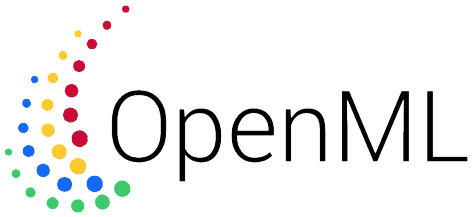
July 3, 2019

Architectural Design Document

Version 1.0.0



Members
Andrei Danila | 1034559
Bogdan Enache | 1035066
Gergana Goncheva | 1037010
Loïc Alexander Hijl | 1002745
Adrian-Stefan Mares | 0993873
Veselin Minev | 1014541
Thanh-Dat Nguyen | 1036672
Antoine Labasse Lutou Nijhuis | 1016657
Claudiu-Teodor Nohai | 1038442
Dragos Mihai Serban | 1033859
Tsvetan Zahariev | 1035269
Sonya Zarkova | 1034611

Project managers
Yuxuan Zhang
Stefan Tanja

Supervisor
Erik Luit
Ion Barosan

Customer
Joaquin Vanschoren

Abstract

This document contains the architectural design, design decisions and components related to the Deep Learning (DL) Extension Library for OpenML, developed by the OpenML Support Squad. The architecture adheres to the requirements layed out in the Software Requirements Document (SRD) [1] as well as those described in the User Requirements Document (URD) [2]. This document complies with the Software Engineering Standard, defined by the European Space Agency (ESA) [3].

Contents

Document Status Sheet

Document Title: Architectural Design Document
Identification: ADD/1.0.0
Version: 1.0.0
Authors: A.Danila, B.Enache, G.Goncheva, L.A.Hijl, T.Nguyen, A.L.L.Nijhuis, T.Zahariev, S.Zarkova

Document History

Version	Date	Author(s)	Summary
0.0.1	25-04-2019	T. Zahariev	Created initial document
0.0.2	01-05-2019	L.A. Hijl	Added chapters and layout
0.0.3	15-05-2019	All authors	First draft
0.0.4	01-06-2019	All authors	Implemented internal feedback
0.1.0	10-06-2019	All authors	Fixed first feedback from supervisor
0.2.0	30-06-2019	All authors	Fixed second feedback from supervisor
1.0.0	01-07-2019	All authors	Final edit

1 | Introduction

1.1 Purpose

This Architectural Design Document (ADD) describes the architectural design of the DL Extension Library built by the OpenML Support Squad. It gives an overview of the library and the design decisions related to its architecture. Furthermore, a description of the individual components of the software product is provided. The dependencies of these components are discussed, as well as their relation to the software requirements part of the SRD.
The purpose of project "Sharing deep learning models" is to create an extension for the already existent OpenML Python API and provide a visualization module. The extension, called DL Extension Library, represents the biggest part of the product that OpenML Support Squad has to deliver. DL Extension Library will enable sharing deep learning models to the OpenML platform. The visualization module allows visualization of a model's structure and performance metrics in Dash.

1.2 Scope

OpenML is an open-source online machine learning platform for sharing and organizing data, machine learning algorithms and experiments. It allows people all over the world to collaborate and build on each other's latest findings, data or results. OpenML is designed to create a seamless network that can be integrated into existing environments, regardless of the tools and infrastructure used. It allows users to upload datasets, models (called OpenML flows) and tasks for OpenML flows. It further provides support for uploading runs that combine a model with a task for it, to allow users to compare results.

The DL Extension Library is part of the OpenML Python API that will be imported by the users in their Python project. The OpenML Python API handles all interactions with the OpenML platform. Whenever necessary, the OpenML Python API calls the DL Extension Library in order to convert the deep learning model to an OpenML flow, the format supported by the OpenML platform. The OpenML Support Squad is not responsible for the development and maintenance of the OpenML Python API. The OpenML Support Squad will not modify the OpenML Python API since it is outside of the scope of the project "Sharing deep learning models".

The contribution of OpenML Support Squad consists of two parts: creating the DL Extension Library and the visualization module.

The DL Extension Library is built to provide support for the following deep learning libraries and specifications on the OpenML platform: Keras, PyTorch, Apache MXNet (MXNet) and Open Neural Network eXchange (ONNX). These libraries have already been created by their respective developers and, as such, are not part of this project and are not under the responsibility of OpenML Support Squad.

The visualization module is built to serve as a prototype which can be integrated into the website. It will allow users to visualize their models and model results.

1.3 Definitions and Abbreviations

1.3.1 Definitions

DL Extension Library	The product to be created in order to convert the deep learning models and the model transfer specifications (i.e. ONNX and MLflow) into a format supported by the OpenML platform.
OpenML Main REST API	An interface supporting the interaction between the OpenML Python API and the OpenML platform. "Has all main functions to download OpenML data or share your own" [4].
OpenML Python API	"A connector to the collaborative machine learning platform OpenML.org. The OpenML Python package allows to use datasets and tasks from OpenML together with scikit-learn and share the results online" [4].
OpenML flow	The representation of untrained machine learning models in the OpenML platform.
Apache MXNet	"Apache MXNet is an open-source deep learning software framework, used to train, and deploy deep neural networks. It is scalable, allowing for fast model training, and supports multiple programming languages (C++, Python, Julia, Matlab, JavaScript, Go, R, Scala, Perl, and Wolfram Language)" [5].
Dash	"Dash is a user interface library for creating analytical web applications" [6].
Dataset	"Datasets are pretty straight-forward. They simply consist of a number of rows, also called instances, usually in tabular form" [4].
Deep learning model	A neural network with more than three layers. Deep learning is a subset of machine learning.
JSON	"JSON (JavaScript Object Notation) is a lightweight data-interchange format" [7].
Keras	"Keras is a high-level neural networks API, written in Python and capable of running on top of TensorFlow, CNTK, or Theano" [8].
Machine learning model	A model that performs a “task without explicit instructions, relying on patterns and inference” [9] through learning from data.
Model	"A mathematical representation of a real-world process" [10].
Neural Network	"Computing systems vaguely inspired by the biological neural networks that constitute animal brains." It is "a framework for many different machine learning algorithms" [11].
OpenML	An online machine learning platform for sharing and organizing data, machine learning algorithms and data experiments.
OpenML Support Squad	Name of the development team.

Pickle	The pickle module implements binary protocols for serializing and de-serializing a Python object structure [12].
Project "Sharing deep learning models"	The project on which the OpenML Support Squad is working on.
Python Package Index	"The Python Package Index (PyPI) is a repository of software for the Python programming language. It helps in finding and installing software developed and shared by the Python community" [13].
PyTorch	"An open source deep learning platform that provides a seamless path from research prototyping to product deployment" [14].
Run	"A run is a particular flow, that is algorithm, with a particular parameter setting, applied to a particular task" [4].
Sklearn	A Python library that provides machine learning algorithms.
Task	"A task consists of a dataset, together with a machine learning task to perform, such as classification or clustering and an evaluation method. For supervised tasks, this also specifies the target column in the data" [4].
TensorFlow	"TensorFlow is a free and open-source software library for dataflow and differentiable programming across a range of tasks" [15].
Torch	"Torch is an open-source machine learning library, a scientific computing framework, and a script language based on the Lua programming language" [16].
Travis Continuous Integration	"Travis CI is a hosted continuous integration service used to build and test software projects hosted at GitHub" [17].
Unittest library	"It supports test automation, sharing of setup and shutdown code for tests, aggregation of tests into collections, and independence of the tests from the reporting framework" [18].

