AIMMX: Artificial Intelligence Model Metadata Extractor

Jason Tsay
jason.tsay@ibm.com
IBM Research
Yorktown Heights, New York, USA

Alan Braz
alanbraz@br.ibm.com
IBM Research
São Paulo, Brazil

Martin Hirzel
Avraham Shinnar
Todd Mummert
hirzel@us.ibm.com
shinnar@us.ibm.com
mummert@us.ibm.com
IBM Research
Yorktown Heights, New York, USA

ABSTRACT

Despite all of the power that machine learning and artificial intelligence (AI) models bring to applications, much of AI development is currently a fairly ad hoc process. Software engineering and AI development share many of the same languages and tools, but AI development as an engineering practice is still in early stages. Mining software repositories of AI models enables insight into the current state of AI development. However, much of the relevant metadata around models are not easily extractable directly from repositories and require deduction or domain knowledge. This paper presents a library called AIMMX that enables simplified AI Model Metadata eXtraction from software repositories. The extractors have five modules for extracting AI model-specific metadata: model name, associated datasets, references, AI frameworks used, and model domain. We evaluated AIMMX against 7,998 open-source models from three sources: model zoos, arXiv AI papers, and state-of-the-art AI papers. Our platform extracted metadata with 87% precision and 83% recall. As preliminary examples of how AI model metadata extraction enables studies and tools to advance engineering support for AI development, this paper presents an exploratory analysis for data and method reproducibility over the models in the evaluation dataset and a catalog tool for discovering and managing models. Our analysis suggests that while data reproducibility may be relatively poor with 42% of models in our sample citing their datasets, method reproducibility is more common at 72% of models in our sample, particularly state-of-the-art models. Our collected models are searchable in a catalog that uses existing metadata to enable advanced discovery features for efficiently finding models.

KEYWORDS

Artificial Intelligence, Machine Learning, Model Mining, Model Metadata, Model Catalog, Metadata Extraction

ACM Reference Format:
Jason Tsay, Alan Braz, Martin Hirzel, Avraham Shinnar, and Todd Mummert. 2020. AIMMX: Artificial Intelligence Model Metadata Extractor. In *17th International Conference on Mining Software Repositories (MSR '20), October 5–6, 2020, Seoul, Republic of Korea.* ACM, New York, NY, USA, 12 pages. https://doi.org/10.1145/3379597.3387448

MSR '20, October 5–6, 2020, Seoul, Republic of Korea

ACM ISBN 978-1-4503-7517-7/20/05…$15.00
https://doi.org/10.1145/3379597.3387448

1 INTRODUCTION

The combination of sufficient hardware resources, the availability of large amounts of data, and innovations in artificial intelligence (AI) models has brought about a renaissance in AI research and practice. For this paper, we define an AI *model* as all the software and data artifacts needed to define the statistical model for a given task, train the weights of the statistical model, and/or deploy the trained model weights for prediction in a service or application. Our definition of *model* includes both traditional machine learning (ML) and deep learning models. AI as an engineering practice is still in its early stages with often unpredictable and costly results (both in terms of time and quality) [18] which are often difficult to reproduce [17]. The sheer amount of possible AI approaches and algorithms [38] and recent increase in released AI frameworks [9] result in a large variety of AI models and representations. The sheer variety and lack of standardization results in models that are difficult to interact with and reason across at scale. For example, even if two models use the same AI framework, they may be in very different domains such as Vision or Natural Language Processing (NLP) or use different algorithms or datasets. Even when a model's code is available, often using or understanding this model requires much manual effort, sometimes even requiring reading associated papers. This manual effort often precludes using these models at scale. We propose that extracting standardized model metadata will reduce this manual effort and even enable programmatically analyzing or interacting with a large quantity of models.

One avenue for standardization is that software and AI development share many of the same languages and tools, such as version control systems. Existing software repository tools and services, such as GitHub, are popular with AI developers to store model definition code and development artifacts such as configurations and training logs. In fact, software repositories are popular methods of disseminating examples of models for these frameworks, such as *model zoos* that collect models for a given framework. Enterprise AI systems also commonly use versioning systems meant for software, to store both AI and non-AI components [7]. One possibility is that existing software repository mining techniques such as software analytics techniques [22] or bug prediction techniques [15, 25] can be adapted or reused for AI development. However, developing (and mining) AI models presents additional challenges over traditional

software engineering. AI development often requires managing many model-specific components that are entangled [7, 29] such as code, data, preprocessing, and hyperparameters. The tools that support software development, such as version control systems, tend to not support representing these entangled components. We expect that mining the repositories of AI models will give insight into AI development, but often information about these components is not directly accessible. For example, an image classification model often contains code that defines the model but information such as the dataset used, papers referred to, and even the domain of the model is absent or hidden in documentation.

We present a library called AIMMX (AI Model Metadata eXtractor) for simplified and standardized extraction of AI model-specific metadata from software repositories. The extractors take existing software repositories which contain AI models and aggregate data from multiple sources, such as documentation, Python code, model definition files, etc. Our extractors aggregate this data for AI model-specific metadata. Aggregation also enables further inferring additional model-specific metadata that is not easily available directly from software repositories. The extraction library contains five main modules to extract model-specific metadata: model name, references, dataset, AI frameworks, and model domain. The model domain inference module in particular uses machine learning to automatically infer a model's domain such as Computer Vision or Natural Language Processing (NLP).

In contrast to other model metadata efforts such as ONNX [5], PMML [16], and PFA [27] that focus on defining the model's low-level computational graph, our metadata extraction is more concerned with higher-level questions such as the domain or which datasets were used to train a given model or how to use a given model rather than model definition specifics such as the topology of the neural network the model uses. We evaluated our extractors by collecting 7,998 models from public software repositories from three sources: 1) 284 "model zoo" example repositories, 2) 3,409 repositories extracted from AI-related papers, and 3) 4,324 repositories associated with state-of-the-art AI models. Using a subset of this dataset, we created test sets and evaluations for each of our five extraction modules as mentioned above as well as a holistic evaluation of the entire system. The automatically extracted metadata have an average precision of 87% and recall of 83%. The evaluation dataset is available as part of the replication package. After extraction, the metadata is ready for consumption in both machine-readable and human-readable states. See Figure 1 for an overview of the extraction system, dataset collected, and preliminary usage of the extracted metadata.

Extracting metadata in a standardized way is useful for furthering engineering support for AI development. Metadata enables large-scale analysis and tools in research and practice that manage multiple varying models. We perform an exploratory analysis across our evaluation dataset for the reproducibility of AI models. Reproducibility in AI papers [17] and Jupyter Notebooks [26] tends to be relatively poor, due to a lack of documentation over method selection, datasets used, or experiments ran. We quantitatively examine our metadata dataset of 7,998 models for signals of both data (datasets used for an AI model) and method (algorithms and design decisions for an AI model) reproducibility [17]. Our exploratory analysis found that data reproducibility tends to be relatively low

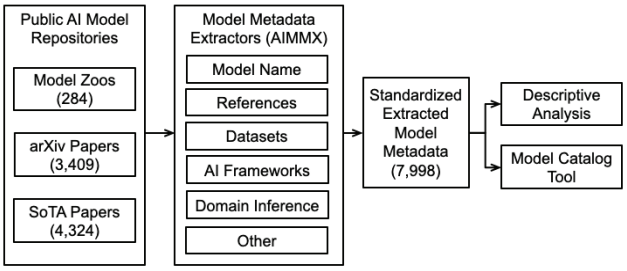

Figure 1: Overview of Extractor System.

at 42% of models in our sample having extractable information about datasets used. Method reproducibility, proxied by extracted references, is higher than data reproducibility at 72% of models in our sample, with state-of-the-art models being particularly high at 92%. As an example of a tool that leverages extracted metadata, we also describe an implementation of a searchable catalog that uses metadata to manage discovering and evaluating collected models. The system is scalable for cataloging thousands of models, allowing model producers to add their own models in a manner that imposes minimal burden due to AIMMX enabling automated metadata extraction. In contrast, other model management systems such as ModelDB [36] provide these features but require that model producers instrument their code. Using AIMMX's extracted metadata in a catalog provides automatic connections between code, datasets, and references which is similar to the manual connections in the Papers With Code website [2]. These connections may also enable automated training or deployment in future tools.

