progress in artificial intelligence. *arXiv preprint arXiv:2008.02577*.
- Buolamwini, J., & Gebru, T. (2018). Gender shades: Intersectional accuracy disparities in commercial gender classification. *Conference on fairness, accountability and transparency*, 77–91.
- Callison-Burch, C., Osborne, M., & Koehn, P. (2006). Re-evaluating the role of Bleu in machine translation research. *11th Conference of the European Chapter of the Association for Computational Linguistics*, 249–256. <https://aclanthology.org/E06-1032>
- Coleman, C., Narayanan, D., Kang, D., Zhao, T., Zhang, J., Nardi, L., Bailis, P., Olukotun, K., Ré, C., & Zaharia, M. (2017). Dawnbench: An end-to-end deep learning benchmark and competition. *Training*, 100(101), 102.
- Crisan, A., Drouhard, M., Vig, J., & Rajani, N. (2022). Interactive model cards: A human-centered approach to model documentation. *2022 ACM Conference on Fairness, Accountability, and Transparency*, 427–439.
- Detlefsen, N. S., Borovec, J., Schock, J., Jha, A. H., Koker, T., Di Liello, L., Stancl, D., Quan, C., Grechkin, M., & Falcon, W. (2022). Torchmetrics-measuring reproducibility in pytorch. *Journal of Open Source Software*, 7(70), 4101.
- Djulonga, J., Yung, J., Tschannen, M., Romijnders, R., Beyer, L., Kolesnikov, A., Puigcerver, J., Minderer, M., D’Amour, A., Moldovan, D., et al. (2021). On robustness and transferability of convolutional neural networks. *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 16458–16468.
- Dziugaite, G. K., Ghahramani, Z., & Roy, D. M. (2016). A study of the effect of jpg compression on adversarial images. *arXiv preprint arXiv:1608.00853*.

- Engstrom, L., Tran, B., Tsipras, D., Schmidt, L., & Madry, A. (2017). A rotation and a translation suffice: Fooling cnns with simple transformations.
- Ferri, C., Hernández-Orallo, J., & Modroi, R. (2009). An experimental comparison of performance measures for classification. *Pattern recognition letters*, 30(1), 27–38.
- Furner, J. (2020). Definitions of “metadata” : A brief survey of international standards. *Journal of the Association for Information Science and Technology*, 71(6), E33–E42.
- Gebru, T., Morgenstern, J., Vecchione, B., Vaughan, J. W., Wallach, H., Iii, H. D., & Crawford, K. (2021). Datasheets for datasets. *Communications of the ACM*, 64(12), 86–92.
- Hendrycks, D., & Dietterich, T. (2019). Benchmarking neural network robustness to common corruptions and perturbations. *arXiv preprint arXiv:1903.12261*.
- Kiela, D., Bartolo, M., Nie, Y., Kaushik, D., Geiger, A., Wu, Z., Vidgen, B., Prasad, G., Singh, A., Ringshia, P., Ma, Z., Thrush, T., Riedel, S., Waseem, Z., Stenetorp, P., Jia, R., Bansal, M., Potts, C., & Williams, A. (2021). Dynabench: Rethinking benchmarking in NLP. *Proceedings of the 2021 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, 4110–4124. <https://doi.org/10.18653/v1/2021.naacl-main.324>
- Krizhevsky, A., Hinton, G., et al. (2009). Learning multiple layers of features from tiny images.
- Li, Z., Hai, R., Bozzon, A., & Katsifodimos, A. (2022). Metadata representations for queryable ml model zoos.
- Lin, C.-Y. (2004). Rouge: A package for automatic evaluation of summaries. *Text summarization branches out*, 74–81.
- Liu, P., Fu, J., Xiao, Y., Yuan, W., Chang, S., Dai, J., Liu, Y., Ye, Z., Dou, Z.-Y., & Neubig, G. (2021). Explainaboard: An explainable leaderboard for nlp. *arXiv preprint arXiv:2104.06387*.
- Ma, Z., Ethayarajh, K., Thrush, T., Jain, S., Wu, L., Jia, R., Potts, C., Williams, A., & Kiela, D. (2021). Dynaboard: An evaluation-as-a-service platform for holistic next-generation benchmarking. *Advances in Neural Information Processing Systems*, 34, 10351–10367.
