Empirical Software Engineering (2022) 27:176
https://doi.org/10.1007/s10664-022-10206-6

Extracting enhanced artificial intelligence model metadata from software repositories

Jason Tsay[1] · **Alan Braz**[2] · **Martin Hirzel**[1] · **Avraham Shinnar**[1] · **Todd Mummert**[1]

Accepted: 2 July 2022

Abstract
While artificial intelligence (AI) models have improved at understanding large-scale data, understanding AI models themselves at any scale is difficult. For example, even two models that implement the same network architecture may differ in frameworks, datasets, or even domains. Furthermore, attempting to use either model often requires much manual effort to understand it. As software engineering and AI development share many of the same languages and tools, techniques in mining software repositories should enable more scalable insights into AI models and AI development. However, much of the relevant metadata around models are not easily extractable. This paper (an extension of our MSR 2020 paper) presents a library called AIMMX for AI Model Metadata eXtraction from software repositories into enhanced metadata that conforms to a flexible metadata schema. We evaluated AIMMX against 7,998 open-source models from three sources: model zoos, arXiv AI papers, and state-of-the-art AI papers. We also explored how AIMMX can enable studies and tools to advance engineering support for AI development. As preliminary examples, we present an exploratory analysis for data and method reproducibility over the models in the evaluation dataset and a catalog tool for discovering and managing models. We also demonstrate the flexibility of extracted metadata by using the evaluation dataset in an existing natural language processing (NLP) analysis platform to identify trends in the dataset. Overall, we hope AIMMX fosters research towards better AI development.

Keywords Artificial intelligence · Machine learning · Mining software repositories · Model mining · Model metadata · Model catalog · Metadata extraction

1 Introduction

Despite recent advances, AI as an engineering practice is still in its early stages with often unpredictable and costly results (both in terms of time and quality) (Hill et al. 2016) that are

Communicated by: Georgios Gousios, Sarah Nadi

This article belongs to the Topical Collection: *Mining Software Repositories (MSR)*

✉ Jason Tsay
jason.tsay@ibm.com

Extended author information available on the last page of the article.

Published online: 20 September 2022

often difficult to reproduce (Gundersen and Kjensmo 2017). The sheer amount of possible AI approaches and algorithms (Wan et al. 2019) and recent increase in released AI frameworks (Braiek et al. 2018) result in a large variety of AI models and representations. For this paper, we define an AI *model* as all the software and data artifacts needed to specify the statistical model for a given task, train its learnable coefficients, and/or deploy the trained model for prediction in a service or application. Our definition of *model* includes both traditional machine learning (ML) and deep learning (DL) models. The sheer variety and lack of standardization in AI development 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 is a barrier for non-experts and hinders mass adoption. We propose that mining standardized model metadata will reduce this manual effort. We then propose that further enhancing the extracted metadata with AI model-specific information enables programmatically analyzing or interacting with models at scale.

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 for 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 (Amershi et al. 2019). One possibility is that existing software repository mining techniques such as software analytics techniques (Menzies and Zimmermann 2013) or bug prediction techniques (Ostrand et al. 2005; Graves et al. 2000) 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 (Amershi et al. 2019; Sculley et al. 2015), 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 buried in documentation.

We present a library called AIMMX (AI Model Metadata eXtractor) for simplified and standardized mining of AI model-specific metadata from software repositories. The extractors take existing software repositories containing AI models, and integrate data from multiple sources, such as documentation, Python code, model definition files, etc. Our extractors integrate this data for AI model-specific metadata. Integration also enables further enhancement via inferring additional model-specific metadata that is not easily available directly from software repositories. The extraction library contains six main modules to extract model-specific metadata: model name, paper references, dataset, AI frameworks, model domain, and ML vs DL. Additionally, the library includes a gatekeeper module that infers whether a given repository contains an AI model. Further, the library uses the extracted metadata to infer additional signals called *readiness* metrics that indicate how close a given model is to *trainability* or *deployability*. The inference modules such as AI,

domain, and ML vs DL use machine learning themselves to automatically infer properties of the model. To standardize the extracted metadata, we also provide an AI model metadata schema based on model-related steps of the AI development lifecycle (Amershi et al. 2019): model definition, training, and post-training. In contrast to other model metadata efforts such as ONNX (ONNX 2017), PMML (Guazzelli et al. 2009), and PFA (Pivarski et al. 2016) that focus on defining the model's low-level computational graph, our metadata schema and mining library are 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.

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 evaluate AIMMX and demonstrate its capabilities by collecting and analyzing 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. After extraction, the metadata is ready for consumption in both machine-readable and human-readable views. Using a subset of this dataset, we created test sets and evaluations for each of our five extraction modules and readiness metrics as mentioned above as well as a holistic evaluation of the entire system. The automatically extracted metadata have an average precision of 85% and recall of 82%. The evaluation dataset is available as part of the replication package. Figure 1 gives an overview of the model metadata mining system, dataset collected, and preliminary usage of the extracted metadata.

We demonstrate the capabilities of mining a large collection of AI model metadata by using the enhanced metadata to perform an exploratory analysis of reproducibility, a supplementary analysis using an existing natural language processing (NLP) analysis platform called Watson Discovery (IBM 2020), and an example model catalog tool. Reproducibility

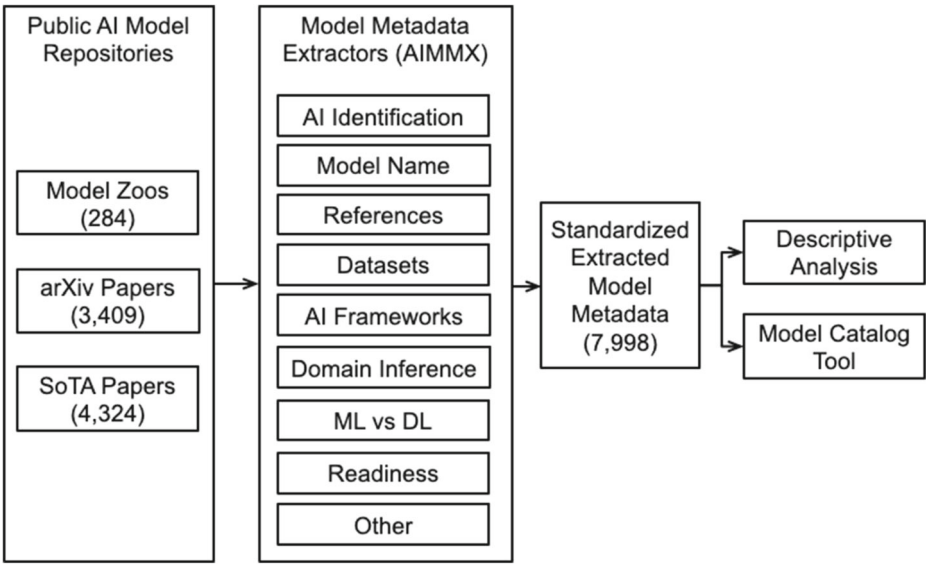

Fig. 1 Overview of AI model metadata mining system

in AI papers (Gundersen and Kjensmo 2017) and Jupyter Notebooks (Pimentel et al. 2019) tends to be relatively poor, due to a lack of documentation over method selection, datasets used, or experiments run. We quantitatively examine our enhanced 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 (Gundersen and Kjensmo 2017). Our exploratory analysis found that data reproducibility tends to be relatively low 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 a demonstration of the versatility of our standardized metadata, we feed the 7,998 documents in the dataset into an existing NLP analysis platform and examine trends in our dataset using the platform. 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 (Vartak et al. 2016) require that model producers instrument their code to support automatic model ingestion. 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 (Code 2020). These connections may also enable automated training or deployment in future tools.

This paper is an extension of our previous work (Tsay et al. 2020) presented at the 17th International Conference on Mining Software Repositories (MSR 2020) which describes an earlier version of AIMMX and a portion of the features, evaluation, and analyses described in this paper. As part of the extension, this version of the paper describes the model metadata schema in detail. This version also includes three new library modules: 1) a gatekeeper that uses machine learning to identify whether a given repository is an AI model, 2) a classifier that uses machine learning to infer if a model uses traditional machine learning or deep learning, and 3) an aggregator that uses extracted metadata to infer readiness for a given model. Each of these modules also includes an accompanying individual evaluation; the overall system evaluation has also been updated. This version also demonstrates the flexibility of our extracted metadata format with an additional analysis of trends using an existing NLP analysis platform called Watson Discovery. Finally, this version more fully describes the implementation of the cataloging tool.

This paper makes the following contributions:

- A standardized AI model metadata schema that covers a wide range of model types, domains, and lifecycle phases (Section 2).
- A tool for extracting standardized AI model-specific metadata from software repositories with currently eight extraction modules (Section 3).
- An evaluation of our tool against a dataset of 7,998 models (Section 4). 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 using an existing NLP platform (Section 5) and a cataloging tool (Section 6).

All in all, we hope our paper will facilitate research into AI development, with the ultimate goal of increasing productivity and improving outcomes.

2 Model Metadata Schema

AI models are diverse, and any attempt to extract standardized information about them must start with establishing a metadata schema to describe these models and where they come from. The diversity of AI models spans dimensions such as domain, lifecycle stage, source, dataset, framework, and so on, where each dimension may significantly affect what a model does and how to use it (Sculley et al. 2015). Our schema takes a model lifecycle-based approach, describing models based on what is known about each lifecycle stage. At a high level, models go through the following lifecycle: pre-training (which includes defining the code or network for a model and for any pre-processing), training (using the definition to learn weights and biases from a training dataset), and post-training (the weights and biases that support making predictions integratable into other applications). The actions that are available on a model are completely different depending on its lifecycle stage. For example, although the TensorFlow and Caffe2 model zoos (see Table 1) both contain ResNet-based models (He et al. 2016), the TensorFlow version is simply code that must undergo training whereas the Caffe2 version is a post-trained binary. Despite both models being from model zoos and having the same topology, domain, and purpose, the actions available and therefore metadata that must be collected differ.