This paper makes the following contributions:

- Tool for extracting AI model-specific metadata from software repositories with currently five extraction modules (Section 2).
- Evaluation of our tools against a dataset of 7,998 models (Section 3). This AI model metadata dataset is also available as part of a replication package.
- Preliminary usage of extracted metadata via an exploratory analysis of the data and method reproducibility of AI models in our dataset and implementation of a cataloging tool (Section 4).

2 AUTOMATED MODEL METADATA EXTRACTION

The core of AIMMX is a Python library that reads software repositories, specifically from GitHub [4], and extracts AI model-related information into standardized model metadata in the JSON format that is machine and human readable. This library is open source and publicly available for use[1]. AIMMX is meant to be simple to use: once it is instantiated with a GitHub API key, then the user calls a function with a desired GitHub URL which then runs the extractors and returns the extracted metadata. The advantages of choosing to use software repositories and GitHub specifically are that they are already in common use for AI development [7]. For example, most major AI-related frameworks such as TensorFlow,

[1]https://github.com/ibm/aimmx

PyTorch, and Caffe2 have public *model zoos*, collections of example or demonstration models, hosted on GitHub. Another advantage is that software repositories often document more than just code, for example, there is a culture of rich documentation through README files that are automatically displayed on GitHub repository pages. Depending on the community, data scientists will often spend extra effort to ensure documentation is updated [33]. GitHub also has a rich Application Programming Interface (API) [14] that enables our tools to integrate with it in a straightforward manner. The extractor supports three forms of URLs: full repositories, subfolders within a repository, and individual files in repositories. For example, the TensorFlow model zoo contains multiple folders, each containing an example model whereas the Keras model zoo contains a folder with multiple Python files, each containing an example model. From the GitHub API, information such as the repository name, description, tags (*topics* in GitHub), authors (*contributors* in GitHub), open source license, primary programming language, date of last code commit, number of stargazers for the repository (a popularity metric similar to *Likes* in Facebook or Twitter [12]), and list of files are directly extractable. Then, the extractor optionally mines additional information depending on whether the repository contains certain files such as the README file, Python code, Python-specific configuration files, and certain types of ML or AI framework-related binary or configuration files. For example, Caffe2 commonly describes the expected dimensions for input data in *value_info.json*. Our tools extract this information and encode it in the metadata as an embedded JSON schema in *input_data_schema*. Specific binary files are automatically identified and placed into the *trained_model* subobject based on the file extension (e.g. `.pb` for Caffe2, `.h5` for Keras, `.onnx` for ONNX), and `Dockerfile` for containerized models.

An issue with using version control systems meant for traditional software is that AI model-specific metadata is not directly available through repositories or associated code or configuration files. However, by analyzing the aggregated metadata, model-specific metadata can be extracted or inferred. This metadata is then able to augment the aggregated metadata that is more directly extractable from software repositories, code, and configuration files. The current version of the extractors contains five such modules: model name, references, associated datasets, AI frameworks used, and model domain inference.

2.1 Model Name Extraction

The first main module attempts to extract a more descriptive name for a given model from available metadata. In many cases, the most obvious name, the repository name, is insufficient or suboptimal. Models often exist as part of subfolders or individual files within repositories, especially in "model zoo" collections which often cannot directly use the repository name. Also, the repository name is often a nickname or a non-obvious abbreviation. For example, a repository may be named "hip-mdp-public" but a more descriptive name would be "Robust and Efficient Transfer Learning with Hidden Parameter Markov Decision Processes." To extract more descriptive names, this module uses a rule-based approach to analyze documentation for potential names. Specifically, the documentation analyzed depends on the repository and what is available. If the model is in a repository subfolder, the subfolder's README file is used if available. If the model is a specific Python file, the docstring, documentation comments at the top of the file, is used if available. If the model is a repository or other files are not available, the repository-level README is analyzed. Once the documentation to analyze is determined, the README or docstring is iterated line-by-line, skipping non-relevant items commonly found at the top of README files such as CI badges, image banners, heading characters (such as *** or ===), and administrative notes such as "**NOTE: This repo...". When the first relevant line is found, then it is stripped of Markdown or HTML characters and any hyperlinks. This cleaned line is returned as a potential name. If this potential name is not found, then the repository name is used as a fallback.

2.2 Reference Extraction

We chose to implement a module to extract references to papers because in preliminary user testing, data scientists tend to discuss models in terms of corresponding academic papers. This module uses three rule-based approaches to extract references: 1) regular expressions to search for common reference formats, 2) search for arXiv IDs with correspond lookups to the arXiv API, and 3) identify and import code blocks containing BibTeX references. The first approach attempts to find a variety of references that may include various conferences or even blog posts while the second and third approaches attempt to find specific formats that are popular with machine learning papers. For all three approaches, the module searches across README files and docstrings using the same rules as the model name module. In the case of overlapping references found by multiple approaches, the reference with the most metadata as measured by fields extracted is kept with a preference for the arXiv and BibTeX approaches over the pattern-matching approach.

The first approach uses nine regular expression patterns to find both references to academic papers and links to blog posts and other webpages. The patterns were developed by examining existing references in documentation for repositories in model zoos. The metadata returned for this approach varies depending on the pattern. The simplest example is a blog post which returns only the article title and the URL while a more complicated pattern may return the title, list of authors, year, arXiv ID, and URL. This approach is the broadest in terms of what types of references are allowed, as any conference, journal, or blog post is potentially valid. However, the pattern-based approach is quite limited in that only references that match the patterns defined will be matched.

The second approach searches for arXiv papers. ArXiv is a preprint hosting service particularly popular with academics in AI fields [3]. Specifically, links to arXiv papers are searched for within the given README and then the arXiv ID is extracted from the link. The ID is then looked up against the arXiv API [1] for additional information such as the article title, authors, and publishing date. The advantage of this approach is that arXiv is very popular amongst machine learning researchers and is commonly used. Using the arXiv API also allows for extracting reference information in a standardized way that is robust to differing citation styles. The disadvantage of using arXiv is that its references tend to be preprints and publishing conference or journal information is often lost or unavailable.

The third approach searches for code blocks within the documentation for BibTeX references. This particular approach relies on searching for code blocks as defined by the Markdown language that GitHub uses for README files. The entire code block must be a valid BibTeX reference (it cannot contain anything except BibTex). Multiple entries in the code block are allowed. Usage of BibTeX seems to be particularly popular to provide a citation to a model repository's associated paper. The advantage of this approach is that BibTeX is a well-established and precise format.

