# Prescriptive and Descriptive Approaches to Machine-Learning Transparency

David Adkins
davidadkins@fb.com
Meta AI
USA

Bilal Alsallakh*
bilalsal@fb.com
Meta AI
USA

Adeel Cheema
acheema@fb.com
Meta AI
USA

Narine Kokhlikyan
narine@fb.com
Meta AI
USA

Emily McReynolds
emcr@fb.com
Meta AI
USA

Pushkar Mishra*
pushkarmishra@fb.com
Meta AI
UK

Chavez Procope
cprocope@fb.com
Meta AI
USA

Jeremy Sawruk*
jsawruk@fb.com
Meta AI
USA

Erin Wang
yulinw@fb.com
Meta AI
USA

Polina Zvyagina
polinaz@fb.com
Meta AI
USA

## ABSTRACT

Specialized documentation techniques have been developed to communicate key facts about machine-learning (ML) systems and the datasets and models they rely on. Techniques such as Datasheets, FactSheets, and Model Cards have taken a mainly descriptive approach, providing various details about the system components. While the above information is essential for product developers and external experts to assess whether the ML system meets their requirements, other stakeholders might find it less actionable. In particular, ML engineers need guidance on how to mitigate potential shortcomings in order to fix bugs or improve the system's performance. We survey approaches that aim to provide such guidance in a prescriptive way. We further propose a preliminary approach, called Method Cards, which aims to increase the transparency and reproducibility of ML systems by providing prescriptive documentation of commonly-used ML methods and techniques. We showcase our proposal with an example in small object detection, and demonstrate how Method Cards can communicate key considerations for model developers. We further highlight avenues for improving the user experience of ML engineers based on Method Cards.

*Equal contribution (authors list in alphabetical order).

*CHI '22, Apr 30– May 05, 2022, New Orleans, LA*
© 2022 Association for Computing Machinery.
ACM ISBN 978-1-4503-XXXX-X/18/06…$15.00
https://doi.org/10.1145/3491101.3519724

## CCS CONCEPTS

**• Software and its engineering → Documentation**; **• Computing methodologies → Machine learning**.

## KEYWORDS

Method Cards, Developer Experience, Transparency

**ACM Reference Format:**
David Adkins, Bilal Alsallakh, Adeel Cheema, Narine Kokhlikyan, Emily McReynolds, Pushkar Mishra*, Chavez Procope, Jeremy Sawruk*, Erin Wang, and Polina Zvyagina. 2022. Prescriptive and Descriptive Approaches to Machine-Learning Transparency. In *Extended Abstracts of the CHI 2022 Conference on Human Factors in Computing Systems.* ACM, New York, NY, USA, 9 pages. https://doi.org/10.1145/3491101.3519724

## 1 INTRODUCTION

With the rapid adoption of machine learning (ML) in practical applications, ML-specific documentation has become crucial to their transparency and to the user experience of different stakeholders of ML. In contrast to traditional software systems, the documentation of ML-based systems is an emerging area that poses new challenges and requirements. A variety of initiatives have been undertaken over the past few years to address these challenges. These initiatives aim to provide systematic ways to document ML-based systems.

The above-mentioned initiatives has focused on describing various components of ML-based systems. For example, Datasheets for Datasets [Gebru et al. 2021] focus on providing key details about the datasets used to develop ML models. Similarly, Model Cards [Mitchell et al. 2019] aim to communicate key facts about individual models such as their intended use, training and evaluation data, and relevant metrics. In addition, FactSheets [Arnold et al. 2019] aim to act as declarations of conformity for AI services by providing relevant details to the consumers of these services. While the above solutions advance the transparency of ML-based systems, their descriptive nature might limit their actionability for certain stakeholders. For example, ML engineers need specific information on how to retrain the system and how to mitigate certain problems, yet such knowledge is often sparsely documented.

We review and propose *prescriptive* solutions that aim to provide greater transparency into ML-based systems and to improve the experience of their developers. Our proposal caters mainly to expert stakeholders such as model developers and external model reviewers. Our contributions encompass:

- Motivating the need for a prescriptive approach to ML transparency, and surveying existing initiatives and forms of this approach (Section 3).
- Proposing Method Cards as a preliminary means to foster ML transparency and reproducibility, and improve the user experience of ML engineers (Section 4).

In section 5 we compare our proposals with previous work, and outline potential directions for future work.

## 2 BACKGROUND AND MOTIVATION

Providing transparency into ML systems involves distinct challenges. Here we provide an overview of three complementary approaches that contribute to ML transparency, along with a motivating example for prescriptive ones.

### 2.1 Transparency Through Documentation

The ABOUT ML initiative [Raji and Yang 2019] advocates documentation as a practical intervention to provide clarity into decision making in ML systems for stakeholders. The authors explain the value of both external and internal documentation such as establishing trust and demonstrating fairness [Holstein et al. 2019]. Furthermore, the authors argue that documentation is both an artifact and a process, showing how developing the documentation fosters ML developers to think critically about every step in the ML lifecycle. Accordingly, the initiative is led by the Partnership on AI consortium which continues to develop this process.

Documenting software systems is a long-standing goal of software engineering. Some of the emerging documentation initiatives for ML-based systems have adapted existing methods while others necessarily take a novel approach. Datasheets [Gebru et al. 2021], and to an extent Model Cards [Mitchell et al. 2019], grew from experiences with hardware specification documentation. Dataset Nutrition Labels [Chmielinski et al. 2020; Holland et al. 2018] and FactSheets [Arnold et al. 2019] are further prominent examples of these initiatives. Hind et al [Hind et al. 2020] report on the experience of AI teams in using FactSheets, and provide recommendations for easing the collection and flexible presentation of AI facts to promote transparency.