2.1 Schema Description

The model metadata schema consists of a top-level model object with discovery attributes such as model name, domain, references, and so on as well as a number of subobjects that correspond to stages of the model lifecycle: *definition* (pre-training), *training*, and *trained_model*. There are also subobjects to describe *evaluations* such as training metrics like accuracy and a model's *provenance*. Figure 2 shows an overview of the schema and its subobjects. Note that any given model may have any or none of these subobjects. Although one might expect a model to strictly accumulate more information throughout its lifecycle, in practice, any given model may not have information about earlier stages. For example, a trained model from the Caffe2 model zoo may not have any pre-training code available and therefore no *definition* subobject. We represent model metadata using JSON, which is the most popular data exchange format in web APIs, ahead of XML (Rodríguez et al. 2016). Consequently, we express its schema in the JSON Schema language (Internet Engineering Task Force 2018; Pezoa et al. 2016). Our schema is available for viewing online at https://ibm.biz/ai-model-catalog-emse-schema.

JSON Schema is a popular JSON-based method of describing and validating the properties of JSON documents in a standardized way. We use JSON Schema because it is widely supported and is machine and human-readable. It supports primitive types (e.g., {*'type': 'integer', 'minimum': 0*}, enumerations (e.g., {*'enum': [42, 'hello', null]*}), arrays (e.g., {*'type': 'array', 'items':* {*'type':'num-'string'*}), objects (e.g., {*'type': 'object', 'properties':* {*'x':* {*'type': 'number'*}, *'y':* {*'type': 'number'*}}}),), logical connectives (*not*, *allOf*, *anyOf*, *oneOf*), and potentially recursive references (e.g., {*'$ref': '#/definitions/foo'*}) to reusable schema fragments. JSON Schema also has the advantage of being flexible enough to extend to support both relatively complex subobjects and new features as described below.

The top level of the AIMMX schema contains high-level attributes that are common to most models regardless of lifecycle stage, such as a model's name, authors, description, tags, domain or intended application type, and any related references such as academic papers or

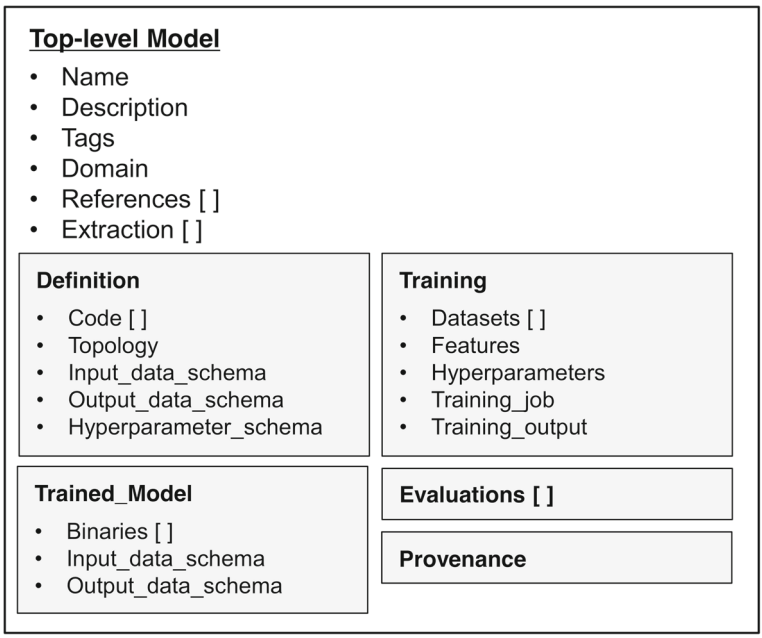

Fig. 2 Model metadata schema overview

blogs. Finally, the top level also stores extraction metadata, such as where a model comes from and metrics related to automated mining such as completeness and confidence.

The *definition* subobject contains the pre-training metadata for a model. It contains references to code that define the model and associated metadata as well as any metadata needed to train the model. The main attribute is *code*, which contains references to source or repositories that define the model including information such as the frameworks (e.g. TensorFlow or Caffe2) and the license (e.g. Apache). The subobject also has schemas defining the shape of input data, output data, and hyperparameters for the model. These schemas are each implemented as embedded JSON Schemas that are definable by the user. For example, the *hyperparameter_schema* is a JSON Schema that may define hyperparameters such as *batch_size* and their shape, such as being an integer with a given minimum and maximum.

The *training* subobject contains information about the training phase of the model, including datasets and hyperparameters that are used to train the model. Whereas the *definition* subobject describes metadata that defines the model and how it may be trained, the *training* subobject describes the specific training configuration. For example, the *definition* subobject contains the hyperparameter schema while the *training* subobject contains actual hyperparameter values used to train a particular model. Similarly, this subobject contains metadata about the actual datasets used during training, which includes information such as location of the data, train/test/validation splits, and any preprocessing steps. Both datasets and hyperparameters may be validated against the schemas defined in the *definition* subobject. The *training* subobject also includes information about the training job, such as the service and/or location and the location of the resulting trained model (but not information about the resulting trained model itself, which is in the *trained_model* subobject).

The *trained_model* subobject contains post-training information about the model weights and biases and any metadata to integrate the trained model into software applications or

model ensembles. The main attribute is *binaries*, with references and information about learned weights and biases. We separated the information about the model binary from the *training* subobject because often, such as in model zoos, there may be information about a trained model binary but no information available about the training process that creates the binary. The *trained_model* subobject also contains schema information about input and/or output data, again using embedded JSON Schemas.

The *evaluations* and *provenance* subobjects respectively contain evaluation metrics and history of any transformations applied to a given model. Both subobjects are not used by the current version of AIMMX but are designed for future extractors. The *evaluations* subobject is meant for performance metrics such as accuracy from training jobs in the most common case. However, the schema we define is flexible enough to include annotations for other subobjects. For example, one may want to evaluate the definition of the data preprocessing pipeline for bias (Shaikh et al. 2017), or the binaries of the trained model for robustness against extraction (Tramèr et al. 2016). The *provenance* subobject is meant to track history of changes. Using the previous examples, if a more fair or more robust model is derived, this subobject would contain information about the original model and any transformations applied.

2.2 Schema Features

While the previous section describes the concrete schema, this section explains how to specify and validate metadata against it, based on standard JSON Schema (Internet Engineering Task Force 2018) Draft 04 with some extensions.

Some properties of the metadata schema are actually definitions of how data should be described. For example, in the *definition* subobject is a property called *hyperparameter_schema* that describes what hyperparameters the model definition code expects and potentially the ranges of these values. In the *training* subobject, these hyperparameters are actually defined. Schemas themselves are a natural fit for describing these types of properties and are implemented in our system by specifying that the relevant attribute contains an embedded JSON Schema. Note that this is not an extension to JSON Schema, but simply an unusual usage. Since the JSON Schema is itself specified (meta-circularly) using a JSON Schema, it is easy to reference and embed.

As in the above example with the *hyperparameter_schema* and *hyperparameters* properties, user-provided embedded schemas may validate other properties. Conceptually, this is similar to dependent types (Augustsson 1998) in programming language theory: the type (schema) for part of the data is given by another part of the data. AIMMX implements this in a backwards-compatible manner by encoding a special *$dependent_schema* property. This format is JSON Schema conformant, and so existing validators will ignore it without failing. The following example encodes that the hyperparameter values in the model metadata should be checked against the schema embedded elsewhere in the model metadata:

```
"hyperparameters": {
  "definitions": {
    "$dependent_schema": {
      "enum": ["/definition/hyperparameter_schema"]
    }
  }
}
```

Using a standardized schema for our schema definition allows for downstream applications to use standard validators to validate input data against our schema. The catalog tool described in Section 6 uses the AJV (Ajv 2018) validator. We augment the validator to support embedded schema validation by first finding all the embedded schema declarations. These are replaced in-memory with a JSON Schema that uses the *$allOf* combinator to combine the existing schema with an empty hole. A *hole map* associates dependent schema paths (for example, *hyperparameter_schema*) to a list of holes to fill. Using this modified schema and hole map, we can now validate data against the schema by first going through the map and filling the holes in the schema with the corresponding dependent schema contained in the data. Standard validation then validates the dependent data (for example, *hyperparameters*) against the filled schema, which now includes the corresponding dependent schema.

3 Automated Model Metadata Extraction

The core of AIMMX is a Python library that reads software repositories, specifically from GitHub (2020), and extracts AI model-related information into standardized model metadata in the JSON format that is machine and human readable by following the schema described in the previous section. This library is open-source at https://github.com/ibm/aimmx and publicly available for use. 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 (Amershi et al. 2019). For example, most major AI-related frameworks such as TensorFlow, 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 often spend extra effort to ensure documentation is updated (Trainer et al. 2015). GitHub also has a rich Application Programming Interface (API) (GitHub 2016) 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, whereas the TensorFlow model zoo contains multiple folders with one model each, the Keras model zoo contains a folder with multiple Python files with one model each. 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 (Dabbish et al. 2012)), 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 *input_data_schema* 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 are 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 can then enhance the aggregated metadata that are more directly extractable from software repositories, code, and configuration files. The current version of the extractors contains seven such modules: AI identification, model name, references, associated datasets, AI frameworks used, model domain inference, and ML vs DL. We also use our existing metadata to infer *readiness* signals for how close a model may be to training or deploying.

3.1 AI Model Identification

A surprisingly difficult task when mining software repositories for AI-related code at a large scale is identifying AI vs non-AI software projects. Gathering software repositories for mining often involves indiscriminately gathering repositories through means such as GitHub search or querying GHTorrent (Gousios 2013) or GH Archive (Archive 2021). Even for the initial development of this library we had difficulty determining which repositories contained AI models. Hence, we gathered repositories from sources more likely to contain AI models (see Section 4.1), yielding fewer repositories than indiscriminate methods.

In response to this difficulty, we developed a gatekeeper module that assists in identifying and filtering out non-AI repositories out of a large indiscriminate set. This module takes a given repository and attempts to use its metadata to identify if the repository is an AI model or not. AI-related tools and frameworks are not considered models in this case. For the current version of this module, we only analyze the contents of the README file.