2.3 Dataset Extraction

Data management is a hard challenge in engineering AI systems [7, 38] and models in software repositories often have no formal descriptions of datasets used. Our module attempts to automatically extract and link models to the datasets used. For this version, the module extracts the name of the dataset and potentially a link to the dataset. The module uses two rule-based approaches: searching for links in the README and searching for references to common datasets. The first approach allows for finding arbitrary datasets and the second approach allows for finding commonly used datasets in machine learning papers. For the first link-based approach, the README is searched for links that contain dataset-related keywords, specifically "dataset", "data", and "corpus." The names and then referenced URLs of the extracted dataset is returned. The second approach uses a set of 640 common dataset names and searches for mentions to these datasets in the README. To avoid partial matching of short dataset names such as "MNIST" versus "Fashion-MNIST," matching datasets must be their own token(s) and surrounded by whitespace or punctuation. If this approach finds a match, then only the dataset name is returned. For cases where both approaches return the same dataset, such as the "New York Times Corpus," the extracted metadata is merged by combining the name and link. This module follows the same rules to the model name module in determining which documentation file to analyze.

The list of common datasets was extracted using the Papers With Code website [2] which compiles machine learning papers and repositories and metadata that links the two. In the Papers With Code data[2], there are common machine learning tasks such as Language Modeling and Semantic Segmentation. For each task, there is a list of datasets and a leaderboard for each dataset with associated papers and associated code repositories for each paper. For example, the Language Modeling task includes the One Billion Word dataset [11]. The module collected each of the datasets for each of the tasks (as of 8/20/2019), resulting in 640 total dataset names that the module searches for in the README. Some dataset names were removed to prevent false positives such as "Datasets." Since the datasets are known, future work should add additional metadata for matched datasets. For example, if "MNIST" is matched, then metadata such where the dataset is available and the schema could also be made available.

2.4 AI Framework Extraction

AI frameworks play an important part towards enabling the model development process. Recent years have seen a spike in the release and adoption of AI frameworks [9] and framework-related questions are a major category of machine learning-related topics on Stack Overflow [8]. Our module identifies which AI frameworks a particular model uses by searching the source code. We focus on Python AI frameworks as they are the most popular [9]. The module then concatenates all Python (.py) and code cells of Jupyter Notebooks (.ipynb) into a single text string. Once all the code is extracted and merged into a single string, a regular expression is used to find the name of the modules imported, specifically cases of 'import module_name' and 'from module_name import function_name' and all its variations (like with 'as nickname', multiple modules at the same line, or functions from submodules). The found module names are then filtered by a fixed list of well-known frameworks such as *Caffe, Keras, Lasagne, MXNet, NLTK, PyTorch (or torch), TensorFlow, Theano, scikit-learn (or sklearn)*. The only exception is the *Caffe2* framework which is not a Python module. Therefore, we check the coexistence of the files: *init_net.pb* and *predict_net.pb*, and if so, its name is added to the frameworks list. A full list of AI frameworks for extraction is in Table 6.

2.5 Automated Domain Inference

This module uses machine learning to infer the domain of a given model based on its available metadata. Here *domain* refers to the genre or type of activity that the model is associated with, for example: Computer Vision, Natural Language Processing (NLP), etc. A general issue with extracting model metadata is that often the domain of a model is not explicitly defined. However, machine learning practitioners often naturally describe models by their domain. We use machine learning on a public dataset of model repositories to create a machine learning model that takes in model metadata as input, and outputs the model's inferred domain and task along with a confidence score.

To create the domain inference model, we created a training and validation dataset of repositories and their associated *domain* and *task* using data from the Papers With Code website [2]. In this case, *domain* is a more general category for models whereas *task* is a more specific activity within the category. Given the previous example in the datasets extractor module, in Papers With Code, Natural Language Processing (NLP) is a domain and Language Modeling is a task within that domain. We use data from Papers With Code because it provides ground truth for the domains and tasks for model repositories which is often unavailable otherwise. We use a total of 2,915 repositories labeled with domains and tasks from Papers With Code along with 300 repositories written in Python that have nothing to do with machine learning as negative examples for a total of 3,215. These negative examples were manually gathered from GitHub's most popular Python repositories. This dataset is then split into training and validation sets with 70% or 2,237 repositories in the training set and 978 in the test set. For the current version of this module, we take a bag-of-words approach with the input model metadata. Specifically, only the README is considered in the domain inference model but it is stripped of all tags and special Markdown characters and then tokenized and vectorized.

Through examining the dataset and empirically, we settled on an ensemble of support vector classification models that work in a two-stage process as seen in Figure 2. The first stage determines

[2] At the time of publishing, their data is available under the CC BY-SA license.

if a given model's domain is Computer Vision, Natural Language Processing (NLP), Other, or Unknown (not a model). Depending on the results of the first stage, the given model is then fed into one of three multiclassification models: 1) Computer Vision tasks, 2) NLP tasks, or 3) Other domains. The result of the ensemble is a domain and task, or in the case of Other domains, just the domain, along with a confidence score. For example, Model A may be determined to fall under the Computer Vision domain in the first stage and is then fed into the Computer Vision task model and has Object Detection as the task with a confidence of 0.68. Model B may be determined to fall under Other domain in the first stage and then is determined to be in the Medical domain in the second stage with a confidence of 0.72. The Computer Vision, NLP, and Other domain split was done due to the unbalanced nature of the ground truth distribution of the dataset. Out of 2,915 labeled model repositories, 1,654 (56.7%) are Computer Vision and 824 (28.3%) are NLP. The other domains make up 15% of the dataset with Playing Games the largest at 171 (5.9%). Additionally, 300 non-model software repositories written in Python were added to the dataset and labeled as "Unknown" to give negative examples. The full list of domains and tasks inferred is available as Appendix B.

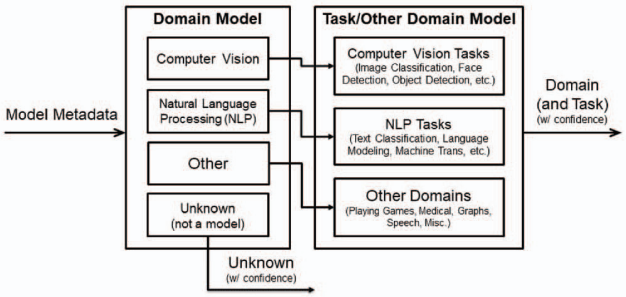

Figure 2: Domain inference machine learning model ensemble diagram.

3 EVALUATION AND PRELIMINARY ANALYSIS

We evaluate our automated AI model metadata extractors through a dataset of 7,998 public models from open source software repositories. We perform individual evaluations for each of our five model-specific metadata extraction modules. Each of the module evaluations uses its own methodology and subset of the collected dataset. We also manually evaluate the system as a whole with a subset of the dataset. The evaluation dataset and each module evaluation data subset are available as part of the replication package[3].

3.1 Evaluation Dataset

To evaluate our extractors, we collected a dataset of public AI model software repositories on GitHub. The challenge was to identify repositories on GitHub that contain AI models rather than just being AI-related. For our dataset, a repository was considered to contain an AI model if it contains artifacts to define and/or train a model with data, or the resulting artifacts of the training process. For example, AI-related frameworks, purely data, or documentation repositories do not count. To solve this challenge, we gathered repositories associated with AI models from three sources: 1) 284 "model zoo" example repositories, 2) 3,409 repositories extracted from AI-related papers on arXiv [3], and 3) 4,324 repositories associated with state-of-the-art AI models [2] (19 models overlap from multiple sources). The dataset is summarized in Table 1.