### 2.2 Transparency Through Reproducibility

The ability of stakeholders to reproduce an ML system significantly increases its transparency [Haibe-Kains et al. 2020]. In addition to testing the system on their own data points, reproducibility enables stakeholders to retrain the ML models independently. Besides ensuring trust in the system and its components, the ability to retrain the ML models is crucial to analyze and debug these models.

Reproducibility in ML cannot be simply solved by making the source code available. The nature of ML algorithms and how they are trained make it challenging to warrant *reproducibility of the results* [Sculley et al. 2015]. Bell and Kampman [Bell and Kampman 2021] proposed ideas to foster this reproducibility, inspired by the recent reformation in psychology. Examples for these are Multiverse analysis [Steegen et al. 2016], preregistration [Nosek et al. 2012], and encouraging the publication of negative results. Likewise, several proposals have been made to quantify and accordingly improve reproducibility [Pineau et al. 2021; Raff 2019].

### 2.3 Transparency Through Interpretability

Lipton [Lipton 2018] analyzes in detail the notion of transparency in interpretability. The author argues how transparency in this context is the opposite of "black-boxness" as it helps understanding the mechanism by which the model works. Such understanding can be at the level of the entire model, at the level of individual components such as parameters, and at the level of the training algorithm.

Weller [Weller 2019] identifies different use cases of transparency, and distinguishes between two types of interpretability solutions to support them: global (how an overall system works) and local (explaining a particular prediction). The author surveys available techniques for both types and argues how the transparency enabled by them can be provide insights into important model characteristics such as robustness and fairness.

### 2.4 Illustrative Example

Consider a team of ML engineers tasked with developing an image object detector for a specific application. The team has a variety of choices to make such as the model type to use (e.g. convolutional networks vs. Vision Transformers), the training paradigm (e.g. supervised vs. self-supervised learning, etc.), the architecture type (e.g. ResNet vs. VGGNet), the optimizer to use, and the input preprocessing operations to mention a few. To speed up the ideation process and to obtain a benchmarking baseline, the team starts by finetuning a Single-Shot Detector (SSD) with a ResNet-50 as a backbone convolutional network pre-trained on ImageNet. This transfer-learning paradigm is very popular, especially when the training data is limited. They resize their images to $640x480$ to maintain the aspect ratio, and train their model following the same training script used to train the backbone model. However, there are a variety of nuances that impact the performance and reliability of their model. A few examples of these issues are:

- The resizing operation in popular deep-learning frameworks has a subtle aliasing flaw [Parmar et al. 2021].
- The input size used was shown to induce parity issues in ResNet models [Alsallakh et al. 2021a], leading to skewness in the learned filters.
- The default 0-padding used in ResNet was shown to impact small object detection [Alsallakh et al. 2021b], leading to blind spots.
- Transfer learning from a supervised model trained on ImageNet can lead to conflicts between the ImageNet categories and the objects of the target domain [Zoph et al. 2020].

Fortunately, it is possible to mitigate the above issues by making careful architectural, training, and preprocessing choices. However, these issues might not be widely known among practitioners and are often left unaddressed. Our goal is to provide ML engineers with actionable guidance on available ML methods and how to use them effectively in the systems they intend to develop. In contrast to the descriptive documentation techniques mentioned in Section 2.1, our guidance follows a prescriptive approach. As an rough metaphor, Model Cards and FactSheets are analogous to user manuals that are shipped with software programs and automobiles. While we aim to create prescriptive solutions that are analogous to computer programming recipes [Press et al. 1989] and car repair manuals.

## 3 A PRESCRIPTIVE APPROACH TO ML TRANSPARENCY

Software engineering has a long history of employing prescriptive models [Ludewig 2003] for various purposes such as effective diffusion [Raghavan and Chand 1989], safety certification [Hawkins et al. 2013], and error analysis [Meng and Amalathas 2019]. A key distinction between descriptive and prescriptive models is that the former describe an existing system, while the latter specify how a new system should be created [Ludewig 2003].

Descriptive approaches such as documenting general information of an existing system can sometimes be viewed by ML practitioners as time-consuming with a lack of incentives and/or intelligibility concerns about the documentation [Miceli et al. 2021]. Prescriptive approaches, on the other hand, provide explicit instructions on how to develop and deploy a solution on how to handle unexpected situations. This is useful in a variety of engineering disciplines both for modeling solutions [Finger and Dixon 1989; Heldal et al. 2016] and for guiding troubleshooting efforts [Eliasson et al. 2015; Wong et al. 2019]. Specific methods have been devised to support the documentation process of prescriptive software models [Clements et al. 2003].

The Machine Learning community has recently started employing prescriptive approaches to developing ML systems. The prescriptive nature of such approaches directly contribute to the transparency of the ML system, especially from the perspective of ML engineers and external reviewers. The following initiatives by the ML community involve prescriptive aspects and are targeted either at ML engineers or at ML researchers.

***Guidelines and Design Patterns.*** A number of design patterns have recently emerged in the ML community, both among practitioners [Apple 2019; Lakshmanan et al. 2020; Microsoft 2021; PAIR 2019; Shibui 2020], and among researchers [Washizaki et al. 2022; Yokoyama 2019]. These patterns tackle various aspects of model development and deployment, data and problem representation, training, serving and operation, reproducibility, quality-control and responsible AI usage. These patterns as well as complementary anti-patterns [Washizaki et al. 2019] capture best practices and are very helpful to replicate successful solutions in new applications.

***Recipes.*** As one of the leading deep-learning frameworks, PyTorch [Paszke et al. 2019] maintains a list of recipes [1], defined as "bite-sized, actionable examples of how to use specific PyTorch features". In contrast to the standard documentation [2], these recipes focus on executable solutions and demonstrate best practices established by the framework developers and its community.