To create this classifier module, we needed a ground truth dataset of AI and non-AI repositories. We used the ML-Universe dataset by Gonzalez et al. (2020), which contains 4,524 AI and 4,101 non-AI repositories, identified manually. We only consider repositories that were available (as of 9/6/2021) with valid README files, since those are used in the classification. After balancing for AI and non-AI, we have 4,039 valid repositories for each case in the dataset (8,072 total). This dataset is then split into training and validation sets with 80% or 6,462 repositories in the training set and 1,616 in the validation 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 but it is stripped of all tags and special Markdown characters and then tokenized and vectorized. For all 8,072 repositories in our dataset, we extract the README and preprocess it as described earlier. After testing a number of classification approaches, we settled on a logistic regression classifier that outputs a Boolean indicating whether the repository is an AI model or not.

3.2 Model Name Extraction

This module attempts to extract a more descriptive name for a given model from available metadata. In many cases, the most obvious choice, 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, it is stripped of Markdown or HTML characters and any hyperlinks. This cleaned line is returned as a potential name. If no potential name is found, the repository name is used as a fallback.

3.3 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) identifying 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, this approach is limited in that only references that match the patterns defined will be found.

The second approach searches for arXiv papers. ArXiv is a preprint hosting service particularly popular with academics in AI fields (1991). Specifically, the approach searches for links to arXiv papers within the given README and then extracts the arXiv ID from the link. The ID is then looked up against the arXiv API (2018) for additional information such as the article title, authors, and publishing date. The advantage of this approach is that arXiv is popular among 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 with 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. 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.

3.4 Dataset Extraction

Data management is a hard challenge in engineering AI systems (Amershi et al. 2019; Wan et al. 2019) 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. The first approach searches the README for links that contain dataset-related keywords, specifically "dataset", "data", and "corpus." It then returns the names and the referenced URLs of the extracted dataset. The second approach uses a set of 640 common dataset names and searches for mentions of these datasets in the README. To avoid partial matching of short dataset names such as "SQuAD" versus "SQuAD2.0," matching datasets must be their own token(s) and surrounded by whitespace or punctuation. Longer dataset names such as "Fashion-MNIST" are preferred over shorter ones such as "MNIST" and are resolved first. 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.

We extracted the list of common datasets using the Papers With Code website (Code 2020), which compiles machine-learning papers and repositories and metadata that links the two. In the Papers With Code data,[1] 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 (Chelba et al. 2013). 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 as where the dataset is available and the schema could also be made available.

3.5 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 (Braiek et al. 2018) and framework-related questions are a major category of machine learning-related topics on Stack Overflow (Bangash et al. 2019). 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 (Braiek et al. 2018). The module then concatenates all Python files (.py) and code cells of Jupyter Notebooks (.ipynb) into a single text string.

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

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 they exist, we add *Caffe2* to the frameworks list. Table 7 has a full list of AI frameworks for extraction.

3.6 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 (2020). 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 validation set. Similar to the AI model identification module, the current version of this module takes a bag-of-words approach and only considers the README, preprocessed by stripping all tags and special Markdown characters and then tokenizing and vectorizing the words.

Through examining the dataset and empirically, we settled on an assembly of support vector classification models that work in a two-stage process as seen in Fig. 3. The first stage determines 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 final result 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

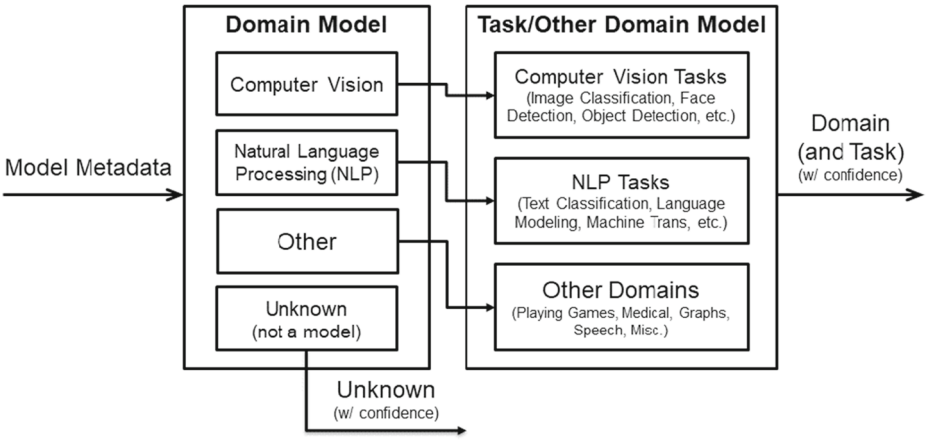

Fig. 3 Domain inference machine learning model ensemble diagram

(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%). We chose to group domains into an "Other" category due to a relatively small number of samples, especially compared to Computer Vision or NLP. Due to the relatively small number of examples for these models, we chose not to further predict the task for these domains. Appendix A lists all domains and tasks inferred.

3.7 Machine Learning/Deep Learning (ML-DL) Inference

An often important distinction within AI models is whether it is a machine learning (ML) or deep learning (DL) model. In this case, although DL is a sub-category of ML, we refer to ML models as "traditional" machine learning models that do not use deep learning techniques such as neural networks. This module uses machine learning to infer whether a given AI model is ML or DL. We train the classification model for this module using the IBM Natural Language Understanding[2] service that analyzes text to extract metadata from content such as concepts, entities, emotion, relations, sentiment among others, as well as the ability to train a custom classification model.

To create the dataset to train and evaluate this module, we use a dataset of GitHub projects and use GitHub topics to categorize repositories as either ML or DL. We used an open-source GitHub crawler[3] to gather 3,000 ML and DL repositories each. To determine which category each repository belongs to, we filter via GitHub topics: ML repositories must have *machine-learning* but *not* *deep-learning* as topics and DL repositories must have *deep-learning* but *not* *machine-learning*. Out of these two mutually exclusive sets of repositories, we gather 3,000 each with the most stars and with valid README files. We applied the same cleanup to README files as discussed earlier to get plain-text versions. We also removed files with fewer than 500 characters and trimmed each text to the first 2,000 characters, which is the maximum length of a sample for the IBM NLU service. After this preprocessing, our dataset has 2,716 ML repositories and 2,802 DL repositories. We then create a training set with the

[2] https://cloud.ibm.com/catalog/services/natural-language-understanding
[3] https://github.com/IBM/github-crawler

top 2,000 repositories of each category in terms of star count, resulting in a balanced dataset with a total of 4,000 samples. We create a test set with the next 500 repositories of each category for a total of 1,000 samples.

We then use the NLU service to train a custom classifier model using the *classification* endpoint. The service also provides an *analyze* endpoint to predict ML/DL classification given a cleaned README file from a given repository. Our module then uses this endpoint to infer ML/DL.

3.8 Readiness Metrics

AIMMX also aggregates the extracted metadata to further infer additional signals about a given AI model. In particular, since much of our metadata is extracted from documentation, we expect that we should be able to infer how ready to reuse a given model is. We call these inferred signals *readiness* metrics and the current version of AIMMX calculates *trainability* and *deployability* metrics for how close a model is to a trainable or deployable state. "Trainable" in this case refers to the feasibility of running a training job on a given model which may include acquiring the dataset, formatting and cleaning the dataset, generating features, and running the training script. "Deployable" refers to feasibly running prediction or inference on a trained model for some unseen data (rather than just running a test set).

As described in Section 2.1, our metadata schema follows a lifecycle approach where we group relevant metadata via subobjects that correspond to stages in the model's lifecycle. For example, extracted metadata about datasets used is under the *training* subobject because it is relevant metadata about running a training job. We make use of these lifecycle subobjects to approximate what stage of the lifecycle a given model is. In the current version of our system, we calculate trainability as the percentage of defined properties in the *definition* and *training* lifecycle subobjects of the metadata. Deployability is similarly defined as the percent of properties defined in the *trained_model* lifecycle subobject. For example, after extracting metadata for a given project, that project may have code repository, AI framework, code files, and dataset metadata in its metadata document. These properties being defined comprise 50% of the *definition* and *training* subobjects in our schema which then is a *trainability* metric of 50%. Adding additional relevant metadata such as the hyperparameter schema would raise the value further. This percentage approach to calculating the metrics has an advantage of being inherently normalized because it relies on presence of properties rather than values. These measures are used in analyses to give a sense of the relative completeness of the model and the catalog tool described later uses them to recommend steps towards training or deployment.

4 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 seven model-specific metadata extraction modules as well as our inferred readiness metrics. Each of the module evaluations uses its own methodology, subset of the collected dataset, and evaluation metrics. 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 in the replication package (https://zenodo.org/record/5655729).

4.1 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 (1991), and 3) 4,324 repositories associated with state-of-the-art AI models (Code 2020). Table 1 summarizes the dataset. Since 19 models overlap from multiple sources, the total number of repositories is 7,998. We note that this dataset was created before implementing the AI Identification module and this challenge of identifying AI model repositories inspired the module's creation.

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 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* (1991) papers published on the Kaggle competition service (Shah 2019), 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 (2020). The website lists various machine learning tasks with associated datasets and leaderboards of the performance of AI papers for these

Table 1 Evaluation dataset summary

Model source/Zoo	# 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)

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 labels available through their public data.

The rest of this section presents a detailed evaluation, for which Table 2 gives a short summary.

4.2 AI Model Identification

To train the AI model identification module (Section 3.1, we held out 20% (or 1,616 out of 8,072 total) repositories as a test set. Half of the repositories are known to be AI models and half are non-AI. Evaluating the module on this test set resulted in an accuracy of 0.915 and an F1 score of 0.915. This evaluation is a standard classification task so we report accuracy.

4.3 Model Name Extraction

To evaluate the model name extraction module (Section 3.2), 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, necessitating 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, "gan" versus "Dist-GAN." 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 default repository name is the full name of a model or approach. For example, "BERT" is correct for the BERT model (Devlin et al. 2018) because that is the full name according to the repository. The percentage correct of the test set was 85.3% or 341 of 400 repositories. We chose correctness as a metric due to the qualitative nature of the evaluation.