Model zoos are good candidates for evaluation because these repositories tend to be well-documented and maintained. We gather 284 models from six model zoos of popular AI frameworks: TensorFlow, Caffe2, Keras, PyTorch, MXNet, and the Model Asset Exchange. In this case, the six model zoos are either a single GitHub repository with multiple folders each containing a model or a collection of multiple GitHub repositories. We expand our dataset by collecting software repositories that are associated with AI-related papers, assuming that these repositories are more likely to contain AI models. From over 41,000 academic papers on a dataset of AI-related *arXiv* [3] papers published on the Kaggle competition service [31], we gathered 3,409 repositories by using bulk paper access to search the papers for GitHub links. After processing the extracted GitHub links to ensure uniqueness and removing malformed or irrelevant links (e.g. links to GitHub itself rather than repositories), the dataset contained 3,938 links. After attempting to extract metadata from this dataset and removing inaccessible and dead repositories, the arXiv dataset contains 3,409 repositories. Additionally, whereas the model zoo dataset mostly uses deep learning, the arXiv dataset contains both deep learning and traditional machine learning models. Lastly, we gather 4,324 repositories associated with state-of-the-art (*SotA*) AI papers using the curated Papers With Code website [2]. The website lists various machine learning tasks with associated datasets and leaderboards of the performance of AI papers for these datasets. One or more software repositories are linked with each of these papers. We used data from this website to train our domain inference module because it contains model-related metadata that is curated and annotated. We also use this labeled metadata to evaluate some of our modules. Since the metadata extracted by our model-specific modules are not directly available, evaluating these modules often requires manual labeling which is available through their public data. The results of the evaluations are summarized in Table 2.

3.2 Model Name Extraction

To evaluate the model name extraction module, we created a test set that is a random sample of 400 repositories or 5% of the collected dataset of 7,998. Due to the nature of the model name extractor, there is a lack of ground truth for model names in the dataset which requires a manual evaluation. Model names in particular are difficult to evaluate automatically because it is possible for multiple model names to be descriptive or correct for a given model. For this evaluation, one of the researchers manually examined the extracted names in the test set. A name is considered correct if it is more descriptive than the default repository name. For example, "entropy-sgd" versus "Entropy-SGD: Biasing Gradient Descent Into Wide Valleys." The name must also not include any formatting characters such as "###Model Name." If the extracted name matches the default repository name, it is considered incorrect unless the

[3]https://zenodo.org/record/3609308

Table 1: Evaluation dataset summary.

Model Zoo	No. Models	URL
TensorFlow Models	73	https://github.com/tensorflow/models
Caffe2 Model Repository	87	https://github.com/caffe2/models
PyTorch Examples	12	https://github.com/pytorch/examples
Keras examples directory	42	https://github.com/keras-team/keras/tree/master/examples
MXNet examples directory	38	https://github.com/apache/incubator-mxnet/tree/master/example
Model Asset Exchange	32	https://developer.ibm.com/code/exchanges/models/
Model Zoo Dataset	284	
arXiv Paper Dataset	3,409	
SotA Paper Dataset	4,324	
Total	7,998	(19 overlap)

Table 2: Evaluation results summary.

Evaluation	Count	Metric	Value
Model Name	400	Correctness	0.853
Reference	4,094	Precision	0.655
Dataset	160	F1	0.757
Framework	252	Precision	1.000
Domain Inference	978	Domain Accuracy	0.859
		Task Accuracy	0.723
System	80	Precision	0.872
		Recall	0.833

default repository name is the full name of a model or approach. For example, "BERT" is correct for the BERT model [13]. The percentage correct of the test set was 85.3% or 341 of 400 repositories.

3.3 Reference Extraction

To evaluate the reference extraction module, we created a test set with 4,094 pairs of paper references and model software repositories. For this evaluation, we needed repositories with known connections to references. We made use of the *SotA* dataset described earlier from Papers With Code [2] as it links together paper references with software repositories. We assume that the link should also work in reverse: each AI model software repository should be associated with its paper. Papers in the test set may be associated with multiple repositories and repositories may be associated with multiple papers. For the evaluation, we measure the precision of our reference extraction module. We chose to use precision due to the direction of the labeled data available. Whereas our extraction has a one-to-many relationship between repositories and references, the labeled data has a one-to-many relationship between references and repositories. To reconcile the two, we identify pairs of references and repositories and examine if the extracted metadata for the repository contains the associated reference. Specifically, we count the pair as correct if the title of the reference in the test set matches one of the references in the extracted model metadata for the repository. The precision of our evaluation was that 2,682 or 65.5% of the pairs in test set were correct.

3.4 Dataset Extraction

To evaluate the dataset extraction module, we created a test set that is a random sample of 160 repositories out of the collected dataset of 7,998. We performed a manual evaluation because we lacked ground truth for datasets associated with models. One of the researchers manually examined each of the repositories in the random sample to create a ground truth dataset of available datasets for each repository. The researcher had access to the same documentation artifacts that the dataset extractor had access to: the README file in most cases or the docstring if the model is a single Python file. Using that documentation, the researcher had to determine which datasets the model used to either train or evaluate the model. For example, a given image classification model may use "ImageNet" to train the model and evaluate the model on "CIFAR-10." For each repository in the sample, we then compare the names of extracted datasets to the manually created ground truth set. The precision of our evaluation was 76.91%, the recall was 75.99%, and the F1 score was 75.75%. In further inspection of the evaluation sample, 86 or 53.8% of the repositories had no extracted datasets with the F1 score of this subsample at 80.2%. In the 74 (46.2%) repositories with extracted datasets, the F1 score was 70.5%.

Table 3: Frameworks extracted from model zoos models.

Model zoo	Count	Framework(s)
Caffe2	87	Caffe2
Keras	42	Keras, TensorFlow, Theano, scikit-learn
MXNet	38	MXNet, Keras, Caffe, PyTorch, scikit-learn
PyTorch	12	PyTorch
TensorFlow	73	TensorFlow, Keras, NLTK, scikit-learn

3.5 Framework Extraction

To evaluate the framework extraction module, we use 284 models from "model zoos" as ground truth as most model zoos are associated with a particular deep learning Python framework as seen in Table 1. The precision of the module can be assessed by whether the AI frameworks extracted from models match the framework the zoo is associated with. For example, a model from the TensorFlow zoo should have the TensorFlow framework in its extracted metadata. A total of 252 models are from these framework-associated model zoos which are summarized in Table 3 along with all of the extracted frameworks. For all cases we see that the expected framework is extracted for a precision of 100%.

3.6 Automated Domain Inference

To train the domain inference module, we created a training dataset from a subset of the *SotA* dataset along with non-model software repositories. Specifically, from the 3,215 repositories labeled with domain information, 30% or 978 were reserved for a test set. Each

Table 4: Domain inference evaluation result summary with breakdown by domain.

Dataset	Size	Domain Accuracy	Task Accuracy
Test Set	978	0.859	0.723
Computer Vision	502	0.940	0.785
NLP	252	0.802	0.583
Other	134	0.597	0.597
Unknown	90	0.956	

of the repositories in the test set were labeled with a domain consisting of: Computer Vision, Natural Language Processing (NLP), Other, or Unknown (not a model). Repositories labeled with Computer Vision or NLP domains are also labeled with an associated task. Repositories labeled with Other domain are also labeled with a more specific domain such as Medical, Playing Games, etc. For the evaluation, we determine the accuracy for both the domain stage and the task/other domain stage of the domain inference ensemble. As Unknown domain models do not go to the task/other domain stage, they are not included in the accuracy calculation for that stage. The domain stage accuracy for the test set is 0.859 and the task stage accuracy for the test set is 0.723. We break down the results by domains in Table 4 and note that the domain stage performs better than the task/other domain stage. Similarly, Computer Vision performs better than NLP which performs better than Other domains, perhaps due to having more examples in the training set. We also note that our module performs very well at discriminating between models and not-models (Unknown) at 0.956, suggesting perhaps future usage of the domain inference module to automatically determine if a given software repository is an AI model.