1.3.2 Abbreviations

ADD	Architectural Design Document.
API	Application Programming Interface.
DL	Deep Learning.
ESA	European Space Agency.
MXNet	Apache MXNet.
ONNX	Open Neural Network eXchange.
SRD	Software Requirements Document.
UI	User Interface.
URD	User Requirements Document.

1.4 List of references

[1] OpenML Support Squad. *DL Extension Library, Software Requirement Document version 1.0.0.*

[2] OpenML Support Squad. *DL Extension Library, User Requirement Document version 1.0.0.*

[3] Software Standardisation and Control. *ESA software engineering standards. 1991.*

[4] OpenML. *OpenML Documentation.* URL: https://docs.openml.org/ (visited on 04/26/2019).

[5] Wikipedia. *Apache MXNet.* URL: https://en.wikipedia.org/wiki/Apache_MXNet (visited on 04/26/2019).

[6] Kyle Kelley. *Introducing Dash.* URL: https://medium.com/@plotlygraphs/introducing-dash-5ecf7191b503 (visited on 06/05/2019).

[7] *JSON.* URL: https://www.json.org/ (visited on 05/16/2019).

[8] Keras. *Keras homepage.* URL: https://keras.io/ (visited on 04/26/2019).

[9] Wikipedia. *Machine Learning.* URL: https://en.wikipedia.org/wiki/Machine_learning (visited on 04/26/2019).

[10] Joydeep Bhattacharjee. *Some Key Machine Learning Definitions.* URL: https://medium.com/technology-nineleaps/some-key-machine-learning-definitions-b524eb6cb48 (visited on 05/01/2019).

[11] Wikipedia. *Artificial Neural Network.* URL: https://en.wikipedia.org/wiki/Artificial_neural_network (visited on 04/30/2019).

[12] *Pickle.* URL: https://docs.python.org/3/library/pickle.html (visited on 05/16/2019).

[13] *Python Package Index.* URL: https://pypi.org/ (visited on 05/20/2019).

[14] PyTorch. *PyTorch homepage.* URL: https://pytorch.org/.

[15] TensorFlow. *TensorFlow wikipedia.* URL: https://en.wikipedia.org/wiki/TensorFlow (visited on 04/26/2019).

[16] *Torch (machine learning).* URL: https://en.wikipedia.org/wiki/Torch_(machine_learning) (visited on 05/15/2019).

[17] *Travis CI.* URL: https://en.wikipedia.org/wiki/Travis_CI (visited on 05/20/2019).

[18] *Unittest library.* URL: https://docs.python.org/3/library/unittest.html (visited on 05/20/2019).

[19] *OpenML Python Package Index.* URL: https://pypi.org/project/openml/ (visited on 05/15/2019).

[20] *PEP 8 - Style Guide.* URL: https://www.python.org/dev/peps/pep-0008/ (visited on 05/15/2019).

1.5 Overview

The remainder of this document contains five chapters. Chapter 2 gives an overview of the system. The chapter is split into three sections. Section 2.1 provides background information

about the system, Section 2.2 discusses its context and basic design, Section 2.3 documents the design decisions related to the system.
Chapter 3 describes the relationships between the DL Extension Library and other external systems. The relation to each of these can be found in its corresponding section. The OpenML Python API, Keras, PyTorch, MXNet, ONNX and Dash are among those external systems.
Chapter 4 presents the system design, consisting of the design method - Section 4.1, the component decomposition - Section 4.2 and the file structure - Section 4.3.
Chapter 5 considers the feasibility and the resource estimates with regards to building the DL Extension Library. Section 5.1 discusses resource requirements, while Section 5.2 focuses on performance.
Chapter 6 provides requirements traceability matrices showing the relation of software requirements to components - Section 6.1, and the relation of components to software requirements - Section 6.2.

2 | System Overview

2.1 Background

The OpenML platform has been created to give researchers and data scientists an online platform on which they can share machine learning models built with different machine learning frameworks. The platform also supports the creation of tasks, so that the results of different machine learning models can be compared on the same task. This is done in order to see how good a model is in comparison to the runs of others on the same task. Having the possibility to benchmark models is very useful, since people can choose the best ones and refine them further.

OpenML is not a platform which runs and trains machine learning models, it only deals with sharing and comparing their results. All models can be retrieved from the platform by the user. Then, they can be run and trained on a local machine. OpenML provides users with data sets, with which the machine learning models can be trained. Users can do all of the aforementioned operations using the already available OpenML Python API.

One of the shortcomings of OpenML results from not supporting deep learning frameworks, such as Keras, PyTorch and Apache MXNet, nor deep learning models specifications such as ONNX. Therefore, the goal of the OpenML Support Squad is to implement an extension to the OpenML Python API, in order for the platform to support deep learning models. More information about the functions and purpose of this extension, called DL Extension Library, can be found in sections 2.3, 2.4 and 2.5 of the SRD [1]. Furthermore, another goal of OpenML Support Squad is to implement a visualizer. This can be used to visualize run associated metrics and data, as well as the structure of a flow.

2.2 Context and Basic Design

The DL Extension Library is part of the OpenML Python API that will be imported by the user in their Python code. The OpenML Python API handles all interactions with the OpenML platform. Whenever necessary, it calls the DL Extension Library in order to convert the deep learning model to an OpenML flow, the format supported by the OpenML platform. The OpenML Support Squad is not responsible for the development and maintenance of the OpenML Python API. The OpenML Support Squad will not modify the OpenML Python API since it is outside of the scope of the project.

The visualization module is not part of OpenML and serves as a prototype which can later be integrated into the website. This module will be used whenever a flow or run needs to be visualized.

To implement the extension packages, the OpenML Support Squad will use the extension interface provided by OpenML Python API. The OpenML Support Squad is not responsible for the structure and design of this interface. This interface is further discussed in Section 2.3.6 and can be seen in figure 2.2.

OpenML Support Squad has to create the visualization module, using Dash for compatibility reasons imposed by the client.

Note, there is the assumption that the user should have a basic understanding of machine learning concepts. He can interact with the system in two ways: by using the OpenML Python API or the OpenML web interface. While the web interface is more user-friendly, its functionality is substantially reduced compared to the OpenML Python API. Hence, users are encouraged to learn to utilize the API. On the website the main available operations are uploading and downloading models. However, with the OpenML Python API other functionality such as conversion, becomes available. The data of OpenML is stored in a server connected through the OpenML Main REST API. This is the place where all the models, runs, tasks and datasets are kept.

An overview of the environment for this project can be seen in figure 2.1.