Table 2 Evaluation results summary

Evaluation	Count	Metric	Value
AI Identification	1,616	Accuracy	0.915
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
ML-DL Classifier	1,000	Accuracy	0.820
Readiness	80	Trainability	0.800
		Deployability	0.450
System	80	Precision	0.858
		Recall	0.829

4.4 Reference Extraction

To evaluate the reference extraction module (Section 3.3), 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 used the *SotA* dataset described earlier from Papers With Code (2020) 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 point back to its paper. Papers in the test set may be associated with multiple repositories and repositories may be associated with multiple papers. We chose to use precision as the metric 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.

4.5 Dataset Extraction

To evaluate the dataset extraction module (Section 3.4), 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. 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 76.9%, the recall was 76.0%, and the F1 score was 75.8%. 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%. We report F1 score and not accuracy for this evaluation because both incorrectly identifying datasets and incorrectly including datasets are sources of error for this extractor.

4.6 Framework Extraction

To evaluate the framework extraction module (Section 3.5), we use 284 models from model zoos as ground truth as most zoos are associated with a particular deep learning 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%. We use precision and

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

not recall because our ground truth dataset for this evaluation only identifies one particular framework that a model should have rather than the entire set of frameworks.

4.7 Automated Domain Inference

To train the domain inference module (Section 3.6), we created a training dataset from a subset of the *SotA* dataset along with non-model software repositories. Ground truth labels are from the Papers With Code website as described ealier. From the 3,215 repositories labeled with domain information, we reserved 30% or 978 for a test set. Each of the repositories in the test set was 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. We used stratified sampling to create the test set to ensure that each domain and task are represented to mitigate biases. 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. Table 4 breaks down the results by domains. 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. This evaluation is a standard classification task so we report accuracy.

4.8 Machine Learning/Deep Learning (ML-DL) Inference

We trained the classification model in our ML-DL inference module (Section 3.7) with the Watson NLU service and held out 20% of the dataset as a test set or 1,000 repositories out of 5,000 total. Half of the repositories in the test set are ML models and half are DL. This service also provides an API endpoint to analyze new samples, so we use it to classify our

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	

test set. Evaluating this test set resulted in an accuracy of 0.82 and an F1 score of 0.8199. This evaluation is a standard classification task so we report accuracy.

4.9 Readiness Metrics

To evaluate our inferred readiness metrics of *trainability* and *deployability* (Section 3.8), we manually evaluated the feasibility of training and deploying a random sample of 80 repositories of the collected dataset of 7,998 and compared against the reported readiness metrics. We first manually created a random sample of 80 repositories consisting of four partitions of 20 repositories each: low *trainability*, high *trainability*, low *deployability*, and high *deployability*. We consider a repository "low" trainability if the metric is 35 or less and "high" if the metric is 50 or higher. We consider a repository "low" deployability if the metric is 50 or less and "high" if the metric is 70 or higher. These thresholds were chosen by examining the distribution of readiness metrics for repositories in our sample. We chose thresholds that would roughly divide the repositories in half. Each repository must be primarily Python with a README file (if it exists) in English. For each of the repositories in the set, a researcher manually created a ground truth dataset by examining the repository and using domain knowledge to determine the feasibility of training and deployment. The researcher did not have access to the readiness score of the repository. *Training* here is defined as a repository that has enough information to run a training job which includes feasibly acquiring the dataset used, formatting and cleaning the dataset, generating features, and running the training scripts. *Deployability* here is defined as a repository that has enough information to run prediction or inference on a pre-trained model (or provide a method for training the model). Specifically, the repository must provide for a method of performing inference on unseen data rather than simply evaluating the trained model on a test set. In both cases, software dependencies (with versions) also must be stated explicitly, usually in the README file or in *requirements.txt* as is common in Python projects. Partial dependencies that cover the primary dependencies (such as TensorFlow = 1.13) were allowed. For the ground truth set, each repository is marked as trainable or not and deployable or not.

Table 5 reports the percentage of trainable repositories for the *trainability* set and the deployable respositories for the *deployability* set. For both trainability and deployability, the "high" case had higher percentages of reusability than the "low" cases. For trainability, 80% of the "high" set were feasibly trainable compared to 30% in the "low" set, suggesting that the metric mostly correctly indicates reusability in terms of training. The deployability metric performs less well as only 45% of the "high" set were feasibly deployable.

4.10 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

Table 5 Summary of readiness metric evaluation

Readiness	Level	Percentage
Trainability	Low	30%
	High	80%
Deployability	Low	25%
	High	45%

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 (references) is missing. Our evaluation weighs each property equally, regardless of whether the property is a single value or list. This is done to prevent biasing in favor of properties with multiple values. As the previous example demonstrates, list properties are counted as one property as it gives a more conservative indication of the performance of the extraction; any error in the list results in the property being marked incorrect. 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. This is due to the system evaluation dataset being created before the implementation of the AI identification module. Due to this, the AI or not AI property is only counted if incorrect such that it can only penalize the final evaluation. The precision of our system evaluation was 85.82%, the recall was 82.87%, and the F1 score was 84.24%. Upon further inspection of the evaluation sample, the performance may be inflated due to properties extracted automatically through GitHub. If we only include metadata returned by AIMMX's eight extraction modules, then the precision drops to 71.98%, the recall to 69.40%, and the F1 score to 70.43%. We report F1 score rather than accuracy because sources of error for this evaluation include both incorrectly identifying a property and incorrectly including a property.

5 Preliminary Metadata Analyses

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 exploratory analyses of model reproducibility. Furthermore, we use an existing analysis and visualization platform to explore trends and relationships in the dataset.

5.1 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 (Gundersen and Kjensmo 2017). Borrowing terminology from Gundersen and Kjensmo (2017), 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 Gundersen and Kjensmo 2017; Pimentel et al. 2019) but rather quantitatively analyze a larger-scale dataset for signals of reproducibility based on literature.

Table 6 reports descriptive statistics for the repositories in the dataset. We split the statistics by source of the repositories as described in the previous section: "model zoos", from arXiv (arXiv 1991) papers, and state-of-the-art AI models (Code 2020) (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 (Pimentel et al. 2019) (a distribution of usage is available in Table 7).

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 AI models tends to be highly tied to data used and its processing (Wan et al. 2019; Breck et al. 2019). In enterprise settings, this reliance on quality data means that sharing and reusing datasets is vitally important (Amershi et al. 2019). We use extracted datasets to explore the degree to which types of models have documentation regarding datasets. Table 6 shows that 42% of models in our sample have an extracted dataset with state-of-the-art models having a higher rate of extracted datasets at 49% and arXiv models having 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 8. 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

Table 6 Evaluation dataset descriptives

Attribute	Overall	Model Zoo Dataset	arXiv Dataset	SotA Dataset
Median Stars	12	17,513	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	7,998	284	3,409	4,324

Table 7 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

testing. Our findings are in line with Gundersen et al.'s study with a similar rate of dataset sharing (49% vs 42%) (Gundersen and Kjensmo 2017).

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 (Wan et al. 2019). From our descriptives in Table 6, 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 8), 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.

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

Table 8 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%

identity that a popular dataset such as MNIST is used and provide meta-features of the dataset such as size and number of classes.

5.2 Watson Discovery Analysis

To demonstrate the flexibility of our standardized model metadata, we feed the 7,998 metadata documents in the evaluation dataset into Watson Discovery (IBM 2020), an AI-powered search and text analytics tool that uses NLP algorithms to aggregate and discover insights from documents, webpages, or other structured or unstructured data. This platform makes it possible to rapidly build cloud-based applications that help to undercover insights hidden in unstructured data. We use this existing NLP analysis platform to demonstrate that, without modification, metadata extracted via AIMMX is general enough to use directly in other analysis platforms. Specifically, we uploaded our 7,998 metadata documents into a collection on our instance of Watson Discovery on IBM Cloud. Then, we were able to use its Query API to aggregate documents.

Figure 4 aggregates documents by creation date and date of last update for repositories in our dataset, showing a large increase in AI model repositories in 2018. As described in Section 4.1, our dataset is mostly repositories from academic AI papers and the state-of-the-art models. This sudden boost may indicate an explosion of deep learning papers stemming from reusable innovations in 2017-2018 such as attention models (Vaswani et al. 2017), BERT (Devlin et al. 2018), and PyTorch (Paszke et al. 2019). The increase in updated vs. created repositories may also indicate a trend away from single-use "tissue code" repositories that accompany a paper towards these reusable AI models.

We also aggregate our documents via enhanced metadata, examining the relationships between AI model domain and framework, dataset, and references. Figure 5 shows the relationships between AI frameworks used by models and their domain. Many of the popular deep learning frameworks like TensorFlow, PyTorch, and Keras are used across domains but a large portion of their models are in Computer Vision. This is in line with Fig. 6 which shows the relationships between the top 20 datasets used by models and their domain which are also primarily Computer Vision datasets. However, there are still some interesting overlaps between primarily Computer Vision datasets and NLP models such as with ImageNet, CIFAR-10, and MNIST. This may indicate some cross-polination between Computer Vision and NLP approaches such as using CNNs to represent documents (e.g. Conneau et al. 2017

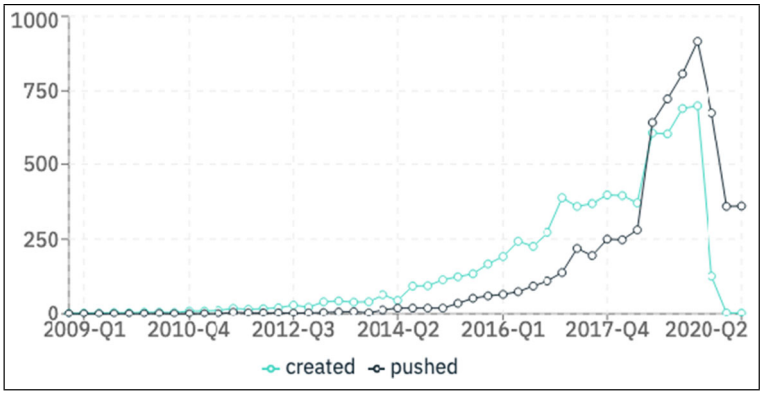

Fig. 4 Distribution of creating/updating repositories in evaluation dataset by quarter