***PWC Methods.*** The Papers With Code (PWC) community project has recently started maintaining a catalog of ML methods proposed in the research community for various ML tasks [3]. The catalog features a categorization of the method by task or data modality as well as an informative community-generated description of each method, along with statistics about its usage (Figure 1). Unlike Model Cards and FactSheets, these descriptions are independent of any specific models or systems, focusing instead on the underlying ML methods available in the literature, with ML researchers as the main audience.

***Reproducibility Checklists.*** Over the past few years, the ML research community has developed a reproducibility checklist, with the goal of ensuring that research articles and results are easy to reproduce [Pineau et al. 2021]. The checklist includes specific instructions regarding the models and algorithms presented, the datasets used, and the code shared, in addition to expectations regarding any theoretical claim or experimental results.

***ML Cheat Sheets.*** A variety of cheat sheets have been developed to provide an overview of ML algorithms. These sheets typically provide a flow chart on how to choose an algorithm for a specific problem, as in the ones developed by Microsoft Azure [4] or by SAS [5]. Similarity, ML Flashcards serve as a quick reference of ML concepts [6]. Additionally, several practitioners have compiled best practices for training ML models [Ang 2018; He et al. 2019]. Such guides help ML developers understand the strengths and limitations of available methods for crucial model components such as normalization and regularization.

***Interactive Hyperparameter Visualizations.*** A variety of interactive widgets aim to help ML engineers in making informed choices of their model's hyperparameters by visualizing how these choices impact the model. Examples include basic convolution arithmetic [7], choosing t-SNE parameters [Wattenberg et al. 2016], and understanding the impact of padding choices [Yuan et al. 2021].

## 4 METHOD CARDS

We propose Method Cards to guide ML engineers throughout the process of model development. Analogous to Model Cards, these cards aim to communicate key information about ML methods. The information comprises both prescriptive and descriptive elements, putting the main focus on ensuring that ML engineers are able to use these methods properly. The cards aim to support developers at multiple stages of the model-development process such as training, testing, and debugging. For this purpose, we propose a structure for the cards, outlined in Figure 2.

[1]Official PyTorch Recipes: https://pytorch.org/tutorials/recipes/recipes_index.html

[2]PyTorch Documentation https://pytorch.org/docs/stable/index.html

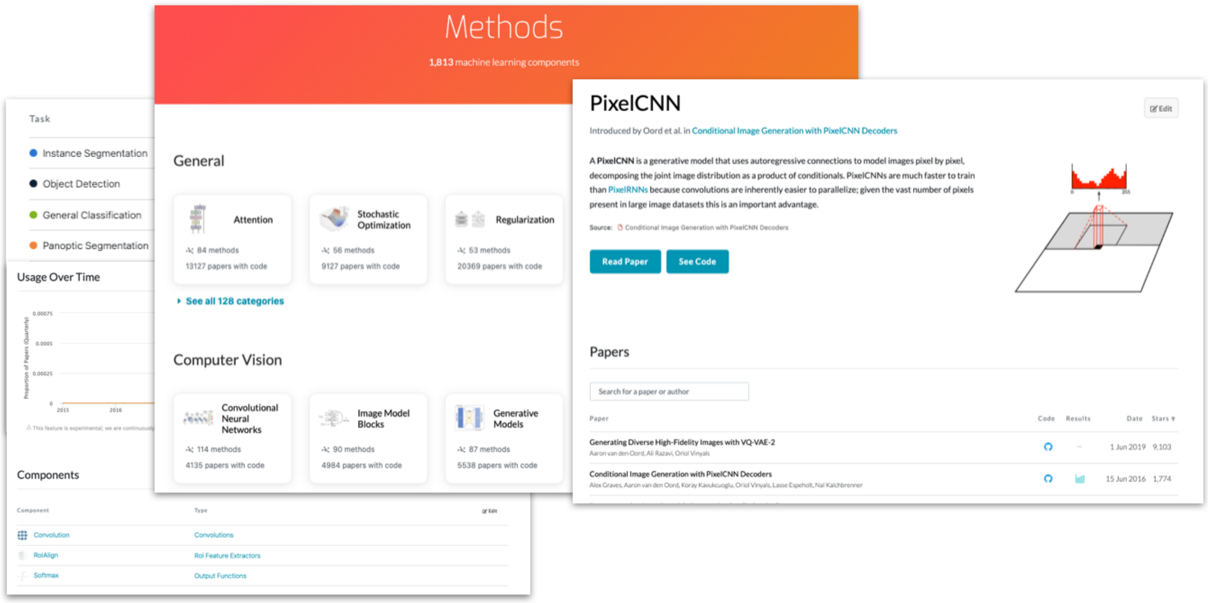
[3]https://paperswithcode.com/methods

[4]https://blogs.sas.com/content/subconsciousmusings/2020/12/09/machine-learning-algorithm-use/

[5]https://docs.microsoft.com/en-us/azure/machine-learning/algorithm-cheat-sheet

[6]https://machinelearningflashcards.com/

[7]Convolution Visualizer by Edward Yang: https://ezyang.github.io/convolution-visualizer/


Methods

1,813 machine learning components

General

Attention — 84 methods — 13127 papers with code

Stochastic Optimization — 56 methods — 9127 papers with code

Regularization — 53 methods — 20369 papers with code

See all 128 categories

Computer Vision

Convolutional Neural Networks — 114 methods — 4135 papers with code

Image Model Blocks — 90 methods — 4984 papers with code

Generative Models — 87 methods — 5538 papers with code

Task

- Instance Segmentation
- Object Detection
- General Classification
- Panoptic Segmentation

Usage Over Time

Components

| Component | Type |
|---|---|
| Convolution | Convolutions |
| RoIAlign | RoI Feature Extractors |
| Softmax | Output Functions |

PixelCNN

Introduced by Oord et al. in Conditional Image Generation with PixelCNN Decoders

A **PixelCNN** is a generative model that uses autoregressive connections to model images pixel by pixel, decomposing the joint image distribution as a product of conditionals. PixelCNNs are much faster to train than PixelRNNs because convolutions are inherently easier to parallelize; given the vast number of pixels present in large image datasets this is an important advantage.