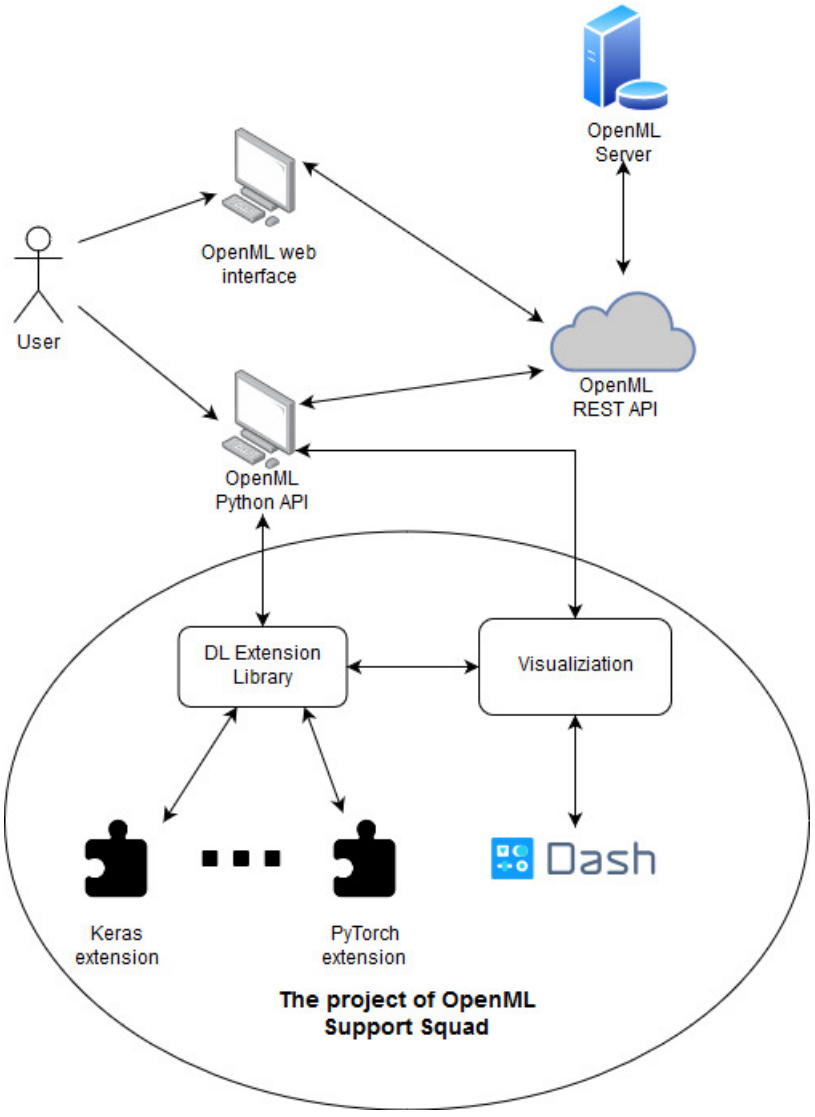


Figure 2.1: Overview diagram of the environment

2.3 Design Decisions

2.3.1 Programming language

With regard to the choice of programming language, the client already decided on this matter. One of the requirements for the project is that the DL Extension Library should be written in Python. There are several reasons for this, first of all is the existence of the OpenML Python API, which provides the functionality that the DL Extension Library extends. Another reason is the fact that Python is the preferred language in the field of machine learning.

Python is a very versatile, widely-used programming language. It is a powerful tool with a high level of readability, due to abstracting from some of the staples of programming, such as curly brackets and semi colons.

2.3.2 Managing dependencies

One of the main requests that the client has is to minimize the dependencies that will occur due to OpenML supporting multiple deep learning frameworks, such as Keras, PyTorch and Apache MXNet, and specifications such as ONNX. The only component where such dependencies can occur is in the extension library of the OpenML Python API, named DL Extension Library. This is because the DL Extension Library is the only component which interacts directly with these frameworks. The purpose of the DL Extension Library is to convert the deep learning models made by frameworks such as Keras, PyTorch and Apache MXNet, into the OpenML flow format supported by OpenML. The same goes for specifications like ONNX. More about the purpose of the DL Extension Library can be found in section 2.3 of the SRD [1].

All other components from the OpenML Python API use only the OpenML flow format to interact with machine learning models. Thus, the focus of the OpenML Support Squad is to minimize the dependencies only in the DL Extension Library since it is the only component that interacts directly with the frameworks. Failing to do so will result in the user being forced to install all the deep learning frameworks that the DL Extension Library supports just to be able to use one extension (for example Keras), even though they might not use/need all of them. Each of these frameworks is big in size and requires some time to be installed locally on the user's machine.

In order to avoid such dependencies, and thus minimize them, the OpenML Support Squad decided to have a separate package for each of the frameworks, which means that, if the user is interested only in Keras, for example, they will only have to import the Keras package from the DL Extension Library by running the following line of code: `from openml.extensions.keras import KerasExtension`.

2.3.3 Back-end integration with deep learning models

Since the OpenML platform needs to store deep learning models and their runs, the OpenML Support Squad decided, in accordance with the customer, to store the deep learning models in the OpenML flow format. In this way, the back-end of OpenML does not need to be modified in any way, because it will interact only with OpenML flows, something which it already does. The OpenML flow format will not be changed in any way. All the layers from the deep learning models will be stored as parameters in the OpenML flow, disregarding their weights since

OpenML flow stores only untrained models. This leaves the back-end and its integration with the OpenML Python API and the OpenML Main REST API outside the project's scope.

2.3.4 Deep learning formats and frameworks

There are several formats and frameworks that need to be addressed during the project, due to the nature of the assignment. A deep learning format dictates the structure of a model. Formats cannot be trained, nor can they run tasks. Frameworks on the other hand are designed with training in mind. For the purpose of the project, the following standards are relevant:

The most important standard is the OpenML flow format since OpenML is build upon this. In the assignment description it is stated that OpenML Support Squad is supposed to convert the formats and frameworks mentioned in the [2] to and from OpenML flow.

Keras is one of the most popular deep learning libraries. The fact that it is mostly built upon Google's TensorFlow and is also created by a Google engineer, makes Keras a prominent framework in this domain. Its ease of use is another reason for its wide spread appeal. It allows anyone to make a deep learning model in just a few lines of code. This also translates well for the purposes of the project, since it simplifies the serialization process.

PyTorch is a Python machine learning framework based on Torch. It is mainly maintained by Facebook and Uber. While being popular, its role and function are somewhat different from Keras. PyTorch provides a more low level interaction with deep learning models, and is not as streamlined as Keras. It is more akin to TensorFlow. This aspect proved to be quite a big factor in the implementation of our project. Due to the way PyTorch works, only serialization of sequential models is possible to be done in a safe way. If other types of models were to be implemented, serialization would introduce security risks, since the instructions found inside some methods cannot be verified to be free of malicious contents. Hence, an attacker could craft a malicious payload, which could compromise the machine of a user that would try to replicate an OpenML Run. This issue is raised by the fact that PyTorch encourages users to have their own custom layers, which contain their own forward functions.

Apache MXNet is a deep learning software framework supporting multiple programming languages, including Python, and is the framework of choice for Amazon. It is one of the most powerful frameworks thanks to the multitude of functions and freedom of options it allows. It offers a comprehensive GPU support, making training a model an easier task compared to Keras and PyTorch. Implementation wise, it is somewhere between PyTorch and Keras with regards to ease of use. This makes it a valid option for the project, allowing for an easy serialization and deserialization process.

ONNX was created because of a multi-company agreement, to serve as an universal format for neural networks. The idea behind its inception was to have an easy way to go from one framework to another. Because an ONNX specification cannot be run, the decision has been made to first convert it to an MXNet model before running it. MXNet has been chosen due to the fact that it is the framework which has the most support for ONNX. All the other supported frameworks (Keras, PyTorch) of the DL Extension Library have implementation flaws in them that hinder their use with ONNX. This is outside of the project's scope.