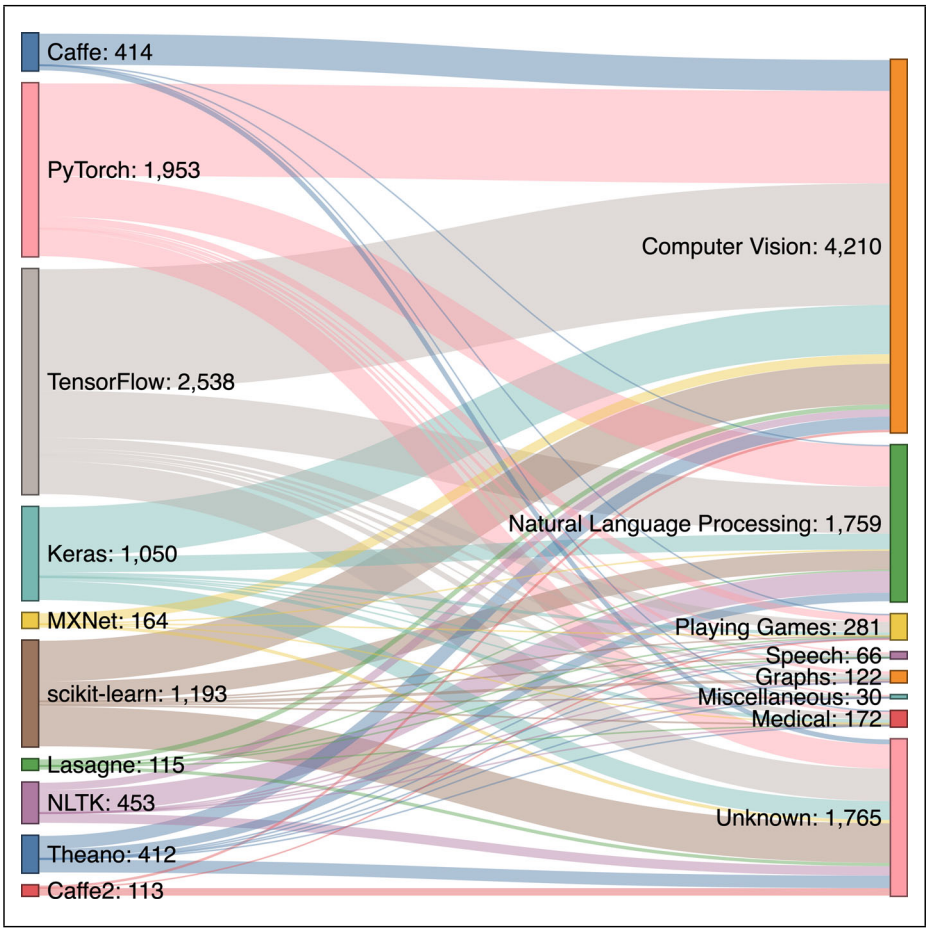

Fig. 5 Relationship between AI Frameworks and model domains in evaluation dataset

in our dataset). A similar cross-polination may be seen in Fig. 7 which shows the relationships between the top 20 papers referenced by models and their domain. Most of the papers are for Computer Vision models but one paper (Ronneberger et al. 2015) is cited often in both Computer Vision and Medical models. One caveat is that all of these aggregations by enhanced metadata are subject to errors in our extractors, such as misclassifying a model's domain.

6 Model Catalog Tool

As an example of a tool that is able to leverage extracted metadata, we implement and describe a catalog web application for the discovery and management of AI models. The web application follows a standard client-server architecture, where the model metadata is stored in a back-end document-oriented Apache CouchDB (Apache 2019) database with a Node.js server that implements a REST API to interact with the model metadata through

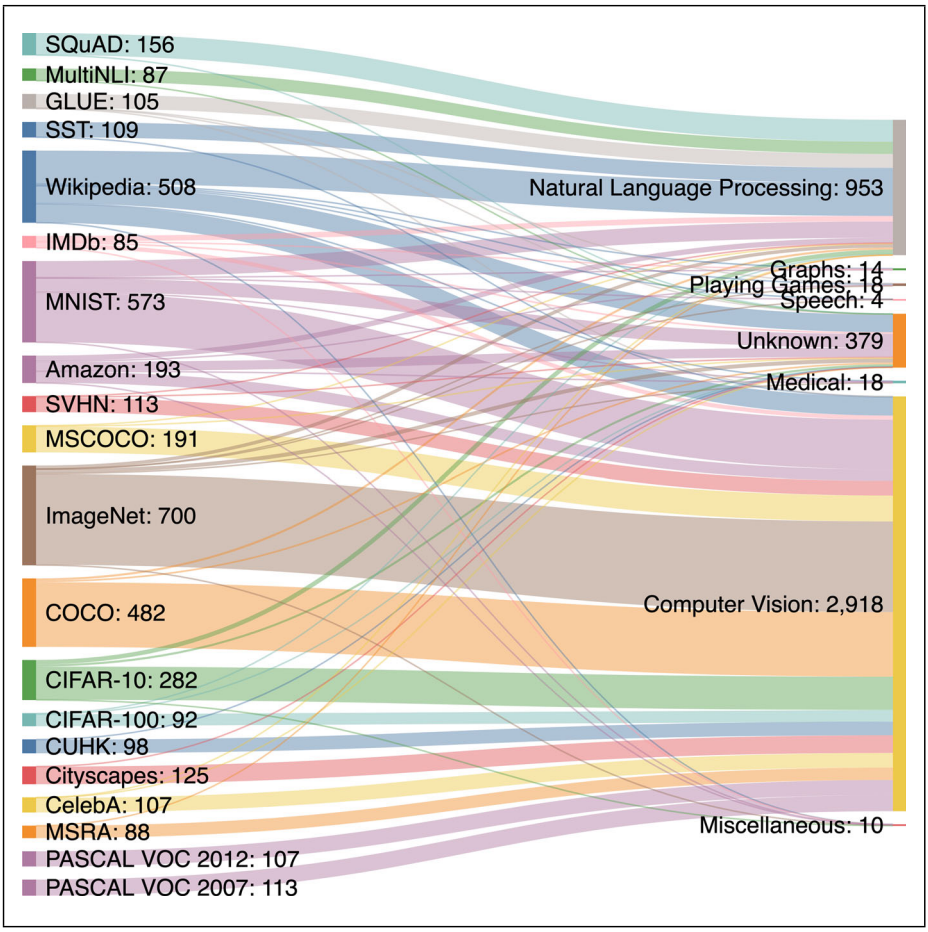

Fig. 6 Relationship between top 20 datasets and model domains in evaluation dataset

basic create-read-update-delete functions. The REST API automatically validates stored or user-input metadata against the model metadata schema from Section 2. The front-end catalog UI is implemented in React and implements a number of discovery features to search for models. The system is available as an online service.[4]

The catalog application consists of three main views: a list of models with filter and search features (see Fig. 8); a page that displays individual model details (see Fig. 9); and a statistics page with visualizations to support data analysis (see Figs. 10 and 11). Models in this catalog are added by providing GitHub repository URLs which are then passed to AIMMX for metadata extraction. The extracted metadata are then inserted into the catalog's database, validated automatically against the schema from Section 2, and then made available for discovery. Users can then update specific fields such as name, description, tags, and domain in the imported model should the automated mining be inaccurate or incomplete.

[4] https://ibm.biz/ai-model-catalog-emse

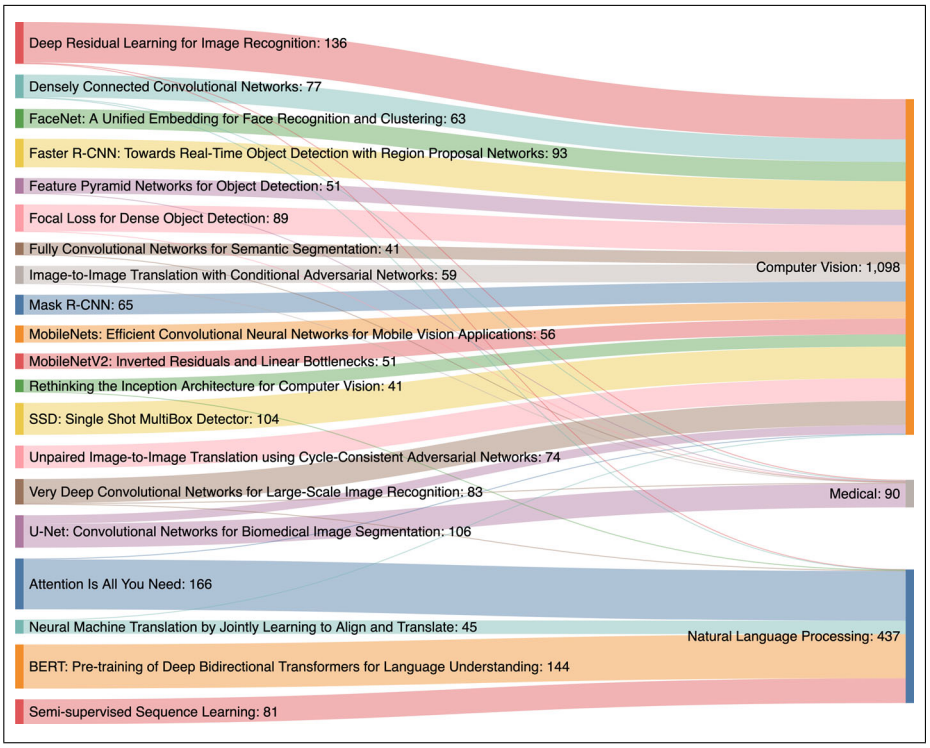

Fig. 7 Relationship between top 20 reference papers and model domains in eval. dataset

If the linked repository changes, the model can be re-imported, which passes the repository back to AIMMX with an option to preserve any manually edited fields.