Source: Conditional Image Generation with PixelCNN Decoders

Read Paper | See Code

Papers

Search for a paper or author

| Paper | Code | Results | Date | Stars |
|---|---|---|---|---|
| Generating Diverse High-Fidelity Images with VQ-VAE-2<br>Aaron van den Oord, Ali Razavi, Oriol Vinyals | | – | 1 Jun 2019 | 9,103 |
| Conditional Image Generation with PixelCNN Decoders<br>Alex Graves, Aaron van den Oord, Koray Kavukcuoglu, Oriol Vinyals, Lasse Espeholt, Nal Kalchbrenner | | | 15 Jun 2016 | 1,774 |

**Figure 1: Documenting ML methods in Papers With Code. For each method, the documentation includes brief textual and graphical descriptions, a list of papers that introduce or use the method, and a list of ML tasks the method is suited for.**

For each prompt in Method Cards, the method creators are expected to provide sufficient instructions and documentation to guide ML engineers. These instructions should help the engineers in choosing suited preprocessing steps, model components, and hyperparameters for their task. Furthermore, the instructions should make these engineers aware of how their choices potentially impact the model's behavior, how to evaluate this impact, and how to handle prediction errors accordingly. Finally, the instructions should explicitly inform the engineers on responsible usage of the method by addressing potential fairness and privacy concerns.

The instructions and documentation needed for the prompts in Method Cards can be enriched with the prescriptive approaches outlined in Section 3. IBM's FactSheets provide several examples that demonstrate how to format and deliver such rich information at multiple levels of detail in a usable way.

## 4.1 Granularity of ML Methods

The sections and prompts we propose in Figure 2 focus on ML methods that are sufficient to produce a proper ML model with defined input, output, and task. Examples for these are object detection methods such as Single-shot Detectors [Liu et al. 2016] and language modelling methods such as Generative Pre-trained Transformers (GPT) [Radford et al. 2019]. It is possible to create Model Cards for the models created using these methods.

We refer to ML methods that focus on specific parts of a model as **ML components**, following the PWC terminology. Examples of these are pooling, regularization, and normalization operations. These components correspond to specific prompts in our proposed Method Cards, specifically, the ones in "Data Preparation" and "Modelling and Training". The ML community is constantly studying these components, identifying their strengths and weaknesses in different ML applications. PWC provides a good overview of these studies, which can help the creators of Method Cards provide actionable recommendations about these components.

Likewise, **ML systems** often encompass multiple ML models, as well as non-ML components such as data acquisition and human-in-the-loop interfaces. Examples of these are fraud detection systems that rely on ML models to identify abnormal behavior and on human reviewers to take appropriate action when an automated decision could not be determined reliably. Providing transparency into such systems and services require artifacts similar to FactSheets.

## 4.2 Maintaining Method Cards

Within corporations, ML teams can benefit from maintaining method cards for their common use cases. These cards can be curated collectively and updated regularly as the team experiments with various components and parameterizations, and as they gain insights about their method's weaknesses and edge cases. Such documentation is important for onboarding new ML engineers and to enable them to maintain the ML models and to retrain them when needed. As such, Method Cards offer a systematic way to communicate best ML practices identified by the team.

Since ML methods share a variety of components and can have similar goals, their cards can naturally overlap. It is often possible to categorize these methods into a hierarchy, and to document shared information into cards that correspond to high levels in the hierarchy. For example, methods based on convolutional neural networks can inherit generic considerations related to these models such as choosing a suited input size (Section 2.4). Likewise, object detection methods can inherent common considerations such as evaluation metrics and data preparation. The hierarchy used to categorize PWC Methods could serve as a useful reference.

**Method Card Template**

**Basic Method Information**

- Name, version, and application domain(s).
- Method purpose and appropriate uses.
- Method definition, published literature, reference implementation.
- Example input and output.

**Safety and Troubleshooting**

- Inappropriate uses and common usage pitfalls.
- Known weaknesses, biases, and privacy leakage.
- How to detect biases in the model internals.
- Common failure modes, potential root causes, and possible mitigations via hyperparameter tuning or training data expansion.

**Data Preparation**

- Input and output format, shape, and data type.
- Data transformation and normalization.
- Recommended sampling and balancing.
- Recommended batching scheme and batch size.
- Required data augmentation and shuffling.
- Validation and train-test splitting schemes.

**Modelling and Optimization**

- Architecture family and components used.
- A list of hyperparameters, along with applicable values and their known impact.
- Training objective(s), loss(es), and optimizer(s).
- Parameter initialization / self pre-training / transfer from a trained baseline (specify datasets).
- Regularization scheme, capacity selection.
- If applicable, learning rate and schedulers.
- Weight quantization, recommended bit depth.
- Possibilities to compile the model graph.
- Parallelization at training and inference time.
- Recommended model compression techniques.

**Method Benchmarking**

- Performance metric(s) and applicable threshold(s).
- Threshold selection.
- Fairness evaluation and subgroup comparison.
- Overfitting detection.
- Training and inference time efficiency.
- Available benchmarks.

**Interpretability and Explainability**

- Applicable feature attribution methods, and how they can help explain model predictions.
- How to identify influential training instances behind a specific model prediction.
- How to identify internal concepts and features learned using the method.