2.3.5 Continuous Integration and Deployment

To make the process of writing the code more efficient, the OpenML Support Squad decided to use Travis Continuous Integration (CI) to develop the DL Extension Library and the visualization module. The Travis CI is configured to check for code formatting. It also tests for functional correctness when running all test cases for the DL Extension Library, all the other components from the OpenML Python API and the visualization module. This is done before deploying the changes to the OpenML Python API. The test cases are performed using the Unittest library. The team has created all the test cases for the DL Extension Library and the visualization module, but the test cases for the other components of the OpenML Python API were already in place at the start of the project.

The DL Extension Library is part of the OpenML Python API. To deploy the DL Extension Library in the OpenML Python API, a new version of the OpenML Python API, which contains the code of DL Extension Library, needs to be deployed. For deploying the new version of the OpenML Python API, Python Package Index (PyPi) can be used. The OpenML Python API has already been registered in the Python Package Index, under the name OpenML [19]. To publish the OpenML Python API a `setup.py` file needs to be configured. This file already contained all the needed dependencies required for using OpenML Python API, thus OpenML Support Squad is not responsible for them. To minimize dependencies, OpenML Support Squad will not include the Keras, PyTorch, ONNX, MXNet and other frameworks in the required dependencies inside `setup.py`, as explained in 2.3.2. Instead, they will be listed as extra dependencies that can be installed if the user wishes to use them. For example, the user can install OpenML with the needed dependencies only for Keras, so that they can use Keras with OpenML Python API.

2.3.6 Extending the OpenML Python API

The OpenML Python API was not created by the OpenML Support Squad and thus it is outside of the scope of project "Sharing deep learning models". The OpenML Python API already contains an interface called *Extension*, further referred as "extension interface". The purpose of this interface is to allow future support for deep learning frameworks such as Keras, PyTorch, etc. Because this extension interface is already present, the OpenML Support Squad has to use it when implementing the extension packages (for Keras, PyTorch, etc). The main task of these packages is to create a conversion library that can make several deep learning formats and frameworks compatible with OpenML. During the development of the project, OpenML Support Squad will not modify OpenML Python API, but rather extend it. The extension interface that the OpenML Python API provides can be seen in figure 2.2. The OpenML Support Squad is not responsible neither for its structure nor for its design.

2.3.7 Visualization

Since OpenML already uses Dash, the visualization module has to be created using Dash. This makes the module easier to integrate into the website and keeps everything stylistically consistent.

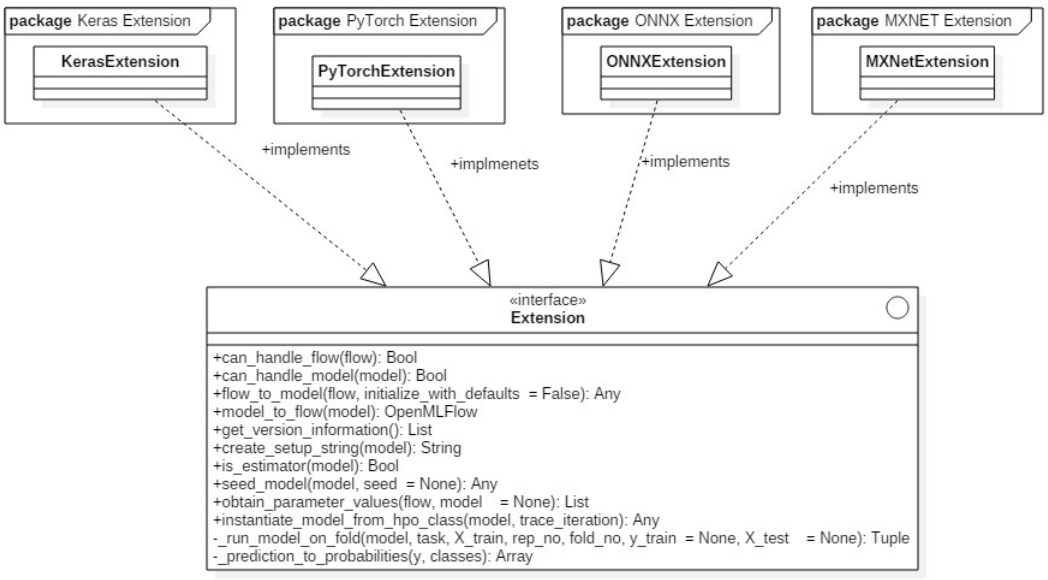


Figure 2.2: Extension interface provided by OpenML Python API

3 | System Context

This section describes the environment in which the DL Extension Library and the visualization module operate, which can be seen on figure 3.1. The environment mainly consists of the OpenML Python API itself, deep learning libraries (such as Keras) used for direct conversions, the ONNX specification and the libraries used to facilitate visualization.
Note: The contents of the tables in the following subsections are derived from existing documentation concerning the discussed frameworks and formats.

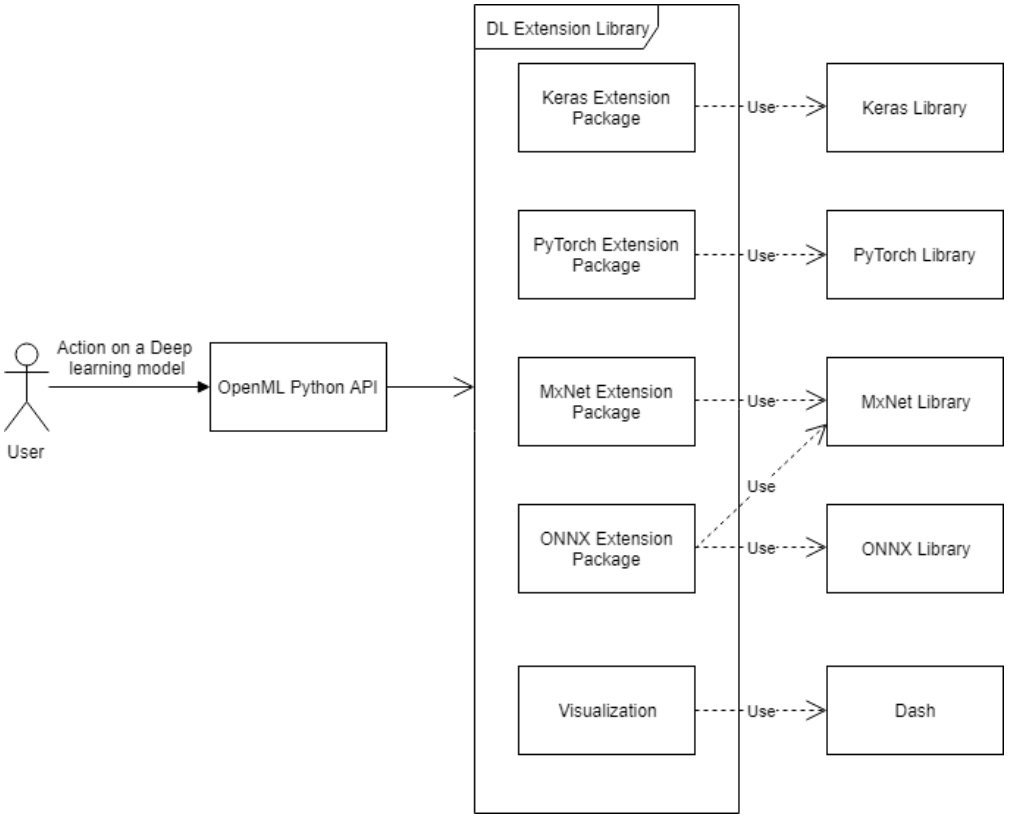


Figure 3.1: Diagram of the environment of the DL Extension Library

3.1 OpenML Python API

OpenML is a platform that facilitates sharing of machine learning models. The OpenML Python API is provided to connect to OpenML. The DL Extension Library takes a model and its parameters/components as input and transforms them into an OpenML flow. To extract a model from a given OpenML flow, the DL Extension Library reverses this process.