The model list view (Fig. 8) 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. A side panel offers the following attributes to filter: lifecycle stage, readiness, domain, frameworks, and tags. Multiple attributes may be selected in an *AND* relationship and multiple values within an attribute may be selected in an *OR* relationship. This enables discovering more specific kinds of models, for example, to find Computer Vision-related TensorFlow or PyTorch model code, the filters of "Computer Vision" in Domain, "TensorFlow" and "PyTorch" filters in Framework, and "Definition" filter in Lifecycle Stages would be selected. The top of the model list view contains a search feature which composes metadata together and also indexes each field as a searchable attribute directly in the CouchDB database. Queries in our system are based on the Apache Lucene syntax (Lucene 2018) which allows for complex queries such as number ranges and logical *AND* or *OR*. For example, typing `deployability:[70 TO Infinity]` into the search box will return all models where the deployability metric is 70% or greater. 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

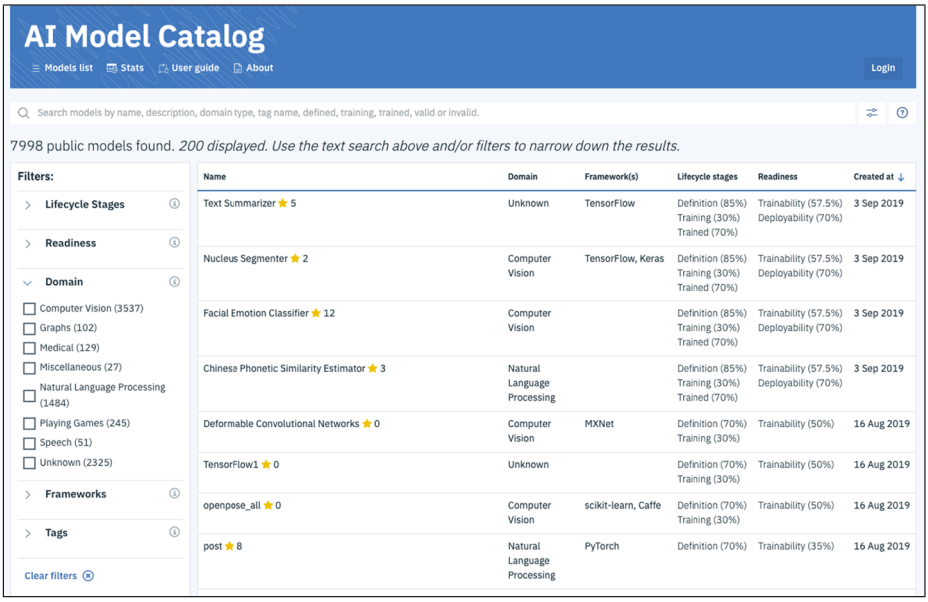

Fig. 8 Catalog main page with list of available models, search bar and filters

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, reference, and schema presence. Search queries are also composable, allowing for complex queries. For example, a query that finds a model that contains the general keyword of "ResNet" but is specifically written in the TensorFlow framework is `resnet AND framework:tensorflow`.

Once a user selects a model, they can use the individual model detail view (Fig. 9) to assess the model for reuse. Our system also provides features to assist in reusing a chosen model. Details shown include information such as tags, extraction source, authors, license, and references to academic papers or blogs. We also display the model's applicable lifecycle stages 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. The model detail view also contains the trainability and/or deployability metrics described earlier and recommendations for steps needed to train or deploy a given model. The steps are based on the readiness metrics described earlier and the recommended steps are missing metadata properties described in prose. For example, a model may be missing the *hyperparameters* property and the corresponding suggestion to increase trainability is to "define hyperparameters for training."

At the top navigation bar, there is a link to the stats page that shows visualizations of the cataloged data. As seen in Fig. 10, the page contains charts of the distribution of repositories by category (source of data, as seen in Table 1), a pie chart of the number of datasets detected per repository, and a bar chart of the distribution of the domains detected. As seen in Fig. 11, the page also includes charts summarizing model lifecycle stages and readiness metrics,

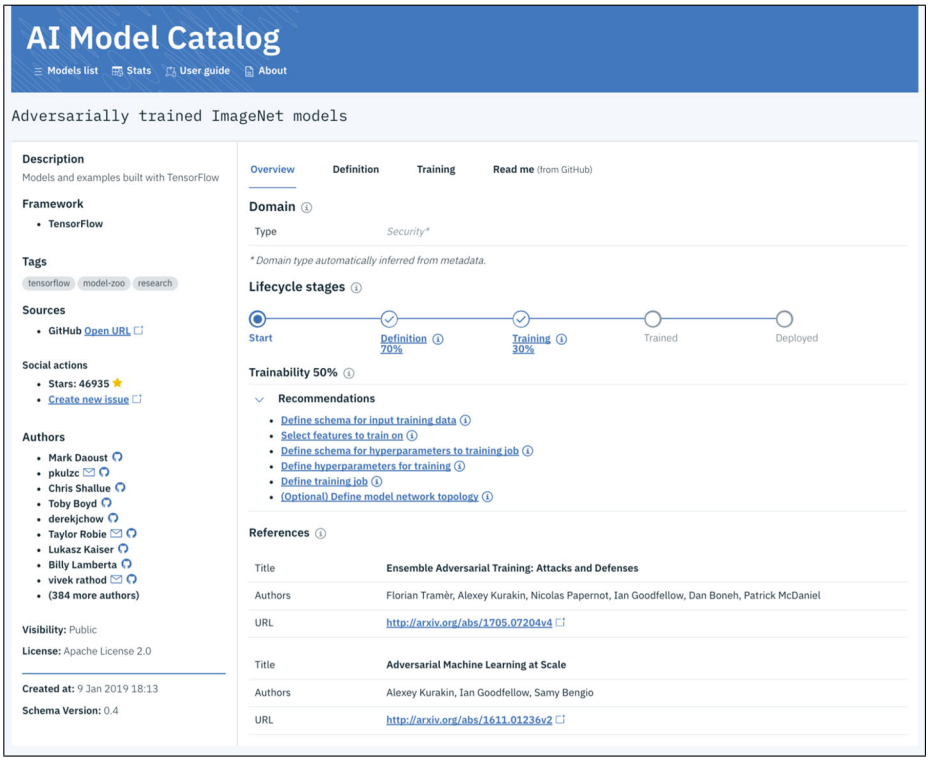

Fig. 9 Catalog individual model detail page

as well as radar charts of the AI frameworks extracted and programming languages used. Additionally there is a bar chart of open source licenses, a word-cloud of tags, and a bar chart that summarizes the count of attributes such as presence of Python code, README files, references, arXiv papers, datasets, and h5 files. This page also offers the feature of filtering the data using the same search language of the models list page, for example filtering by "ResNet" will update all of the charts to only use models related to ResNet.

7 Discussion

Our preliminary analyses and example catalog tool are intended as demonstrations of what is possible using a large collection of enhanced, standardized model metadata. AIMMX lets us envision studies similar to our preliminary analyses at much larger scales. Similarly, we use metadata from GitHub and there are a number of studies that look at collaboration and social coding on that platform (e.g. Dabbish et al. 2012; Gonzalez et al. 2020). Enhancing that metadata with AI model-specific metadata may enable us to better understand collaboration patterns in AI development. For example, Gonzalez et al. (2020) examine authorship and interaction patterns in AI-related repositories. Our enhanced metadata may enable us to understand how interaction differs between domains such as Computer Vision or NLP. We can also envision other tools that leverage standardized model metadata such as management dashboards or DevOps tools.

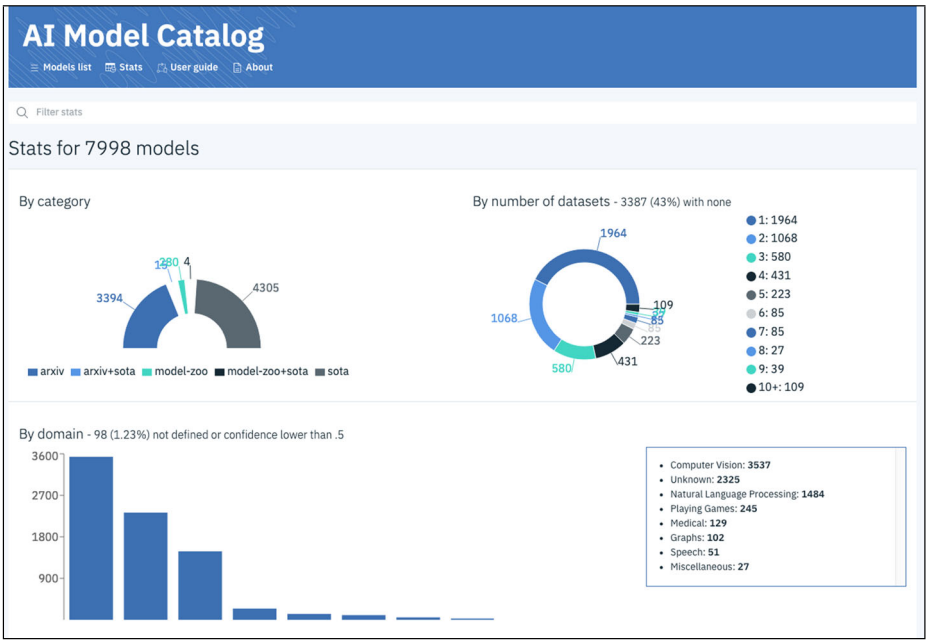

Fig. 10 Catalog stats page top

7.1 Metadata for Reuse

We envision that besides reasoning over a multitude of AI models, AIMMX also supports understanding and reusing individual models by automatically providing more needed context. Reproducibility for AI experiments is known to be poor (Gundersen and Kjensmo 2017), and the results of our preliminary analysis support this. Part of this may be due to inherent difficulties in fully documenting AI models: the entangled nature of components (Sculley et al. 2015) means reusing a given model is not just providing code but also the dataset, hyperparameters, features, software dependencies with versions, etc. This burden of the "extra work" to make the model reusable often falls on the data scientists. Often, this extra work does not directly benefit these data scientists (Trainer et al. 2015), perhaps explaining the low rates of reproducibility. We envision that a library like AIMMX can automatically perform some of this extra work or at least nudge the data scientist towards completing it via metrics such as readiness. For example, future work may provide an extension to a data scientist's IDE or Jupyter notebook displaying *trainability* and give hints on how to provide documentation to improve that metric, similar to our example catalog tool. Understanding how to reuse a model may also enable better engineering practices in AI development such as using DevOps-style tools for AI development (Hummer et al. 2019). One such integration may be to understand what datasets a model uses and then continuously check that code changes correctly converge on a portion of that dataset.