**Robustness**

- Known vulnerabilities to adversarial attacks, and recommended mitigation.
- Out-of-distribution behavior.
- Detecting and mitigating data and model drifts.

**Figure 2: Suggested sections in Method Cards, along with the prompts they capture. Providing instructions for each prompt offers guidance to ML engineers and helps external reviewers evaluate the maturity of the model development processes.**

.

## 4.3 An Example of Method Cards

In the appendix we provide an example method card, created for traffic light detection using an SSD. The card exemplifies how a team of ML engineers can collectively document the nuances mentioned in Section 2.4, both based on published literature and on their own observations and experiences. Notice how the information presented is independent of a specific model. For example, the challenges of using SSDs for small object detection are inherent to the method, and can potentially impact any model developed using this method. Likewise, many challenges arise from the nature of the problem, such as the similarity of red traffic lights with automobile tail lights.

In the example, we used concise descriptions and instructions to avoid turning the curation and consumption of Method Cards into a burden. Nevertheless, the curators can elaborate on certain prompts by providing links to executable notebooks, articles, and further materials needed to clarify the prompts.

## 5 DISCUSSION

*Possible Applications.* Besides serving as a tool for prescriptive documentation, Method Cards, can further support building interactive tools to improve ML development processes. For example, a formal specification of certain prompts can be used to allow generating ML code templates. These templates serve as a useful starting points that ML developers can adapt, and help reducing potential coding errors as demonstrated in ClassyVision templates [Adcock et al. 2019]. As another application, Method Cards can power ML-aware code analyzers that aim can issue warnings when the code uses unrecommended parameter combinations. For examples a Method Card can include a rule that an `RMSProp` optimizer should be used if MobileNet or EfficientNet are used as a convolutional feature extractor instead of the default SGD with momentum.

Our effort to explore prescriptive solutions started in response to preliminary interviews we conducted with ML engineers who work on various ML problems in practical applications. Of their frequent pain points are the ability to make informed choices when designing and training new models, as well as finding solutions to improve the accuracy of existing models. As such, Method Cards aim to fill one of the gaps identified by the Model Cards authors:

> *It seems unlikely, at least in the near term, that model cards could be standardized or formalized to a degree needed to prevent misleading representations of model results (whether intended or unintended). It is therefore important to consider model cards as one transparency tool among many, which could include, for example, algorithmic auditing by third-parties, [...]" - [Mitchell et al. 2019, p. 9].*

While Model Cards and FactSheets put main focus on documenting existing models, Method Cards focus more on the underlying methodical and algorithmic choices that need to be considered when creating and training these models. As a rough analogy, if Model Cards and FactSheets provide nutrition information about cooked meals, Method Cards provide the recipes.

*Comparison with PWC Methods.* PWC Methods have mainly focused on offering a catalog of methods extracted from ML literature, designed to be usable mainly for researchers. On the other hand, Method Cards have ML engineers and algorithmic reviewers as target users, focusing on transparent, reproducible, and responsible usage of these methods. Ultimately, interested stakeholders in the ML community can augment PWC Methods with the prescriptive information we proposed in our template to help meet these requirements.

*The Usability of Method Cards.* A major challenge in offering documentation-based transparency is the effort needed by ML engineers to develop the needed artifacts [Miceli et al. 2021], compared with the perceived benefits. Finding the appropriate level of detail is a key to alleviate the burden and maximize the value, as the authors of FactSheets reported [Hind et al. 2020]. Accordingly, we consider the template suggested in Figure 2 to be an reference example, and encourage ML engineers to refine the prompts to fit their needs. Furthermore, interactive interfaces can help presenting the information at multiple levels of detail. For example, detailed answers can help onboard new team members, while summarized bullet lists are more suited for external reviewers.

## 6 CONCLUSION

We presented an overview of prescriptive approaches in ML documentation and their ability to facilitate transparency into ML algorithms. We further presented Method Cards that leverage descriptive and prescriptive elements to directly address the challenges surrounding algorithmic transparency. Unlike Model Cards, these cards do not describe specific ML models, focusing instead on providing guidance on how to properly use ML methods to define and train these models. This helps increase reproducibility of ML and enables external reviewers to determine the appropriateness of the methods used in ML-based solutions. Furthermore, this enables ML engineers to develop best practices to mitigate potential shortcomings and to improve the system's performance. We presented an example of Method Cards along with potential applications in generating templates and powering static analyzers of ML code.

## REFERENCES

A. Adcock, V. Reis, M. Singh, Z. Yan, L. van der Maaten, K. Zhang, S. Motwani, J. Guerin, N. Goyal, I. Misra, L. Gustafson, C. Changhan, and P. Goyal. 2019. Classy Vision. https://github.com/facebookresearch/ClassyVision.

Bilal Alsallakh, Narine Kokhlikyan, Vivek Miglani, Shubham Muttepawar, Edward Wang, Sara Zhang, David Adkins, and Orion Reblitz-Richardson. 2021a. Debugging the Internals of Convolutional Networks. In *eXplainable AI approaches for debugging and diagnosis - NeurIPS Workshop.*

Bilal Alsallakh, Narine Kokhlikyan, Vivek Miglani, Jun Yuan, and Orion Reblitz-Richardson. 2021b. Mind the Pad – CNNs Can Develop Blind Spots. In *Intl. Conference on Learning Representations (ICLR).*

Long Ang. 2018. A bunch of tips and tricks for training deep neural networks. (2018). https://towardsdatascience.com/a-bunch-of-tips-and-tricks-for-training-deep-neural-networks-3ca24c31ddc8

Apple. 2019. Human interface guidelines for machine learning. (2019). https://developer.apple.com/design/human-interface-guidelines/machine-learning/

Matthew Arnold, Rachel KE Bellamy, Michael Hind, Stephanie Houde, Sameep Mehta, Aleksandra Mojsilović, Ravi Nair, K Natesan Ramamurthy, Alexandra Olteanu, David Piorkowski, et al. 2019. FactSheets: Increasing trust in AI services through supplier's declarations of conformity. *IBM Journal of Research and Development* 63, 4/5 (2019), 6–1.