3.1.1 OpenMLFLow

The OpenML flow provides the functionality necessary to manipulate OpenML flows and transform machine learning models from/to an OpenML flow. These properties are shown in table 3.1.

Property	Type	Description
parameters	Dictionary	Mapping from parameter name to the parameter default value.
components	Dictionary	Mapping from component identifier to an OpenML flow object.
dependencies	String	A list of dependencies necessary to run the flow.
external_version	String	Version number of the software the flow is implemented in. Is used together with the attribute "name" as a uniquer identifier of the flow.
name	String	Name of the flow.
flow_id	Integer	Flow ID is assigned by the server for identification of a flow.
get_structure	Function	Returns for each sub-component of the flow the path of identifiers that should be traversed to reach this component.

Table 3.1: OpenMLFlow properties used by DL Extension Library

3.1.2 Other

Other functions and parameters used from OpenML Python API. They are shown in table 3.2.

Property	Type	Description
setups.OpenMLParameter .flow_name	String	The name of the flow to which this parameter is associated.
flows.functions._check _flow_for_server_id	Function	Checks if all components of a flow are uploaded to the server.

Table 3.2: Other properties used by DL Extension Library related to OpenMLFlow

3.2 Keras

The most widely-used Deep Learning library supported by the extension is Keras. Deep learning models built using Keras can be directly converted into an OpenML flow. The DL Extension Library interacts with the Keras library directly, making use of its library functions to provide

the intended functionality. The Keras library properties are given in table 3.3.

Property	Type	Description
layers.deserialize	Function	Instantiates a layer from a dictionary.
optimizers.deserialize	Function	A deserialization function.
backend.backend	Function	Publicly accessible method for determining the current backend.
backend.argmax	Function	Returns the index of the maximum value along an axis.
backend.eval	Function	Evaluates the value of a variable.
compile	Function	Configures the model for training.
get_config	Function	Gets configuration of a Keras model.
optimizer	String	Name of optimizer.

Table 3.3: Keras properties used by DL Extension Library

3.3 PyTorch

PyTorch is a DL library that, which allows much more customization, although it less popular than Keras. The drawback is that it is harder to use due to this increased customization. To provide the intended functionality, such as processing the dataset and training the model, the DL Extension Library uses the PyTorch functions, listed in table 3.4.

Property	Type	Description
cuda.is_available	Function	Checks whether the user's GPU is compatible with NVIDIA's CUDA platform.
tensor.cuda	Function	Enable CUDA computing on the tensor.
from_numpy	Function	Transforms an n-dimensional array into a tensor.
utils.data.TensorDataset	Function	Transforms tensors into a dataset for training the model.
utils.data.DataLoader	Function	Combines a dataset and a sampler, and provides single or multiprocess iterators over the dataset.
tensor	Function	Instantiates a tensor.

Table 3.4: PyTorch properties used by DL Extension Library

3.4 ONNX

ONNX is a universal format for specifying deep learning models. It is used by OpenML to allow conversions of numerous models. Two of its library functions are called upon within the ONNX

extension. They are shown in table 3.5.

Property	Type	Description
ModelProto	Function	Returns an empty ModelProto, a top-level container for bundling a deep learning model.
save	Function	Saves the ONNX specified model.

Table 3.5: ONNX properties used by DL Extension Library

3.5 MXNet

The MXNet library is used not only by the MXNet extension, but also by the ONNX extension. ONNX uses MXNet to run a model, since ONNX is not a DL library itself. The Gluon library in MXNet is used to build and train the ONNX models. The Autograd package from MXNet is used for manipulation of N-dimensional arrays. The functions depicted in table 3.6 are used for running models in the extensions mentioned above.

Property	Type	Description
init.Uniform	Function	Creates an initializer that assigns random values with uniform distribution to weights and biases of a model.
nd.argmax	Function	Returns indices of the maximum values along an axis.
nd.array	Function	Instantiates an array from any object exposing the array interface.
gluon.Train	Function	Defines trainer using optimizer from configuration.
Autograd.record	Function	Returns an autograd recording scope context to be used in "with" statement and captures code that needs gradients to be calculated.
contrib.onnx.import_to_gluon	Function	Imports the ONNX model files, passed a parameter, into a Gluon object.
sym.var	Function	Creates a symbolic variable with specified name
symbol.load_json	Function	Loads symbol from JSON string. Symbol is an interface necessary to construct a neural network.
gluon.SymbolBlock	Function	A constructor used to create a block from a symbol.
init.Normal	Function	Initializes random weights sampled from a normal distribution
gluon.data.arraydataset	Function	Function that combines multiple datasets.

gluon.data.DataLoader	Function	Loads data from a dataset and returns mini-batches of data.
gluon.trainer	Function	Applies an optimizer on a set of parameters.
autograd.record	Function	Records computation history.

Table 3.6: MXNet properties used by DL Extension Library

3.6 Google Protobuf

Google Protobuf is a method of serializing structured data. It is used by the ONNX extension to serialize and deserialize models. In doing so, utilizing the properties listed in table 3.7.

Property	Type	Description
json_format.ParseDict	Function	Parses a JSON dictionary representation into a message.
json_format.MessageToDict	Function	Converts protobuf message to a dictionary.

Table 3.7: Google Protobuf properties used by DL Extension Library

3.7 JSON and Pickle

JSON and Pickle are formats used to store and transfer data. The JSON format is used to serialize/abstract over model parameters and also to deserialize OpenML flow. Pickle is used to serialize/deserialize python objects into pickle format. It is used to create copies of models which is necessary for running models on a task. To do so, the Keras and PyTorch extensions use only their "load" and "dumps" functions described in table 3.8.

Property	Type	Description
json.load	Function	Turns a parameter value into a JSON string format.
json.dumps	Function	Parses a JSON string and extracts its parameter value.
pickle.load	Function	Turns a parameter value into a pickle string format.
pickle.dumps	Function	Parses a pickle string and extracts its parameter value.

Table 3.8: JSON and Pickle properties used by DL Extension Library

3.8 Dash

Dash is a Python framework for building analytical web applications [6]. It is used by OpenML already and as such it was the preferred visualization framework for project "Sharing deep learning models". The used functions are presented in table 3.9

Property	Type	Description
dash.Dash	Function	Constructor for dash application
dash_core_components .Input	Function	Creates input field.
dash_core_components .Checklist	Function	Creates checklist.
dash_core_components .Dropdown	Function	Creates dropdown list.
dash_core_components .Graph	Function	Creates graph.
dash_html_components.Div	Function	Creates HTML Div component.
dash_html_components.H1	Function	Creates HTML H1 (header) component.
dash_html_components.H3	Function	Creates HTML H3 (smaller header) component.
dash_html_components .Button	Function	Creates HTML Button component.
dash_html_components .Iframe	Function	Creates HTML IFrame component.
dash.dependencies.Input	Function	Gives input parameters for dash component.
dash.dependencies.Output	Function	Gives output location for dash component.
dash.dependencies.State	Function	Saves the input state without passing the values immediately.