7.2 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 (Miao et al. 2016; Sethi et al. 2018)

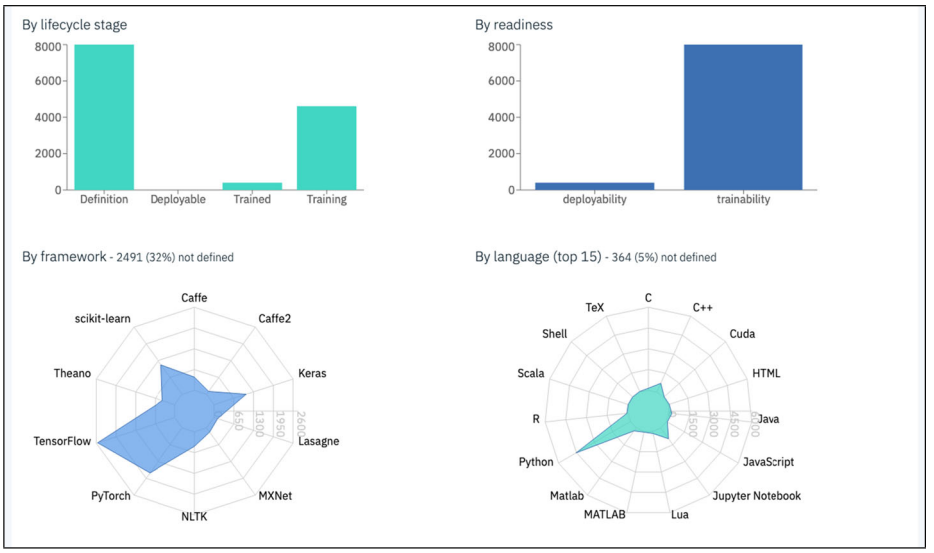

Fig. 11 Catalog stats page sequence

or user intervention (Vanschoren et al. 2014; Vartak et al. 2016), the current version only supports software repositories in GitHub. Mining GitHub has its own potential challenges (Kalliamvakou et al. 2014) 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 non-deep machine learning models. However, this is mitigated by our evaluation dataset that contains at least 1,139 non-deep 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, readiness, 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. For the readiness evaluation, while the researcher did not have access to the readiness score, they did know which partition each repository belonged to which potentially introduces bias. We attempt to mitigate this bias by describing the criteria needed to train or deploy rather than rely purely on domain knowledge. 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 at the time. This difficulty inspired us to implement the AI identification module.

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.

8 Related Work

This section discusses related work on studying Software and AI Development, model metadata mining and inference, and model catalogs.

8.1 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. (2016) 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. (2019) 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 model. 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. (2015) 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. (2019) 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 Wan et al. 2019; Breck et al. 2019) from software repositories. Wan et al. (2019) 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. (2019) 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 and Kjensmo (2017) 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.

8.2 Model Metadata Mining and Inference

Machine learning has had a close and long relationship with data mining (Witten et al. 2016), so it is natural that data mining techniques are applied to machine learning and AI models to analyze and enhance them. Sethi et al. (2018) extracted network topologies from certain diagrams in academic papers about deep learning models. Vaziri et al. (2017) extracted conversational agents from web API specifications. Machine learning experiment management tools (Tsay et al. 2018; Vartak et al. 2016) 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 (MLFlow 2019). 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 (Ma et al. 2018). Baudart et al. (2020) mine natural-language documentation, and Rak-amnouykit et al. (2021) mine Python code, to extract metadata for use in AutoML. Their work focuses on hyperparameter schemas and is thus complementary to our work, which extracts several other pieces of metadata. Projects like ML-Schema (Publio et al. 2018), an ontology for machine learning algorithms, datasets, and experiments, have identified a lack of interoperability between machine learning platforms. Our solution was to develop a standardized model metadata schema that focuses on a high-level and contextual view of AI models. This is in contrast to similar efforts such as ONNX (ONNX 2017), PMML (Guazzelli et al. 2009), ML-Schema (Publio et al. 2018), or PFA (Pivarski et al. 2016) 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. Szegedy et al. 2017), 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 instead describe in detail the neural network layers and its weights and biases. Thus, the metadata we extract is complementary with these network representation formats. Gonzalez et al. (2020) analyze ownership, authorship, and interaction patterns within a dataset of 5,224 AI-related GitHub repositories, which is similar to our evaluation dataset except that it includes AI tools/applications as well as model repositories.

8.3 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 (Vartak et al. 2016), 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 (Miao et al. 2016), which focuses on managing results and versions

of deep learning models. Their system includes a discovery system with a model comparison and ranking feature (Miao et al. 2017). In contrast, OpenML (Vanschoren et al. 2014) focuses on cataloging datasets and machine learning tasks with the intention of promoting collaboration between data scientists. The Machine Learning Bazaar (Smith et al. 2019) contains hand-curated metadata for a collection of models for the purpose of AutoML. Similarly, the Lale library (Baudart et al. 2020, 2021) contains model metadata for AutoML, with a notion of lifecycle stages including planned (pre-training), trainable, and trained. 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 (Code 2020). We also use this website as a source of ground truth data for our domain inference model.

9 Conclusions

This paper describes AIMMX, our AI Model Metadata eXtractor for software repositories. We intend AIMMX as a step towards furthering engineering support for AI development through providing standardized metadata for existing AI models that can be acted upon through analysis or tools. We offer the AIMMX extractor library itself as the primary contribution of this work, along with an evaluation dataset of 7,998 public models and two examples of what is possible using our tools: an exploratory analysis of reproducibility for the models in our dataset and a catalog tool to discover and manage models. Our vision is that AIMMX and the metadata it extracts are a step both towards managing models at scale and towards adapting mining software repositories techniques and approaches to AI models.

Appendix A: 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
  - 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

References

Ajv (2018) Ajv: another JSON schema validator. https://ajv.js.org/ (Retrieved September 2018)

Amershi S, Begel A, Bird C, DeLine R, Gall H, Kamar E, Nagappan N, Nushi B, Zimmermann T (2019) Software engineering for machine learning: a case study. In: International conference on software engineering: software engineering in practice (ICSE-SEIP), pp 291–300. https://doi.org/10.1109/ICSE-SEIP.2019.00042

Apache (2019) Apache CouchDB. https://couchdb.apache.org. Accessed 21 Jan 2019

Archive G (2021) GH Archive. https://www.gharchive.org/. Accessed 27 Oct 2021

arXiv (1991) arXiv.org e-Print archive. https://arxiv.org/. Accessed 13 Mar 2020

arXiv (2018) arXiv.org help - arXiv API. https://arxiv.org/help/api/index. Accessed 13 Mar 2020

Augustsson L (1998) Cayenne—a language with dependent types. In: International conference on functional programming (ICFP), pp 239–250. http://doi.acm.org/10.1145/289423.289451

Bangash AA, Sahar H, Chowdhury S, Wong AW, Hindle A, Ali K (2019) What do developers know about machine learning: a study of ML discussions on StackOverflow. In: Conference on mining software repositories (MSR), pp 260–264. https://doi.org/10.1109/MSR.2019.00052

Baudart G, Hirzel M, Kate K, Ram P, Shinnar A (2020) Lale: consistent automated machine learning. In: KDD workshop on automation in machine learning (AutoML@KDD). arXiv:2007.01977

Baudart G, Hirzel M, Kate K, Ram P, Shinnar A, Tsay J (2021) Pipeline combinators for gradual autoML. In: Advances in neural information processing systems (neurIPS)

Baudart G, Kirchner P, Hirzel M, Kate K (2020) Mining documentation to extract hyperparameter schemas. In: ICML Workshop on automated machine learning (autoML@ICML). arXiv:2006.16984

Braiek HB, Khomh F, Adams B (2018) The Open-Closed principle of modern machine learning frameworks. In: Conference on mining software repositories (MSR), pp 353–363

Breck E, Polyzotis N, Roy S, Whang SE, Zinkevich M (2019) Data validation for machine learning. In: Conference on systems and machine learning (sysML)

Chelba C, Mikolov T, Schuster M, Ge Q, Brants T, Koehn P (2013) One billion word benchmark for measuring progress in statistical language modeling. CoRR arXiv:1312.3005

Code PW (2020) Papers with code: the latest in machine learning. https://paperswithcode.com. Accessed 13 Mar 2020

Conneau A, Schwenk H, Cun Y, Barrault L (2017) Very deep convolutional networks for text classification. In: Long papers—continued, 15th conference of the European chapter of the Association for Computational Linguistics, EACL 2017—Proceedings of conference. Association for Computational Linguistics (ACL), pp 1107–1116. Publisher Copyright: © 2017 Association for Computational Linguistics; 15th