Karsten Behrendt, Libor Novak, and Rami Botros. 2017. A deep learning approach to traffic lights: Detection, tracking, and classification. In *2017 IEEE International Conference on Robotics and Automation (ICRA).* IEEE, 1370–1377.

Samuel J Bell and Onno P Kampman. 2021. Perspectives on Machine Learning from Psychology's Reproducibility Crisis. *ICLR Workshop on Science and Engineering of Deep Learning arXiv:2104.08878* (2021).

Kasia S Chmielinski, Sarah Newman, Matt Taylor, Josh Joseph, Kemi Thomas, Jessica Yurkofsky, and Yue Chelsea Qiu. 2020. The dataset nutrition label (2nd Gen): Leveraging context to mitigate harms in artificial intelligence. In *NeurIPS Workshop on Dataset Curation and Security.*

Paul Clements, David Garlan, Reed Little, Robert Nord, and Judith Stafford. 2003. Documenting software architectures: views and beyond. In *25th International Conference on Software Engineering, 2003. Proceedings.* IEEE, 740–741.

Ulf Eliasson, Rogardt Heldal, Patrizio Pelliccione, and Jonn Lantz. 2015. Architecting in the automotive domain: Descriptive vs prescriptive architecture. In *2015 12th Working IEEE/IFIP Conference on Software Architecture.* IEEE, 115–118.

Susan Finger and John R Dixon. 1989. A review of research in mechanical engineering design. Part I: Descriptive, prescriptive, and computer-based models of design processes. *Research in engineering design* 1, 1 (1989), 51–67.

Timnit Gebru, Jamie Morgenstern, Briana Vecchione, Jennifer Wortman Vaughan, Hanna Wallach, Hal Daumé III, and Kate Crawford. 2021. Datasheets for Datasets. *Commun. ACM* 64, 12 (nov 2021), 86–92. https://doi.org/10.1145/3458723

Robert Geirhos, Patricia Rubisch, Claudio Michaelis, Matthias Bethge, Felix A. Wichmann, and Wieland Brendel. 2019. ImageNet-trained CNNs are biased towards texture; increasing shape bias improves accuracy and robustness.. In *International Conference on Learning Representations.* https://openreview.net/forum?id=Bygh9j09KX

Benjamin Haibe-Kains, George Alexandru Adam, Ahmed Hosny, Farnoosh Khodakarami, Levi Waldron, Bo Wang, Chris McIntosh, Anna Goldenberg, Anshul Kundaje, Casey S Greene, et al. 2020. Transparency and reproducibility in artificial intelligence. *Nature* 586, 7829 (2020), E14–E16.

Richard Hawkins, Ibrahim Habli, Tim Kelly, and John McDermid. 2013. Assurance cases and prescriptive software safety certification: A comparative study. *Safety science* 59 (2013), 55–71.

Tong He, Zhi Zhang, Hang Zhang, Zhongyue Zhang, Junyuan Xie, and Mu Li. 2019. Bag of tricks for image classification with convolutional neural networks. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition.* 558–567.

Rogardt Heldal, Patrizio Pelliccione, Ulf Eliasson, Jonn Lantz, Jesper Derehag, and Jon Whittle. 2016. Descriptive vs prescriptive models in industry. In *Proceedings of the acm/ieee 19th international conference on model driven engineering languages and systems.* 216–226.

Michael Hind, Stephanie Houde, Jacquelyn Martino, Aleksandra Mojsilovic, David Piorkowski, John Richards, and Kush R Varshney. 2020. Experiences with improving the transparency of AI models and services. In *Extended Abstracts of the CHI Conference on Human Factors in Computing Systems.* 1–8.

Sarah Holland, Ahmed Hosny, Sarah Newman, Joshua Joseph, and Kasia Chmielinski. 2018. The dataset nutrition label: A framework to drive higher data quality standards. *arXiv preprint arXiv:1805.03677* (2018).

Kenneth Holstein, Jennifer Wortman Vaughan, Hal Daumé III, Miro Dudik, and Hanna Wallach. 2019. Improving fairness in machine learning systems: What do industry practitioners need?. In *Proceedings of the 2019 CHI conference on human factors in computing systems.* 1–16.

Valliappa Lakshmanan, Sara Robinson, and Michael Munn. 2020. *Machine learning design patterns.* O'Reilly Media.

Zachary C Lipton. 2018. The Mythos of Model Interpretability: In machine learning, the concept of interpretability is both important and slippery. *Queue* 16, 3 (2018), 31–57.

Wei Liu, Dragomir Anguelov, Dumitru Erhan, Christian Szegedy, Scott Reed, Cheng-Yang Fu, and Alexander C Berg. 2016. SSD: Single shot multibox detector. In *European Conference on Computer Vision*. 21–37.

Jochen Ludewig. 2003. Models in software engineering–an introduction. *Software and Systems Modeling* 2, 1 (2003), 5–14.

Wong Hoo Meng and Sagaya Sabestinal Amalathas. 2019. A new approach towards developing a prescriptive analytical logic model for software application error analysis. In *Proceedings of the Computational Methods in Systems and Software*. Springer, 256–274.

Milagros Miceli, Tianling Yang, Laurens Naudts, Martin Schuessler, Diana Serbanescu, and Alex Hanna. 2021. Documenting Computer Vision Datasets: An Invitation to Reflexive Data Practices. In *Proceedings of the 2021 ACM Conference on Fairness, Accountability, and Transparency*. 161–172.