Table 3.9: Dash properties used by DL Extension Library

4 | System Design

This chapter contains a detailed description of the system design for the DL Extension Library and visualization module. More specifically, it focuses on the design method used and the decomposition of the system, while also providing the file structure of the project "Sharing deep learning models".

4.1 Design Method

4.1.1 DL Extension Library

The DL Extension Library is an extension of the existing OpenML Python API. Therefore, OpenML Support Squad complies with the standards and system design that OpenML is using at the time of the project. The OpenML Python API determines the structure of the project. The OpenML Support Squad only builds additional extensions upon the already existing structure by implementing the Extension interface. This is done to decrease coupling and allow new extensions to be added whenever this is needed. Since the Extension interface is already in place at the time that the project started, the team does not have much influence on how the architecture of the DL Extension Library would look. OpenML Support Squad does not have to deal with anything other than the DL Extension Library so it is also very hard to deconstruct the system into components.
The DL Extension Library is written in Python 3.5 and follows the PEP8 coding standard [20]. New comments keep the structure of the already existing ones. This is necessary to keep the documentation updated automatically.

4.1.2 Visualization Module

The visualization module is a module that allows visualization of OpenML flows and runs. It is a separate module from OpenML and is created as a prototype which could later be integrated into the OpenML website. The visualization module is restricted to using Dash due to the existing architecture of the OpenML website.
The visualization is written in Python 3.5 and follows the PEP8 coding standard [20].

4.2 Decomposition Description

4.2.1 DL Extension Library

Because OpenML Support Squad is not allowed to modify the architecture in any major ways it is not possible to decompose the DL Extension Library into components and describe each separately. All of the components correspond to the different DL libraries and are all identical when viewed from an architectural standpoint which does not look into code specifics. This means that it would be more meaningful to look into the general structure of one of them and describe its architectural specifics, while keeping in mind that all other libraries have an identical structure.

4.2.1.1 Extension Interface

The DL extensions are built on the already existing Extension interface, which is an interface that, at the start of the project, is only implemented for the Sklearn library. This Sklearn extension, as well as the Extension interface can be found in the extensions folder (see 4.3.1). For every additional DL library extension (such as Keras, PyTorch etc.), a new folder is created inside the extensions folder. The newly created folder contains an "*__init__*" file used to create packages. Every extension has this file, because it represents a package for a corresponding framework. It also contains a file implementing the Extension interface, called *extension.py*. This already entails most of the structure for the project "Sharing deep learning models".

4.2.1.2 Extension Implementation

For every supported DL framework OpenML Support Squad is implementing the Extension interface. As mentioned above, this is done in the respective *extension.py* file implementing the Extension interface for the specific library. The contents of these files differ slightly from one another to accommodate the different libraries. The public methods of the *extension.py* files are the same for all these extensions and so is their overall structure.
Currently supported extensions: Keras, PyTorch, MXNet, ONNX are depicted in figure 4.1.

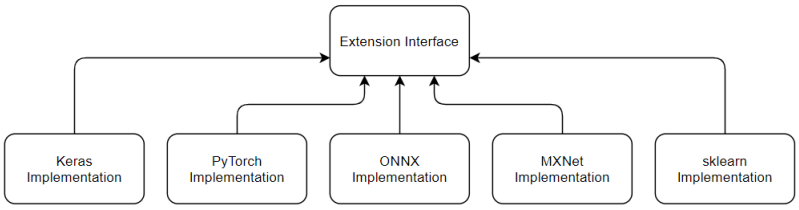


Figure 4.1: Diagram depicting the Extension Interface implementation

4.2.2 Visualization Module

By using Dash, the visualization module is a client-server web application. Nonetheless, the latest version of Dash, at the time of developing the visualization module, does not allow for extensive modularization and separation of client and server responsibilities. This cannot be achieved without degrading the overall architecture of the module. For the aforementioned reason, the module contains the following components: Visualizer component and Utility component. These will be explained in the following subsections.

4.2.2.1 Visualizer module

This module contains the layout of the front-end User Interface (UI) and the Dash callback functions. In Dash, updates on the layout are performed by using callbacks, where changes, that occur in UI components, trigger callback functions, which in turn update other UI components. Due to current limitations of Dash, the layout and the callback functions are tightly coupled. Separating them in different files requires creating cyclic dependencies between the files, which degrades the overall architecture of the module and as a result, this module is fully contained in `visualizer.py`.

4.2.2.2 Utility component

Utility functions and constants are created and separated into the *Utility component*, since there is overlapping logic between many of the defined Dash callbacks. This component is used by the visualizer component in order to generalize common behavior and improve extensibility of the module. This utility component is contained within the `utils.py` and `constants.py` files.