3.7 System Evaluation

To evaluate the entire extraction system holistically, we manually evaluated extracted metadata for a random sample of 80 repositories of the collected dataset of 7,998. We first manually created a ground truth dataset from this sample. The researcher who created the ground truth dataset had access to the same sources as the automated extraction: GitHub repository, README files, and Python code. Using domain knowledge, the researcher manually annotated the extracted model metadata sample by comparing to this ground truth dataset, listing two cases of errors: properties that are present but incorrect and properties that are missing. For example, the automated extractor may extract three properties from a model: name is “MNIST model”, dataset is “MNIST”, and the model has three authors: A, B, and C. The ground truth dataset may then note that the authors list is actually A, B, and D and that the README file also has references to two papers. In this case, there are two errors: 1 property (authors list) is incorrect and 1 property is missing (references). As the previous example demonstrates, properties that are lists are counted as one property as it gives a more conservative indication of the performance of the extraction. We then calculate precision and recall for our sample based on the number of correct and missing extracted properties.

For the system evaluation, the researcher additionally had to determine whether the repository was actually an AI model using the criteria described earlier. Out of the original 80 sampled repositories from the paper dataset, 66 (82.5%) of the repositories actually contained models. For this evaluation, the documentation of the model also needed to be in English. Sixteen ineligible repositories (14 non-models, 2 non-English) were removed and iteratively replaced with random samples from the dataset of 7,998 until 80 eligible total model repositories were collected. The system evaluation was performed on this sample.

The precision of our system evaluation was 87.17%, the recall was 83.34%, and the F1 score was 85.14%. Upon further inspection of the evaluation sample, if the extracted properties were restricted to only what was returned by the five extraction modules described earlier, then the precision drops to 70.73%, the recall to 66.83%, and the F1 score to 68.48%.

4 PRELIMINARY METADATA USAGE

Automatically extracting standardized AI model metadata enables quantitative analysis and tool support across a wide set of AI models. We use our evaluation dataset of 7,998 models in both an exploratory analysis of model reproducibility and in an example catalog tool.

4.1 Exploratory Reproducibility Analysis

We demonstrate the potential of the extracted metadata by quantitatively analyzing the evaluation dataset for AI model data and method reproducibility. AI research papers tend to be poorly documented for reproducibility [17]. Borrowing terminology from Gundersen et al. [17], we examine two types of reproducibility for AI models in our evaluation dataset: *data* and *method* reproducibility. Data reproducibility is the data used in AI experiments whereas method reproducibility are the algorithms used and decisions behind algorithm selection. We examine extracted datasets to explore data reproducibility in our models and extracted references to explore method reproducibility. Our analysis is exploratory because we do not attempt to manually reproduce AI models (such as in [17, 26]) but rather quantitatively analyze a larger-scale dataset for signals of reproducibility based on literature.

We first report descriptive statistics for the repositories in the dataset which are summarized in Table 5. We split the statistics by source of the repositories as described in the previous section: “model zoos”, from arXiv [3] papers, and state-of-the-art AI models [2] (with 19 models that overlap). We report the median Stars of repositories, the percentage of repositories that primarily use Python (includes Jupyter Notebooks which tend to be popular with data scientists), repositories with README files (which our extractors use as a source of information), repositories with inferred domains (cannot be "Unknown"), at least one extracted reference, at least one extracted dataset, and at least one extracted AI-related framework. We note that most (72%) models in the dataset contain at least one extracted reference, supporting a suggestion from preliminary user testing that data scientists tend to discuss models in terms of papers. We also note that the high level of extracted AI frameworks is a positive sign for reproducibility, as knowing the module dependencies in Jupyter notebooks also promoted reproducibility [26] (a distribution of usage is available in Table 6).

For data reproducibility, we explore extracted datasets in model metadata as a signal for documentation of datasets used in AI models. Compared to traditional software engineering, the success of

Table 5: Evaluation dataset descriptives.

Attribute	Overall	Model Zoo Dataset	arXiv Dataset	SotA Dataset
Median Stars	12	17513	34	2
Uses Python	74%	96%	55%	87%
Has README	99%	100%	98%	100%
Domain Inferred	70%	45%	46%	90%
References Found	72%	43%	49%	92%
Dataset Found	42%	51%	31%	49%
AI Framework Found	98%	100%	96%	100%
Count	7998	284	3409	4324

Table 6: AI frameworks extracted with usage by repository.

AI Framework	Repository Count
Caffe	415
Caffe2	113
Keras	1056
Lasagne	115
MXNet	164
NLTK	455
PyTorch	1744
TensorFlow	2556
Theano	411
scikit-learn	1139

AI models tends to be highly tied to data used and its processing [10, 38]. In enterprise settings, this reliance on quality data for success means that sharing and reusing datasets is vitally important [7]. We use extracted datasets to explore the degree to which types of models have documentation regarding datasets. Table 5 shows that 42% of models in our sample have an extracted dataset with state-of-the-art models having a higher rate of having an extracted dataset at 49% and arXiv models at a lower rate at 31%. When we split the models by domain (with "Unknown" domain models removed), there is a noticeable increase in models with datasets, particularly for the popular domains of Computer Vision (53%) and Natural Language Processing (49%). The domain split is summarized in Table 7. We note that a disproportionately small amount of datasets tend to be used by most models, as the distribution of datasets to repositories in our sample is highly skewed (skewness 6.07) with each dataset having an average of 26.0 repositories but a median of 4.0. As a limitation in our current extractor, we are not able to automatically determine if the dataset extracted from an AI model is used for training, validation, or testing. Our findings are in line with Gundersen et al.'s study with a similar rate of dataset sharing (49% vs 42%) [17].

For method reproducibility, we explore extracted references in model metadata as a signal for documentation of algorithm selection and design choices. We again borrow terminology from Gundersen et al. to distinguish between *AI program* and *AI method* where the *method* is the conceptual idea that the *program* implements. In this case, we consider the software repository as the *program* and papers referred to as describing the *method*. In particular, method reproducibility also considers design decisions because often AI development is much more flexible than traditional software development, with tens to hundreds of candidates to be considered [38]. From our descriptives in Table 5, we see that 72% of models in our sample have at least one reference extracted with state-of-the-art models having a much higher rate of 92% whereas arXiv models are much lower at 49%. When we split the models by domain (Table 7), we note that our known domains have higher rates of having references, such as Vision with 86% and NLP with 80%. Similar to datasets, a small amount of references also tend to be used by most models. The distribution of references to repositories in our sample is also highly skewed (skewness 15.12) with each reference having an average of 2.1 repositories with a median of 1.0.

Table 7: Repositories with datasets or references by domain ("Unknown" is excluded).

Domain	Count	Datasets	References
Computer Vision	3537	53%	86%
NLP	1484	49%	80%
Playing Games	245	29%	85%
Medical	129	23%	88%
Graphs	102	65%	85%
Speech	51	22%	84%
Misc	27	59%	89%
Total	7998	42%	72%

Our findings suggest that both state-of-the-art models and particular domains tend to have more documentation that supports reproducibility. The concentration of references to particular datasets and papers suggests that there may be low-hanging fruit in better supporting these popular approaches and datasets. For example, future work for AIMMX may identity that a popular dataset such as MNIST is used and provide meta-features of the dataset such as size and number of classes.