Conference of the European Chapter of the Association for Computational Linguistics, EACL 2017; Conference date: 03-04-2017 Through 07-04-2017
Dabbish L, Stuart C, Tsay J, Herbsleb J (2012) Social coding in GitHub: transparency and collaboration in an open software repository. In: Conference on computer supported cooperative work (CSCW), pp 1277–1286. https://doi.org/10.1145/2145204.2145396
Devlin J, Chang MW, Lee K, Toutanova K (2018) Bert: pre-training of deep bidirectional transformers for language understanding. arXiv:1810.04805
GitHub (2016) GitHub API v3 | GitHub Developer Guide. https://developer.github.com/v3/. Accessed 13 Mar 2020
GitHub (2020) The world's leading software development platform–GitHub. https://github.com/. Accessed 13 Mar 2020
Gonzalez D, Zimmermann T, Nagappan N (2020) The state of the ml-universe: 10 years of artificial intelligence & machine learning software development on github. In: Proceedings of the 17th international conference on mining software repositories, MSR '20. Association for Computing Machinery, New York, pp 431–442. https://doi.org/10.1145/3379597.3387473
Gousios G (2013) The ghtorrent dataset and tool suite. In: Proceedings of the 10th working conference on mining software repositories, MSR '13. IEEE Press, Piscataway, pp 233–236. http://dl.acm.org/citation.cfm?id=2487085.2487132
Graves TL, Karr AF, Marron JS, Siy H (2000) Predicting fault incidence using software change history. IEEE Trans Softw Eng 26(7):653–661. https://doi.org/10.1109/32.859533
Guazzelli A, Zeller M, Lin WC, Williams G et al (2009) Pmml: an open standard for sharing models. R J 1(1):60–65
Gundersen OE, Kjensmo S (2017) State of the art: reproducibility in artificial intelligence. In: Conference on artificial intelligence (AAAI)
He K, Zhang X, Ren S, Sun J (2016) Deep residual learning for image recognition. In: Proceedings of the IEEE conference on computer vision and pattern recognition (CVPR)
Hill C, Bellamy R, Erickson T, Burnett M (2016) Trials and tribulations of developers of intelligent systems: a field study. In: Symposium on visual languages and human-centric computing (VL/HCC), pp 162–170
Hummer W, Muthusamy V, Rausch T, Dube P, El Maghraoui K, Murthi A, Oum P (2019) ModelOps: cloud-based lifecycle management for reliable and trusted AI. In: 2019 IEEE International conference on cloud engineering (IC2e), pp 113–120. https://doi.org/10.1109/IC2E.2019.00025
IBM (2020) Watson Discovery product page. https://www.ibm.com/cloud/watson-discovery. Accessed 12 Nov 2020
Internet Engineering Task Force (2018) JSON Schema specification. http://json-schema.org/specification.html. (Retrieved September 2018)
Kalliamvakou E, Gousios G, Blincoe K, Singer L, German DM, Damian D (2014) The promises and perils of mining GitHub. In: Conference on mining software repositories (MSR), pp 92–101. http://doi.acm.org/10.1145/2597073.2597074
Kim M, Zimmermann T, DeLine R, Begel A (2016) The emerging role of data scientists on software development teams. In: International conference on software engineering (ICSE), pp 96–107. http://doi.acm.org/10.1145/2884781.2884783
Lucene A (2018) https://lucene.apache.org/. Accessed 23 Feb 2018
Ma Y, Fakhoury S, Christensen M, Arnaoudova V, Zogaan W, Mirakhorli M (2018) Automatic classification of software artifacts in Open-Source applications. In: Conference on mining software repositories (MSR), pp 414–425
Menzies T, Zimmermann T (2013) Software analytics: so what? IEEE Softw 30(4):31–37. https://doi.org/10.1109/MS.2013.86
Miao H, Li A, Davis LS, Deshpande A (2016) ModelHub: towards unified data and lifecycle management for deep learning. CoRR. arXiv:1611.06224
Miao H, Li A, Davis LS, Deshpande A (2017) On model discovery for hosted data science projects. In: Workshop on data management for end-to-end machine learning, DEEM'17, pp 6:1–6:4. http://doi.acm.org/10.1145/3076246.3076252
MLFlow (2019) MLFlow—a platform for the machine learning lifecycle. https://mlflow.org/. Accessed 13 Mar 2020
ONNX (2017) ONNX. https://onnx.ai/. Accessed 13 Mar 2020
Ostrand TJ, Weyuker EJ, Bell RM (2005) Predicting the location and number of faults in large software systems. IEEE Trans Softw Eng 31(4):340–355. https://doi.org/10.1109/TSE.2005.49
Paszke A, Gross S, Massa F, Lerer A, Bradbury J, Chanan G, Killeen T, Lin Z, Gimelshein N, Antiga L, Desmaison A, Kopf A, Yang E, DeVito Z, Raison M, Tejani A, Chilamkurthy S, Steiner B, Fang L, Bai J, Chintala S (2019) Pytorch: an imperative style, high-performance deep learning library. In: Advances in

neural information processing systems 32. Curran Associates, Inc., pp 8024–8035. http://papers.neurips.cc/paper/9015-pytorch-an-imperative-style-high-performance-deep-learning-library.pdf
Pezoa F, Reutter JL, Suarez F, Ugarte M, Vrgoč D. (2016) Foundations of JSON schema. In: International conference on world wide web (WWW), pp 263–273. https://doi.org/10.1145/2872427.2883029
Pimentel JF, Murta L, Braganholo V, Freire J (2019) A large-scale study about quality and reproducibility of jupyter notebooks. In: Conference on mining software repositories (MSR), pp 507–517. https://doi.org/10.1109/MSR.2019.00077
Pivarski J, Bennett C, Grossman RL (2016) Deploying analytics with the portable format for analytics (pfa). In: Conference on knowledge discovery and data mining (KDD), pp 579–588. http://doi.acm.org/10.1145/2939672.2939731
Publio GC, Esteves D, ŁAwrynowicz A, Panov P, Soldatova L, Soru T, Vanschoren J, Zafar H (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
Rak-amnouykit I, Milanova A, Baudart G, Hirzel M, Dolby J (2021) Extracting hyperparameter constraints from code. In: ICLR Workshop on security and safety in machine learning systems (secML@ICLR). https://aisecure-workshop.github.io/aml-iclr2021/papers/18.pdf
Rodríguez C, Baez M, Daniel F, Casati F, Trabucco JC, Canali L, Percannella G (2016) REST APIS: a large-scale analysis of compliance with principles and best practices. In: International conference on web engineering (ICWE), pp 21–39. https://doi.org/10.1007/978-3-319-38791-8_2
Ronneberger O, Fischer P, Brox TNavab N, Hornegger J, Wells WM, Frangi AF (eds) (2015) U-Net: convolutional networks for biomedical image segmentation. Springer International Publishing, Cham
Sculley D, Holt G, Golovin D, Davydov E, Phillips T, Ebner D, Chaudhary V, Young M, Crespo JF, Dennison D (2015) Hidden technical debt in machine learning systems. In: Conference on neural information processing systems (NIPS), pp 2503–2511
Sethi A, Sankaran A, Panwar N, Khare S, Mani S (2018) Dlpaper2code: auto-generation of code from deep learning research papers. In: Conference on artificial intelligence (AAAI), pp 7339–7346. https://www.aaai.org/ocs/index.php/AAAI/AAAI18/paper/view/17100
Shah N (2019) ARXIV data from 24,000+ papers Version 2. https://www.kaggle.com/neelshah18/arxivdataset/home. Accessed 15 Jan 2019
Shaikh S, Vishwakarma H, Mehta S, Varshney KR, Ramamurthy KN, Wei D (2017) An end-to-end machine learning pipeline that ensures fairness policies. In: Data for good exchange. https://arxiv.org/abs/1710.06876
Smith MJ, Sala C, Kanter JM, Veeramachaneni K (2019) The machine learning bazaar: harnessing the ML ecosystem for effective system development. https://arxiv.org/abs/1905.08942
Szegedy C, Ioffe S, Vanhoucke V, Alemi AA (2017) Inception-v4, Inception-ResNet and the impact of residual connections on learning. In: Conference on artificial intelligence (AAAI)
Trainer EH, Chaihirunkarn C, Kalyanasundaram A, Herbsleb JD (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), pp 417–430. http://doi.acm.org/10.1145/2675133.2675172
Tramèr F, Zhang F, Juels A, Reiter MK, Ristenpart T (2016) Stealing machine learning models via prediction APIs. In: USENIX security symposium, pp 601–618
Tsay J, Mummert T, Bobroff N, Braz A, Hirzel M (2018) Runway: machine learning model experiment management tool. In: Conference on systems and machine learning (sysML)
Tsay J, Braz A, Hirzel M, Shinnar A, Mummert T (2020) Aimmx: artificial intelligence model metadata extractor. In: Proceedings of the 17th international conference on mining software repositories, MSR '20. Association for Computing Machinery, New York, pp 81–92. https://doi.org/10.1145/3379597.3387448
Vanschoren J, van Rijn JN, Bischl B, Torgo L (2014) openML: networked science in machine learning. SIGKDD Explor Newsl 15(2):49–60. http://doi.acm.org/10.1145/2641190.2641198
Vartak M, Subramanyam H, Lee WE, Viswanathan S, Husnoo S, Madden S, Zaharia M (2016) ModelDB: a system for machine learning model management. In: Workshop on human-in-the-loop data analytics (HILDA), pp 14:1–14:3. http://doi.acm.org/10.1145/2939502.2939516
Vaswani A, Shazeer N, Parmar N, Uszkoreit J, Jones L, Gomez AN, Kaiser LU, Polosukhin I (2017) Attention is all you need. In: Guyon I, Luxburg UV, Bengio S, Wallach H, Fergus R, Vishwanathan S, Garnett R (eds) Advances in neural information processing systems, vol 30. Curran Associates. https://proceedings.neurips.cc/paper/2017/file/3f5ee243547dee91fbd053c1c4a845aa-Paper.pdf
Vaziri M, Mandel L, Shinnar A, Siméon J, Hirzel M (2017) Generating chat bots from web api specifications. In: Symposium on new ideas, new paradigms, and reflections on programming and software (Onward!), pp 44–57. http://doi.acm.org/10.1145/3133850.3133864
Wan Z, Xia X, Lo D, Murphy GC (2019) How does machine learning change software development practices? IEEE Trans Softw Eng 1. https://doi.org/10.1109/TSE.2019.2937083

Witten IH, Frank E, Hall MA, Pal CJ (2016) Data mining: practical machine learning tools and techniques. Morgan Kaufmann

Affiliations

Jason Tsay[1] · Alan Braz[2] · Martin Hirzel[1] · Avraham Shinnar[1] · Todd Mummert[1]

Alan Braz
alanbraz@br.ibm.com

Martin Hirzel
hirzel@us.ibm.com

Avraham Shinnar
shinnar@us.ibm.com

Todd Mummert
mummert@us.ibm.com

[1] IBM Research, Yorktown Heights, NY, USA

[2] IBM Research Brazil, São Paulo, Brazil