4.3 File Structure

4.3.1 DL Extension Library

root
- [appveyor] . . . Continuous integration tool
  - run_with_env.cmd . . . Versioning of Python & SDK
- [ci_scripts] . . . Re-usable continuous integration scripts
- [doc] . . . Document files
  - _[static] . . . Documentation color settings
  - _[templates] . . . Layout templates
- [examples] . . . General examples for OpenML usage
  - [keras] . . . Keras examples
    - keras_functional_classification_model.py . . . Keras functional classification model example
    - keras_sequential_classification_model.py. . . Keras sequential classification model example
    - keras_sequential_regression_model.py . . . . . Keras sequential regression model example
  - [mxnet] . . . MxNet examples
    - mxnet_mnist_visdom.py . . . MxNet Visdom visualization example
    - mxnet_sequential_classification_model.py . . . . . MxNet sequential classification model example
    - mxnet_sequential_regression_model.py . . . . .MxNet sequential regression model example
  - [onnx] . . . ONNX examples
    - onnx_mxnet_model.py . . . ONNX model example
  - [pytorch] . . . Pytorch examples
    - pytorch_mnist_visdom.py . . . PyTorch Visdom visualization example
    - pytorch_sequential_classification_model.py . . . . PyTorch sequential classification example
    - pytorch_sequential_regression_model.py . . . . . PyTorch sequential regression example
  - [sklearn] . . . Scikit-learn examples
    - openml_run_example.py . . . Running a task on Scikit-learn model
- [openml] . . . OpenML components
  - [datasets] . . . Data used to train/test models
  - [evaluations] . . . Model Performance
  - [extensions] . . . Library extensions

```
    ├── [keras] ..... Keras integration
    │   ├── __init__.py ..... Initialization file
    │   └── extension.py ..... Implementation
    ├── [mxnet] ..... MxNet integration
    │   ├── __init__.py ..... Initialization file
    │   ├── config.py ..... Configuration file
    │   └── extension.py ..... Implementation
    ├── [onnx] ..... ONNX integration
    │   ├── __init__.py ..... Initialization file
    │   ├── config.py ..... Configuration file
    │   └── extension.py ..... Implementation
    ├── [pytorch] ..... PyTorch integration
    │   ├── [layers] ..... PyTorch layers
    │   ├── __init__.py ..... Initialization file
    │   ├── config.py ..... Configuration file
    │   └── extension.py ..... Implementation
    ├── [sklearn] ..... Scikit-learn integration
    │   ├── __init__.py ..... Initialization file
    │   └── extension.py ..... Implementation
    ├── __init__.py ..... Initialization file
    ├── extension_interface.py ..... Extension interface
    └── functions.py ..... Additional functions to handle extensions
├── [flows] ..... OpenML flows
├── [runs] ..... OpenML runs
├── [setups] ..... Setup information
├── [study] ..... OpenML study
├── [tasks] ..... OpenML tasks
├── __init__.py ..... Initialization file
├── __version__.py ..... Version file
├── _api_calls.py ..... File for API calls
├── config.py ..... Configuration file
├── exceptions.py ..... File containing exceptions
├── testing.py ..... Testing file
└── utils.py ..... Utilization files
[tests] ..... Unit tests for OpenML
├── [files] ..... Miscellaneous files
├── [test_datasets] ..... Unit tests regarding datasets
├── [test_evaluations] ..... Unit tests regarding evaluations
├── [test_examples] ..... Unit tests regarding examples
├── [test_extensions] ..... Unit tests regarding extensions
    └── [test_keras_extension] ..... Unit tests for Keras extension
        ├── __init__.py ..... Initialization file
        ├── test_keras_additional_functions.py ..... Test cases for auxiliary functions
        ├── test_keras_deserialization.py ..... Test cases for deserialization
        ├── test_keras_run_model.py ..... Test cases for running a model
        └── test_keras_serialization.py ..... Test cases for serialization
```

```
    ├── [test_mxnet_extension] ..... Unit tests for MxNet extension
    │   ├── __init__.py ..... Initialization file
    │   ├── test_mxnet_additional_functions.py ..... Test cases for auxiliary functions
    │   ├── test_mxnet_deserialization.py ..... Test cases for deserialization
    │   ├── test_mxnet_run_model.py ..... Test cases for running a model
    │   └── test_mxnet_serialization.py ..... Test cases for serialization
    ├── [test_onnx_extension] ..... Unit tests for ONNX extension
    │   ├── __init__.py ..... Initialization file
    │   ├── onnx_model_utils.py ..... Utility functions
    │   ├── test_onnx_additional_functions.py ..... Test cases for auxiliary functions
    │   ├── test_onnx_deserialization.py ..... Test cases for deserialization
    │   ├── test_onnx_run_model.py ..... Test cases for running a model
    │   └── test_onnx_serialization.py ..... Test cases for serialization
    ├── [test_pytorch_extension] ..... Unit tests for PyTorch extension
    │   ├── __init__.py ..... Initialization file
    │   ├── test_pytorch_additional_functions.py ..... Test cases for auxiliary functions
    │   ├── test_pytorch_deserialization.py ..... Test cases for deserialization
    │   ├── test_pytorch_run_model.py ..... Test cases for running a model
    │   └── test_pytorch_serialization.py ..... Test cases for serialization
    ├── [test_sklearn_extension] ..... Unit tests for Scikit-learn extension
    │   ├── __init__.py ..... Initialization file
    │   └── test_sklearn_extension.py ..... Test cases for serialization and deserialization
    ├── __init__.py ..... Initialization file
    └── test_functions.py ..... Test cases for general functions
├── [test_flows] ..... Unit tests regarding flows
├── [test_openml] ..... Unit tests regarding OpenML
├── [test_runs] ..... Unit tests regarding runs
├── [test_setups] ..... Unit tests regarding setups
├── [test_study] ..... Unit tests regarding study
├── [test_tasks] ..... Unit tests regarding tasks
└── [test_utils] ..... Unit tests regarding utils
```

4.3.2 Visualization Module

```
root
└── [visualization] ..... Main folder
    ├── [static] ..... Holds stored static files
    ├── [tests] ..... Unit testing folder
    │   ├── __init__.py ..... Initialization File
    │   ├── base.py ..... Setup tests
    │   ├── test_visualization_flow.py ..... Unit tests for flow visualization
    │   ├── test_visualization_run.py ..... Unit tests for run visualization
    │   ├── test_visualization_utils.py ..... Unit tests for utilities of visualization
    │   └── utils.py ..... Helper functions for testing
    ├── __init__.py ..... Initialization File
    └── constants.py ..... File defining constant values
```

utils.py . . . . . Helper functions for visualization
visualizer.py . . . . . Main class

5 | Feasibility & Resource Estimates

In this chapter the minimum software and hardware resources requirements are specified when using the DL Extension Library. A performance estimate is also provided.

5.1 Resource Requirements

The minimum resource requirements for using the DL Extension Library are presented below.

CPU	$\geq$ 2.0 GHz x64 or equivalent
RAM	$\geq$ 8GB
Disk space	$\geq$ 10GB
Operating system	Windows 7 or later, macOS, and Linux
Python	Python 64-bit version 3.5, 3.6 or 3.7
Internet connection	needed for downloading and uploading OpenML flows as well as publishing results on runs
Dependencies	Keras, PyTorch, ONNX, MXNet depending on the extension of the DL Extension Library used and Dash for vizualization

5.2 Performance

The following section presents the expected performance for working with the DL Extension Library.

Download an OpenML flow as a Keras model	≤ 5 s
Upload a Keras model as an OpenML flow	≤ 5 s
Download an OpenML flow as a PyTorch model	≤ 5 s
Upload a PyTorch model as an OpenML flow	≤ 5 s
Download an OpenML flow as an MXNet model	≤ 5 s
Upload an MXNet model as an OpenML flow	≤ 5 s
Download an OpenML flow as a deep learning model supported by a deep learning library	≤ 5 s
Upload a deep learning model supported by a deep learning library as an OpenML flow	≤ 5 s
Run a deep learning model on a single task	≤ 8 s
Visualize an OpenML flow	≤ 8 s
Visualize a run	≤ 8 s

6 | Requirements Traceability Matrix

6.1 Software Requirements to Components

SRF	Keras Extension	PyTorch Extension	ONNX Extension	MXNet Extension	Visualization Module
SRF - 1.1	X	X	X	X	
SRF - 1.2	X	X	X	X	
SRF - 1.3	X	X	X	X	
SRF - 1.4	X	X	X	X	
SRF - 1.5	X	X	X	X	
SRF - 1.6	X	X	X	X	
SRF - 1.7	X	X	X	X	
SRF - 1.8	X	X	X	X	
SRF - 1.9	X	X	X	X	
SRF - 1.10	X	X	X	X	
SRF - 1.11	X	X	X	X	
SRF - 2.1	X				
SRF - 2.2	X				
SRF - 2.3	X				
SRF - 2.4	X				
SRF - 2.5	X				
SRF - 2.6	X				
SRF - 2.7	X				
SRF - 2.8	X				
SRF - 2.9	X				
SRF - 2.10	X				
SRF - 2.11	X				
SRF - 2.12	X				
SRF - 2.13	X				
SRF - 2.14	X				
SRF - 2.15	X				
SRF - 2.16	X				
SRF - 3.1		X			
SRF - 3.2		X			