4.2 Model Catalog Tool

As an example of a tool that is able to leverage extracted metadata, we implemented a catalog web application for the discovery and evaluation of AI models. The catalog application consists of two main views: a list of models with filter and search features (see Figure 3) and a page that displays individual model details (see Figure 4). Models in this catalog are added through providing GitHub repository URLs which are then passed to AIMMX for metadata extraction. The metadata are then inserted into the catalog's document database, validated automatically, and then made available for discovery. The system is available as an online service[4].

The model list view is the main page for discovering models and contains summary information for each model such as name, stars, domain, frameworks used, and lifecycle stages. While the model list itself is browsable by users, the main method of discovering models is through the search and filter features, which allow for querying or selecting multiple attributes that are based on properties in extracted model metadata. The model list view contains a side panel with metadata attributes for filtering such as domain, frameworks, and tags. Multiple attributes may be selected, enabling the discovery of more specific models, for example, to find Computer Vision-related TensorFlow models, the filter of "Computer Vision" in Domain and the "TensorFlow" filter in Framework would be selected. The top of the model list view contains a search feature

[4]https://ai-model-catalog-msr.us-south.cf.appdomain.cloud/

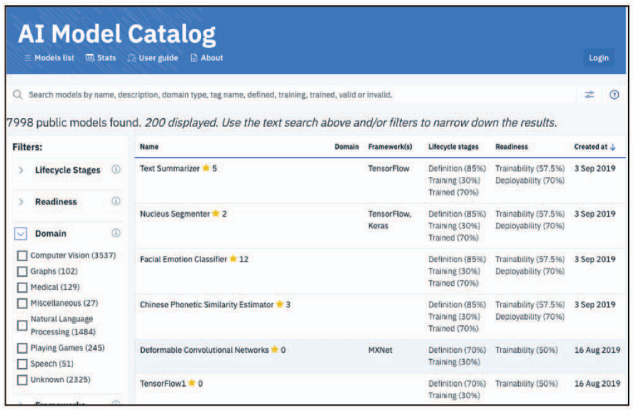

Figure 3: Catalog list of available models, search bar, filters.

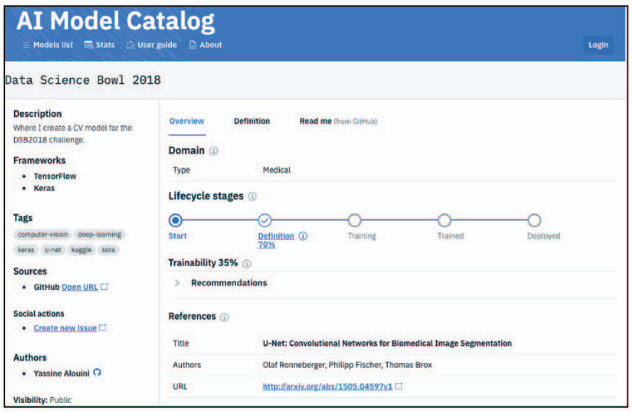

Figure 4: Catalog individual model detail page.

which composes metadata together and also indexes each field as a searchable attribute directly in the CouchDB database. We index searchable metadata in two ways which enables two methods of searching: by general keywords or by filtering for specific attributes. In the case of general keywords, searchable attributes are aggregated into a single string and then tokenized. This approach then allows for users to simply type any keyword into the search box to search by any of these attributes. For example, typing `resnet` into the search box will return models with "ResNet" in the name, description, references, and so on. Specific attribute filters are also searchable, similar to the filter feature but including additional attributes such as name, description, and reference.

Once a user selects a model, the individual model detail view enables users to assess a model for reuse. Details shown include information such as tags, extraction source, authors, and license. The signals for data and method reproducibility such as extracted datasets and paper references are also visualized on this page. We display the model's applicable lifecycle stages ("definition" or pre-training, training, and trained) as well as a percentage indicator of how much information is available for a given lifecycle stage for the selected model. For each lifecycle stage that is applicable for a model, a tab is available that displays additional lifecycle-specific information. for example, we display information about code artifacts in pre-training, datasets in training, and trained model weights in trained models. Users can also update specific fields such as name, description, tags, and domain in the imported model should the automated mining be inaccurate or incomplete. If the linked repository changes, the model can be re-imported, which passes the repository back to the automated extractors with an option to preserve any manually edited fields. The model detail view also contains recommendations for steps needed to train or deploy a given model. For example, a model may be missing the *datasets* property in its metadata and the corresponding suggestion to increase trainability is to "add datasets for training."

5 THREATS TO VALIDITY

External Validity. Despite our attempts to design AIMMX as generally as possible by not requiring the use of specific ML or AI frameworks [23, 30] or user intervention [35, 36], the current version only supports software repositories in GitHub. Mining GitHub has its own potential challenges [19] but we mitigate many of them in our evaluation dataset through not focusing on commits, pull requests, nor the social network aspects of GitHub in our extraction. Additionally, although AIMMX should work on most repositories, many of our features and evaluation repositories are biased towards Python. Although Python is popular with data scientists, future work should examine differences in models that use other popular data science-related languages such as R, Java, Matlab, or Julia.

Internal Validity. During the development of the extractors, we used model zoos as canonical examples (see Table 1). The model zoos chosen are popular but only contain deep learning models. That means that our system may have unforeseen biases or errors against traditional machine learning models. However, this is mitigated by our evaluation dataset that contains at least 1,139 traditional machine learning models as well (based on the usage of the scikit-learn framework). Similarly, our selection of models from arXiv and SotA papers may have introduced biases towards research code rather than enterprise or application code. We attempt to mitigate this by also including model zoos in our dataset. Another potential issue is that some of our evaluations relied on the domain knowledge of one of the researchers to qualitatively determine the ground truth of extracted metadata, specifically for the name, dataset, and system evaluations. With any qualitative analysis, there is the chance of subjectivity in the evaluation of incorrect or missing metadata properties. Also, a single researcher performed the analysis whereas using multiple researchers and measuring agreement would have been more robust. Regarding the exploratory analysis presented, we determined that the evaluation dataset contains some erroneous repositories that do not actually contain models. Although these repositories were not included in the extraction evaluation, they were included in the exploratory analysis due to lacking a scalable method of identifying and filtering out these repositories. Future work should implement a classifier to identify whether a repository contains AI model information.

Construct Validity. Our extractor evaluation focused on the precision and recall in the context of what is possible using its current features. Due to the extractors not analyzing the model Python code itself except for the docstrings, the evaluation also did not make use of it. Our extractors then are missing theoretically possible model

metadata, for example, perhaps inferring input and/or output data schemas or hyperparameters from provided Python code. For the exploratory reproducibility analysis, we have not evaluated our findings by manually reproducing AI models.