- Marie, B., Fujita, A., & Rubino, R. (2021). Scientific credibility of machine translation research: A meta-evaluation of 769 papers. *arXiv preprint arXiv:2106.15195*.
- Mathur, N., Baldwin, T., & Cohn, T. (2020). Tangled up in BLEU: Reevaluating the evaluation of automatic machine translation evaluation metrics. *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics*, 4984–4997. <https://doi.org/10.18653/v1/2020.acl-main.448>
- Mitchell, M., Wu, S., Zaldivar, A., Barnes, P., Vasserman, L., Hutchinson, B., Spitzer, E., Raji, I. D., & Gebru, T. (2019). Model cards for model reporting. *Proceedings of the conference on fairness, accountability, and transparency*, 220–229.
- Papineni, K., Roukos, S., Ward, T., & Zhu, W.-J. (2002). Bleu: A method for automatic evaluation of machine translation. *Proceedings of the 40th Annual Meeting of the Association for Computational Linguistics*, 311–318. <https://doi.org/10.3115/1073083.1073135>
- Paszke, A., Gross, S., Massa, F., Lerer, A., Bradbury, J., Chanan, G., Killeen, T., Lin, Z., Gimelshein, N., Antiga, L., et al. (2019). Pytorch: An imperative style, high-performance deep learning library. *Advances in neural information processing systems*, 32.

- Post, M. (2018). A call for clarity in reporting BLEU scores. *Proceedings of the Third Conference on Machine Translation: Research Papers*, 186–191. <https://www.aclweb.org/anthology/W18-6319>
- Raff, E. (2019). A step toward quantifying independently reproducible machine learning research. *Advances in Neural Information Processing Systems*, 32.
- Reiter, E. (2018). A structured review of the validity of bleu. *Computational Linguistics*, 44(3), 393–401.
- Riley, J. (2017). Understanding metadata. Washington DC, United States: National Information Standards Organization (<http://www.niso.org/publications/press/UnderstandingMetadata.pdf>), 23, 7–10.
- Russakovsky, O., Deng, J., Su, H., Krause, J., Satheesh, S., Ma, S., Huang, Z., Karpathy, A., Khosla, A., Bernstein, M., Berg, A. C., & Fei-Fei, L. (2015). ImageNet Large Scale Visual Recognition Challenge. *International Journal of Computer Vision (IJCV)*, 115(3), 211–252. <https://doi.org/10.1007/s11263-015-0816-y>
- Schelter, S., Boese, J.-H., Kirschnick, J., Klein, T., & Seufert, S. (2017). Automatically tracking metadata and provenance of machine learning experiments.
- von Werra, L., Tunstall, L., Thakur, A., Luccioni, A. S., Thrush, T., Piktus, A., Marty, F., Rajani, N., Mustar, V., Ngo, H., et al. (2022). Evaluate & evaluation on the hub: Better best practices for data and model measurement. *arXiv preprint arXiv:2210.01970*.
- Willis, C., & Stodden, V. (2020). Trust but verify: How to leverage policies, workflows, and infrastructure to ensure computational reproducibility in publication.
- Wolf, T., Debut, L., Sanh, V., Chaumond, J., Delangue, C., Moi, A., Cistac, P., Rault, T., Louf, R., Funtowicz, M., et al. (2020). Transformers: State-of-the-art natural language processing. *Proceedings of the 2020 conference on empirical methods in natural language processing: system demonstrations*, 38–45.
- Yadav, D., Jain, R., Agrawal, H., Chattopadhyay, P., Singh, T., Jain, A., Singh, S. B., Lee, S., & Batra, D. (2019). Evalai: Towards better evaluation systems for ai agents. *arXiv preprint arXiv:1902.03570*.
- Yuan, W., Neubig, G., & Liu, P. (2021). Bartscore: Evaluating generated text as text generation. In M. R. o, A. Beygelzimer, Y. Dauphin, P. Liang, & J. W. Vaughan (Eds.), *Advances in neural information processing systems* (pp. 27263–27277). Curran Associates, Inc. <https://proceedings.neurips.cc/paper/2021/file/e4d2b6e6fdeca3e60e0f1a62fee3d9dd-Paper.pdf>