SRF - 3.3		X			
SRF - 3.4		X			
SRF - 3.5		X			
SRF - 3.6		X			
SRF - 3.7		X			
SRF - 3.8		X			
SRF - 3.9		X			
SRF - 3.10		X			
SRF - 3.11		X			
SRF - 3.12		X			
SRF - 3.13		X			
SRF - 3.14		X			
SRF - 3.15		X			
SRF - 3.16		X			
SRF - 3.17		X			
SRF - 3.18		X			
SRF - 3.19		X			
SRF - 3.20		X			
SRF - 3.21		X			
SRF - 3.22		X			
SRF - 3.23		X			
SRF - 3.24		X			
SRF - 3.25		X			
SRF - 3.26		X			
SRF - 3.27		X			
SRF - 3.28		X			
SRF - 3.29		X			
SRF - 3.30		X			
SRF - 3.31		X			
SRF - 3.32		X			
SRF - 3.33		X			
SRF - 4.1			X		
SRF - 4.2			X		
SRF - 4.3			X		
SRF - 4.4			X		
SRF - 4.5			X		
SRF - 4.6			X		
SRF - 4.7			X		

SRF - 4.8			X		
SRF - 4.9			X		
SRF - 4.10			X		
SRF - 4.11			X		
SRF - 5.1				X	
SRF - 5.2				X	
SRF - 5.3				X	
SRF - 5.4				X	
SRF - 5.5				X	
SRF - 5.6				X	
SRF - 5.7				X	
SRF - 5.8				X	
SRF - 5.9				X	
SRF - 5.10				X	
SRF - 5.11				X	
SRF - 5.12				X	
SRF - 5.13				X	
SRF - 5.14				X	
SRF - 5.15				X	
SRF - 5.16				X	
SRF - 5.17				X	
SRF - 5.18				X	
SRF - 5.19				X	
SRF - 5.20				X	
SRF - 5.21				X	
SRF - 6.1					X
SRF - 6.2					X
SRF - 6.3					X
SRF - 6.4					X
SRF - 6.5					X
SRF - 6.6					X
SRF - 6.7					X

6.2 Components to Software Requirements

Component	**SRF**

Keras	SRF-1.1, SRF-1.1, SRF-1.2, SRF-1.3, SRF-1.4, SRF-1.5, SRF-1.6, SRF-1.7, SRF-1.8, SRF-1.9, SRF-1.10, SRF-1.11, SRF-2.1, SRF-2.2, SRF-2.3, SRF-2.4, SRF-2.5, SRF-2.6, SRF-2.7, SRF-2.8, SRF-2.9, SRF-2.10, SRF-2.11, SRF-2.12, SRF-2.13, SRF-2.13, SRF-2.14, SRF-2.15, SRF-2.16
PyTorch	SRF-1.1, SRF-1.1, SRF-1.2, SRF-1.3, SRF-1.4, SRF-1.5, SRF-1.6, SRF-1.7, SRF-1.8, SRF-1.9, SRF-1.10, SRF-1.11, SRF-3.1, SRF-3.2, SRF-3.3, SRF-3.4, SRF-3.5, SRF-3.6, SRF-3.7, SRF-3.8, SRF-3.9, SRF-3.10, SRF-3.11, SRF-3.12, SRF-3.13, SRF-3.14, SRF-3.15, SRF-3.16, SRF-3.17, SRF-3.18, SRF-3.19, SRF-3.20, SRF-3.21, SRF-3.22, SRF-3.23, SRF-3.24, SRF-3.25, SRF-3.26, SRF-3.26, SRF-3.27, SRF-3.28, SRF-3.29, SRF-3.30, SRF-3.31, SRF-3.32, SRF-3.33
ONNX	SRF-1.1, SRF-1.1, SRF-1.2, SRF-1.3, SRF-1.4, SRF-1.5, SRF-1.6, SRF-1.7, SRF-1.8, SRF-1.9, SRF-1.10, SRF-1.11, SRF-4.1, SRF-4.2, SRF-4.3, SRF-4.4, SRF-4.5, SRF-4.6, SRF-4.7, SRF-4.8, SRF-4.9, SRF-4.10, SRF-4.11
MXNet	SRF-1.1, SRF-1.1, SRF-1.2, SRF-1.3, SRF-1.4, SRF-1.5, SRF-1.6, SRF-1.7, SRF-1.8, SRF-1.9, SRF-1.10, SRF-1.11, SRF-5.1, SRF-5.2, SRF-5.3, SRF-5.4, SRF-5.5, SRF-5.6, SRF-5.7, SRF-5.8, SRF-5.9, SRF-5.10, SRF-5.11, SRF-5.12, SRF-5.13, SRF-5.14, SRF-5.15, SRF-5.16, SRF-5.17, SRF-5.18, SRF-5.19, SRF-5.20, SRF-5.21
Visualization	SRF-6.1, SRF-6.2, SRF-6.3, SRF-6.4, SRF-6.5, SRF-6.6, SRF-6.7

Bibliography

[1] OpenML Support Squad. *DL Extension Library, Software Requirement Document version 1.0.0.*

[2] OpenML Support Squad. *DL Extension Library, User Requirement Document version 1.0.0.*

[3] Software Standardisation and Control. *ESA software engineering standards. 1991.*

[4] OpenML. *OpenML Documentation.* URL: https://docs.openml.org/ (visited on 04/26/2019).

[5] Wikipedia. *Apache MXNet.* URL: https://en.wikipedia.org/wiki/Apache_MXNet (visited on 04/26/2019).

[6] Kyle Kelley. *Introducing Dash.* URL: https://medium.com/@plotlygraphs/introducing-dash-5ecf7191b503 (visited on 06/05/2019).

[7] *JSON.* URL: https://www.json.org/ (visited on 05/16/2019).

[8] Keras. *Keras homepage.* URL: https://keras.io/ (visited on 04/26/2019).

[9] Wikipedia. *Machine Learning.* URL: https://en.wikipedia.org/wiki/Machine_learning (visited on 04/26/2019).

[10] Joydeep Bhattacharjee. *Some Key Machine Learning Definitions.* URL: https://medium.com/technology-nineleaps/some-key-machine-learning-definitions-b524eb6cb48 (visited on 05/01/2019).

[11] Wikipedia. *Artificial Neural Network.* URL: https://en.wikipedia.org/wiki/Artificial_neural_network (visited on 04/30/2019).

[12] *Pickle.* URL: https://docs.python.org/3/library/pickle.html (visited on 05/16/2019).

[13] *Python Package Index.* URL: https://pypi.org/ (visited on 05/20/2019).

[14] PyTorch. *PyTorch homepage.* URL: https://pytorch.org/.

[15] TensorFlow. *TensorFlow wikipedia.* URL: https://en.wikipedia.org/wiki/TensorFlow (visited on 04/26/2019).

[16] *Torch (machine learning).* URL: https://en.wikipedia.org/wiki/Torch_(machine_learning) (visited on 05/15/2019).

[17] *Travis CI.* URL: https://en.wikipedia.org/wiki/Travis_CI (visited on 05/20/2019).

[18] *Unittest library.* URL: https://docs.python.org/3/library/unittest.html (visited on 05/20/2019).

[19] *OpenML Python Package Index.* URL: https://pypi.org/project/openml/ (visited on 05/15/2019).

[20] *PEP 8 - Style Guide.* URL: https://www.python.org/dev/peps/pep-0008/ (visited on 05/15/2019).