6 RELATED WORK

6.1 Software and AI Development

Artificial intelligence (AI) development as an engineering practice has many intersections with software engineering practices, including mining software repositories. Kim et al. [20] interviewed the emerging role of data scientists on software development teams, identifying five working styles that data scientists take on in these teams, from insight providers, to team leads, and more relevant to our work, model-building specialists whose models get integrated into software applications. Bangash et al. [8] identified machine learning-related questions asked on Stack Overflow, finding that questions fell into broad categories of: framework, implementation, sub-domain, and algorithms. Our work also infers information related to these broad categories, such as the AI framework, code artifacts, domain, and paper references for each models. It is our hope that our extracted metadata enables similar quantitative analyses across AI models rather than Stack Overflow questions. Software engineering concepts have also been applied to machine learning (ML) and AI systems, such as work by Sculley et al. [29] examining hidden technical debt in real-world ML systems. Relevant to our work, they highlight the importance of strong abstractions for ML systems and managing ML-specific artifacts such as datasets and configurations. Amershi et al. [7] identify through interviews fundamental differences between ML and non-ML software development: the complexity of dealing with data, model customization and reuse require unique skills, and components are difficult to modularize due to often being "entangled." Our work is motivated by the insight that these important entangled components such as datasets are often not directly observable (echoed in other papers such as [10, 38]) from software repositories. Wan et al. [38] also use interviews to focus on the differences between ML and non-ML in many phases of software development such as requirements, design, and testing. They find that the reliance on data and inherent uncertainty in the development process create unique challenges for ML systems. Our work assists in documenting some of the important ML-specific choices made in the development process such as dataset and method selection. Our work also builds upon existing work on reproducibility in both software and AI development. Pimentel et al. [26] quantitatively studied the reproducibility of Jupyter notebooks which are popular with data scientists. Relevant to our work, they found that the most common causes of failure to reproduce were missing dependencies, hidden states, and data accessibility. Gundersen et al. [17] found that AI research papers tend to be poorly documented for method, data, and experiment reproducibility. We borrow the concepts of AI method and data reproducibility for our exploratory reproducibility study.

6.2 Model Metadata Mining and Inference

Machine learning has had a close and long relationship with data mining [39], so it is natural that data mining techniques are applied to machine learning and AI models to analyze and enhance them. Sethi et al. [30] extracted network topologies from certain diagrams in academic papers about deep learning models. Vaziri et al. [37] extracted conversational agents from web API specifications. Machine learning experiment management tools [34, 36] often semi-automatically extract model metadata by requiring users to instrument their model code with framework-specific instrumentation libraries. Our software repository-based approach is also similar to the experiment tracking MLFlow service [6]. However, AIMMX is concerned more with extracting high-level contextual information to reuse models such as papers, datasets, and domains rather than automatically tracking the outcomes of experiments. There are also many examples of machine learning being applied to mining, such as automatic classification of software artifacts [21]. Projects like ML-Schema [28], an ontology for machine learning algorithms, datasets, and experiments, have identified a gap of the lack of interoperability between machine learning platforms. Our solution was to extract standardized model metadata that focuses on a high-level and contextual view of AI models. This is in contrast to similar efforts such as ONNX [5], PMML [16], or PFA [27] which focus on specifically defining the model's computational network. For example, for the same model, our metadata would describe the domain of the model, references to relevant papers (e.g. [32]), descriptions about where and what the model code and definitions are (which may be ONNX, PMML, PFA, etc.), and descriptions of where and what the training dataset is. A network definition representation of the same model would describe in detail the neural network layers and its parameters. In this way, the metadata we extract is complementary with other network representation formats.

6.3 Model Catalogs

Related work has also identified a need to catalog and manage AI models and their associated pipelines and artifacts. The catalog tool in our tool suite is a type of model management tool: it stores, tracks, and indexes AI models. A similar tool of this type is ModelDB [36] which automatically tracks Scikit-learn, Spark, and R models by instrumenting code and allows users to view and compare models. A similar system with a different scope is ModelHub [23] which focuses on managing results and versions of deep learning models. Their system includes a discovery system with a model comparison and ranking feature [24]. In contrast, OpenML [35] focuses on cataloging datasets and machine learning tasks with the intention of promoting collaboration between data scientists. We also note that every major deep learning framework has at least one model zoo, a collection or catalog of example models (Table 1). The automatic connections between domain, references, datasets, and repositories in our extracted metadata is similar to the manual connections made in the Papers With Code website [2]. We also use this website as a source of ground truth data for our domain inference model.

7 CONCLUSIONS

This paper describes AIMMX which we intend as a step towards furthering engineering support for AI development through providing standardized metadata for existing AI models. We envision that generating analyzable metadata for disparate models is both the first step towards managing models at scale and adapting existing mining software repositories techniques to AI models.

REFERENCES

[1] [n.d.]. arXiv.org help - arXiv API. https://arxiv.org/help/api/index Accessed: 2020-03-13.

[2] [n.d.]. Papers With Code: the latest in machine learning. https://paperswithcode.com Accessed: 2020-03-13.

[3] 1991. arXiv.org e-Print archive. https://arxiv.org/ Accessed: 2020-03-13.

[4] 2008. The world's leading software development platform - GitHub. https://github.com/ Accessed: 2020-03-13.

[5] 2017. ONNX. https://onnx.ai/ Accessed: 2020-03-13.

[6] 2019. MLFlow - A platform for the machine learning lifecycle. https://mlflow.org/ Accessed: 2020-03-13.

[7] Saleema Amershi, Andrew Begel, Christian Bird, Robert DeLine, Harald Gall, Ece Kamar, Nachiappan Nagappan, Besmira Nushi, and Thomas Zimmermann. 2019. Software Engineering for Machine Learning: A Case Study. In *International Conference on Software Engineering: Software Engineering in Practice (ICSE-SEIP)*. 291–300. https://doi.org/10.1109/ICSE-SEIP.2019.00042

[8] Abdul Ali Bangash, Hareem Sahar, Shaiful Chowdhury, Alexander William Wong, Abram Hindle, and Karim Ali. 2019. What Do Developers Know about Machine Learning: A Study of ML Discussions on StackOverflow. In *Conference on Mining Software Repositories (MSR)*. 260–264. https://doi.org/10.1109/MSR.2019.00052

[9] H Ben Braiek, F Khomh, and B Adams. 2018. The Open-Closed Principle of Modern Machine Learning Frameworks. In *Conference on Mining Software Repositories (MSR)*. 353–363.

[10] Eric Breck, Neoklis Polyzotis, Sudip Roy, Steven Euijong Whang, and Martin Zinkevich. 2019. Data Validation for Machine Learning. In *Conference on Systems and Machine Learning (SysML)*.

[11] Ciprian Chelba, Tomas Mikolov, Mike Schuster, Qi Ge, Thorsten Brants, and Phillipp Koehn. 2013. One Billion Word Benchmark for Measuring Progress in Statistical Language Modeling. *CoRR* abs/1312.3005 (2013). arXiv:1312.3005 http://arxiv.org/abs/1312.3005

[12] Laura Dabbish, Colleen Stuart, Jason Tsay, and Jim Herbsleb. 2012. Social coding in GitHub: transparency and collaboration in an open software repository. In *Conference on Computer Supported Cooperative Work (CSCW)*. 1277–1286. https://doi.org/10.1145/2145204.2145396

[13] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. 2018. BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding. *arXiv preprint arXiv:1810.04805* (2018).

[14] GitHub. 2016. GitHub API v3 | GitHub Developer Guide. https://developer.github.com/v3/ Accessed: 2020-03-13.

[15] T. L. Graves, A. F. Karr, J. S. Marron, and H. Siy. 2000. Predicting fault incidence using software change history. *IEEE Transactions on Software Engineering* 26, 7 (July 2000), 653–661. https://doi.org/10.1109/32.859533

[16] Alex Guazzelli, Michael Zeller, Wen-Ching Lin, Graham Williams, et al. 2009. PMML: An open standard for sharing models. *The R Journal* 1, 1 (2009), 60–65.

[17] Odd Erik Gundersen and Sigbjørn Kjensmo. 2017. State of the art: Reproducibility in artificial intelligence. In *Conference on Artificial Intelligence (AAAI)*.

[18] Charles Hill, Rachel Bellamy, Thomas Erickson, and Margaret Burnett. 2016. Trials and tribulations of developers of intelligent systems: A field study. In *Symposium on Visual Languages and Human-Centric Computing (VL/HCC)*. 162–170.