Microsoft. 2021. The HAX Toolkit. (2021). https://www.microsoft.com/en-us/haxtoolkit/

Margaret Mitchell, Simone Wu, Andrew Zaldivar, Parker Barnes, Lucy Vasserman, Ben Hutchinson, Elena Spitzer, Inioluwa Deborah Raji, and Timnit Gebru. 2019. Model cards for model reporting. In *Proceedings of the conference on fairness, accountability, and transparency*. 220–229.

Brian A Nosek, Jeffrey R Spies, and Matt Motyl. 2012. Scientific utopia: II. Restructuring incentives and practices to promote truth over publishability. *Perspectives on Psychological Science* 7, 6 (2012), 615–631.

Google PAIR. 2019. People + AI Guidebook. (2019). https://pair.withgoogle.com/guidebook

Gaurav Parmar, Richard Zhang, and Jun-Yan Zhu. 2021. On Buggy Resizing Libraries and Surprising Subtleties in FID Calculation. *arXiv preprint arXiv:2104.11222* (2021).

Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, et al. 2019. PyTorch: An Imperative Style, High-Performance Deep Learning Library. In *Advances in Neural Information Processing Systems (NeurIPS)*. 8024–8035.

Joelle Pineau, Philippe Vincent-Lamarre, Koustuv Sinha, Vincent Larivière, Alina Beygelzimer, Florence d'Alché Buc, Emily Fox, and Hugo Larochelle. 2021. Improving reproducibility in machine learning research: a report from the NeurIPS 2019 reproducibility program. *Journal of Machine Learning Research* 22 (2021).

William H Press, Brian P Flannery, Saul A Teukolsky, William T Vetterling, et al. 1989. Numerical recipes.

Alec Radford, Jeff Wu, Rewon Child, David Luan, Dario Amodei, and Ilya Sutskever. 2019. Language Models are Unsupervised Multitask Learners. (2019).

Edward Raff. 2019. A step toward quantifying independently reproducible machine learning research. *Advances in Neural Information Processing Systems* 32 (2019), 5485–5495.

Sridhar A. Raghavan and Donald R. Chand. 1989. Diffusing software-engineering methods. *IEEE software* 6, 4 (1989), 81–90.

Inioluwa Deborah Raji and Jingying Yang. 2019. ABOUT ML: Annotation and benchmarking on understanding and transparency of machine learning lifecycles. *arXiv preprint arXiv:1912.06166* (2019).

David Sculley, Gary Holt, Daniel Golovin, Eugene Davydov, Todd Phillips, Dietmar Ebner, Vinay Chaudhary, Michael Young, Jean-Francois Crespo, and Dan Dennison. 2015. Hidden technical debt in machine learning systems. *Advances in neural information processing systems* 28 (2015), 2503–2511.

Y. Shibui. 2020. Machine learning system design patterns. (2020). https://github.com/mercari/ml-system-design-pattern

Sara Steegen, Francis Tuerlinckx, Andrew Gelman, and Wolf Vanpaemel. 2016. Increasing transparency through a multiverse analysis. *Perspectives on Psychological Science* 11, 5 (2016), 702–712.

Hironori Washizaki, Foutse Khomh, Yann-Gaël Guéhéneuc, Hironori Takeuchi, Naotake Natori, Takuo Doi, and Satoshi Okuda. 2022. Software Engineering Design Patterns for Machine Learning Applications. *IEEE Computer* 55, 3 (2022), 1–9.

Hironori Washizaki, Hiromu Uchida, Foutse Khomh, and Yann-Gaël Guéhéneuc. 2019. Studying software engineering patterns for designing machine learning systems. In *2019 10th International Workshop on Empirical Software Engineering in Practice (IWESEP)*. IEEE, 49–495.

Martin Wattenberg, Fernanda Viégas, and Ian Johnson. 2016. How to use t-SNE effectively. *Distill* 1, 10 (2016), e2.

Adrian Weller. 2019. Transparency: motivations and challenges. In *Explainable AI: Interpreting, Explaining and Visualizing Deep Learning*. Springer, 23–40.

Hoo Meng Wong, Sagaya Sabestinal Amalathas, and Tatana Zitkova. 2019. A prescriptive logic model for software application root cause analysis. *European Journal of Electrical Engineering and Computer Science* 3, 5 (2019).

Haruki Yokoyama. 2019. Machine learning system architectural pattern for improving operational stability. In *2019 IEEE International Conference on Software Architecture Companion (ICSA-C)*. IEEE, 267–274.

Jun Yuan, Bilal Alsallakh, Narine Kokhlikyan, Vivek Miglani, and Orion Reblitz-Richardson. 2021. Convolution Can Incur Foveation Effects. In *Beyond static papers: Rethinking how we share scientific understanding in ML-ICLR 2021 workshop*.

Barret Zoph, Golnaz Ghiasi, Tsung-Yi Lin, Yin Cui, Hanxiao Liu, Ekin D Cubuk, and Quoc V Le. 2020. Rethinking pre-training and self-training. *arXiv preprint arXiv:2006.06882* (2020).

Method Card - Traffic Light Detection

## Basic Method Information

***Name, version, and application domain(s):*** Traffic light detection based on SSD, V1, for street scene analysis.

***Method purpose and appropriate uses:*** Detect rectangular traffic-light instances in a street scene, represented by a monocular camera image. The detector can be used in offline analysis, e.g., to support street scene retrieval based on traffic criteria and in similar traffic analysis tasks.

***Method definition:*** Traffic light detection based on the SSD object detection architecture. Published in [Behrendt et al. 2017], reference implementation in https://github.com/bosch-ros-pkg/bstld.

***Example input and output:*** The Output is a list of bounding boxes representing the detected objects, along with detection score and detected traffic light color.

## Safety and Troubleshooting

***Inappropriate use and common usage pitfalls:***

- Must NOT be used in autonomous driving, it is not designed to handle all traffic scenarios.
- Should not be used with night-time scenes.

***Known weaknesses and biases:***

- Can only handle rectangular bounding boxes during training and inference.
- Biased against red traffic lights as they are very similar to vehicle tail lights.

***How to detect biases in the model internals:***

- Feed a zero input and look for activation artifacts
- Average the feature maps over random inputs.
- Is the average kernel per conv. layer symmetric?

***Common failure modes, root causes, and mitigation:***

- Red traffic lights likely undetected if in proximity of vehicles (will likely be deemed as tail lights). Consider generating training samples with such scenarios, or first applying a car detector and masking detected cars.
- Green traffic lights with vegetation background. Consider using a backbone that has higher shape bias and lower texture bias [Geirhos et al. 2019].
- Traffic lights with street ads in the background. Consider augmentation with generating data.

## Data Preparation

***Input and output format, shape, and data type:***

- The input is a 1280×840 RGB image, with integer intensity values in the range $[0, 225]$. Do not resize the image as this can compromise the aspect ratio of the traffic lights.
- The output is a ranked list of detected traffic lights. Each item in the list contains the bounding box $((x1, y1), (x2, y2))$ in a relative coordinate system $[0, 1] \times [0, 1]$, and the production confidence normalized to $[0, 1]$.

***Data transformation and normalization:*** The input image is first normalized to the range $[0, 1]$.

***Recommended sampling and balancing:***

- The frames in available datasets are captured at a constant interval. Hence, scenes at red traffic lights tend to be almost identical and over-represented. Consider similarity-based sampling.
- Yellow traffic lights are limited in available datasets and often have a larger size as they tend to be captured close to intersections. Data augmentation should focus on increasing their frequency.

***Recommended batching scheme and batch size.***

- Use a batch size of 4 or higher if the GPU allows.
- Ensure that batches contain diverse scenes instead of consecutive frames.
- Use `InstanceNormalization` for a batch size of 1 or 2 and `BatchNorm` with batch size of 4.

***Required data augmentation and shuffling.***

- Make sure to enable horizontal image flipping.
- Shuffle individual frames instead of batches.

***Validation and train-test splitting schemes:*** Ensure sufficient diversity of the different splits, especially given the sequential nature of the collected scenes. Ensure that yellow traffic lights are sufficiently present in the validation set as they are likely a minority.

Method Card - Traffic Light Detection (continued)

## Modelling and Optimization

***Architecture family and components:***

- The SSD multi-box architecture [Liu et al. 2016] with a box predictor for each traffic light color.
- The backbone CNN can be a ResNet, a MobileNet, EfficientNet or another architecture depending on available computing capacity.
- Avoid backbones that use dilation.
- Use maxpooling instead of strided convolution.

***Hyperparameters, applicable values, known impact:***

- Use mirror padding to reduce artifacts. 0-padding can create blind spots with small objects.
- Resize each input dimension to a $a \cdot 2^d + 1$ where $d$ is the number of pooling layers an $a$ is a scalar. This avoids asymmetric filters.

***Training objective(s), and optimizer(s):*** Use

- `weighted sigmoid` as classification loss and `weighted smooth L1` for localization loss.
- Hard negative miner and NMS to reduce FPs.
- RMSProp optimizer with MobileNet and `SGD with Momentum` for ResNet backbone.

***Parameter initialization:*** Use Xavier's method if training from scratch. If finetuning, use a checkpoint trained with self-supervision.

***Regularization scheme, capacity selection:*** We do NOT recommend using dropout in the box-predictor of the SSD. We recommend an L2 regularizer with a factor of 0.00004.

***Weight quantization, recommended bit depth:*** Given the small size of the target objects, we recommend 32-bit FP values to encode the weights without quantization.

***Parallelization at training and inference time:*** We recommend identical GPUs to parallelize the training, and ensuring the batches are distributed randomly across them.

***Recommended model compression techniques:*** Given the small size of the target objects, compression can potentially wash out fundamental features.

## Method Benchmarking

***Performance metric(s) and applicable threshold(s):***

- Use Average Precision (AP) to compare models overall, per light category, and per object size.
- Use the IoU overlap threshold to determine if a detected object covers the ground truth instance.
- A score threshold defines the operating point.

***Threshold selection:*** Calibrate the IoU and score thresholds based on the intended use. If using lower thresholds for higher recall, consider human reviewers to double check.

***Overfitting detection:*** Standard analysis on training and test set performance.

***Training and inference time efficiency:*** SSDs are generally fast both at training and at inference. A MobileNet backbone converges within hours on BSTLD, and can perform detection in 38$ms$ on a single-core machine.

***Available benchmarks:*** Results on BSTLD: https://github.com/bosch-ros-pkg/bstld#results

## Interpretability and Explainability

***Applicable feature attribution methods:*** `GradCAM`, `LIME` (applied to super-pixels), and window occlusion help determine scene features involved in the detection.

***How to identify influential training instances:*** Use algorithms based on comparing embeddings, such as `TracIn`, to identify proponents and opponents.

***How to identify internal concepts and features learned:*** Use `TCAV` when example scenes for street concepts are available. Use `NetDissect` when scenes with segmentation masks are available.

## Robustness

***Out-of-distribution behavior:*** The method is likely to depend on regional street features, and hence to underperform on scenes from cities outside of the training set. Also, distorted or significantly different tones of green, yellow, or red will likely be undetected.

***Detecting and mitigating data and model drifts:*** Traffic lights are constantly being updated, and hence the training data need to be regularly updated. Novel objects might resemble traffic lights, incurring new false positives.