[19] Eirini Kalliamvakou, Georgios Gousios, Kelly Blincoe, Leif Singer, Daniel M German, and Daniela Damian. 2014. The Promises and Perils of Mining GitHub. In *Conference on Mining Software Repositories (MSR)*. 92–101. https://doi.org/10.1145/2597073.2597074

[20] Miryung Kim, Thomas Zimmermann, Robert DeLine, and Andrew Begel. 2016. The Emerging Role of Data Scientists on Software Development Teams. In *International Conference on Software Engineering (ICSE)*. 96–107. http://doi.acm.org/10.1145/2884781.2884783

[21] Y Ma, S Fakhoury, M Christensen, V Arnaoudova, W Zogaan, and M Mirakhorli. 2018. Automatic Classification of Software Artifacts in Open-Source Applications. In *Conference on Mining Software Repositories (MSR)*. 414–425.

[22] T. Menzies and T. Zimmermann. 2013. Software Analytics: So What? *IEEE Software* 30, 4 (July 2013), 31–37. https://doi.org/10.1109/MS.2013.86

[23] Hui Miao, Ang Li, Larry S. Davis, and Amol Deshpande. 2016. ModelHub: Towards Unified Data and Lifecycle Management for Deep Learning. *CoRR* abs/1611.06224 (2016). https://arxiv.org/abs/1611.06224

[24] Hui Miao, Ang Li, Larry S Davis, and Amol Deshpande. 2017. On Model Discovery For Hosted Data Science Projects. In *Workshop on Data Management for End-to-End Machine Learning (DEEM'17)*. 6:1—-6:4. https://doi.org/10.1145/3076246.3076252

[25] T. J. Ostrand, E. J. Weyuker, and R. M. Bell. 2005. Predicting the location and number of faults in large software systems. *IEEE Transactions on Software Engineering* 31, 4 (April 2005), 340–355. https://doi.org/10.1109/TSE.2005.49

[26] João Felipe Pimentel, Leonardo Murta, Vanessa Braganholo, and Juliana Freire. 2019. A Large-Scale Study about Quality and Reproducibility of Jupyter Notebooks. In *Conference on Mining Software Repositories (MSR)*. 507–517. https://doi.org/10.1109/MSR.2019.00077

[27] Jim Pivarski, Collin Bennett, and Robert L. Grossman. 2016. Deploying Analytics with the Portable Format for Analytics (PFA). In *Conference on Knowledge Discovery and Data Mining (KDD)* (San Francisco, California, USA). 579–588. https://doi.org/10.1145/2939672.2939731

[28] Gustavo Correa Publio, Diego Esteves, Agnieszka ÅĄawrynowicz, PanÄĮe Panov, Larisa Soldatova, Tommaso Soru, Joaquin Vanschoren, and Hamid Zafar. 2018. ML Schema: Exposing the Semantics of Machine Learning with Schemas and Ontologies. In *Reproducibility in Machine Learning Workshop (RML)*. https://openreview.net/forum?id=B1e8MrXVxQ

[29] D Sculley, Gary Holt, Daniel Golovin, Eugene Davydov, Todd Phillips, Dietmar Ebner, Vinay Chaudhary, Michael Young, Jean-François Crespo, and Dan Dennison. 2015. Hidden Technical Debt in Machine Learning Systems. In *Conference on Neural Information Processing Systems (NIPS)*. 2503–2511.

[30] Akshay Sethi, Anush Sankaran, Naveen Panwar, Shreya Khare, and Senthil Mani. 2018. DLPaper2Code: Auto-generation of Code from Deep Learning Research Papers. In *Conference on Artificial Intelligence (AAAI)*. 7339–7346. https://www.aaai.org/ocs/index.php/AAAI/AAAI18/paper/view/17100

[31] Neel Shah. [n.d.]. ARXIV data from 24,000+ papers Version 2. https://www.kaggle.com/neelshah18/arxivdataset/home Accessed: 2019-01-15.

[32] Christian Szegedy, Sergey Ioffe, Vincent Vanhoucke, and Alexander A Alemi. 2017. Inception-v4, Inception-ResNet and the Impact of Residual Connections on Learning. In *Conference on Artificial Intelligence (AAAI)*.

[33] Erik H. Trainer, Chalalai Chaihirunkarn, Arun Kalyanasundaram, and James D. Herbsleb. 2015. From Personal Tool to Community Resource: What's the Extra Work and Who Will Do It?. In *Conference on Computer Supported Cooperative Work (CSCW)*. 417–430. http://doi.acm.org/10.1145/2675133.2675172

[34] Jason Tsay, Todd Mummert, Norman Bobroff, Alan Braz, and Martin Hirzel. 2018. Runway: Machine Learning Model Experiment Management Tool. In *Conference on Systems and Machine Learning (SysML)*.

[35] Joaquin Vanschoren, Jan N. van Rijn, Bernd Bischl, and Luis Torgo. 2014. OpenML: Networked Science in Machine Learning. *SIGKDD Explorations Newsletter* 15, 2 (June 2014), 49–60. http://doi.acm.org/10.1145/2641190.2641198

[36] Manasi Vartak, Harihar Subramanyam, Wei-En Lee, Srinidhi Viswanathan, Saadiyah Husnoo, Samuel Madden, and Matei Zaharia. 2016. ModelDB: A System for Machine Learning Model Management. In *Workshop on Human-In-the-Loop Data Analytics (HILDA)*. 14:1–14:3. http://doi.acm.org/10.1145/2939502.2939516

[37] Mandana Vaziri, Louis Mandel, Avraham Shinnar, Jérôme Siméon, and Martin Hirzel. 2017. Generating Chat Bots from Web API Specifications. In *Symposium on New Ideas, New Paradigms, and Reflections on Programming and Software (Onward!)*. 44–57. http://doi.acm.org/10.1145/3133850.3133864

[38] Z Wan, X Xia, D Lo, and G C Murphy. 2019. How does Machine Learning Change Software Development Practices? *IEEE Transactions on Software Engineering* (2019), 1. https://doi.org/10.1109/TSE.2019.2937083

[39] Ian H Witten, Eibe Frank, Mark A Hall, and Christopher J Pal. 2016. *Data Mining: Practical machine learning tools and techniques*. Morgan Kaufmann.

A APPENDIX: TOOL AVAILABILITY

AIMMX as described in this paper is available at the time of writing as an open source library under Apache License 2.0[5]. The evaluation dataset and individual model evaluation sample datasets are available as part of a replication set[6]. Instructions on installing and using the AIMMX library are included in the replication set.

B LIST OF DOMAINS AND TASKS INFERRED

- Computer Vision
  - Face Detection
  - Face Verification
  - Image Classification
  - Image Denoising
  - Image Generation
  - Image-to-Image Translation
  - Object Detection
  - Object Localization
  - Person Re-Identification
  - Pose Estimation
  - Scene Text Detection

[5] https://github.com/ibm/aimmx

[6] https://zenodo.org/record/3609308

  - Semantic Segmentation
  - Visual Question Answering
  - Vision Other
- Natural Language Processing
  - Dependency Parsing
  - Language Modelling
  - Machine Translation
  - Named Entity Recognition (NER)
  - Natural Language Inference
  - Part-Of-Speech Tagging
  - Question Answering
  - Sentiment Analysis
  - Text Classification
  - Text Generation
  - NLP Other
- Other Domains
  - Graphs
  - Medical
  - Playing Games
  - Speech
  - Miscellaneous
- Unknown