Machine Learning Language (MLL) – A declarative language for machine learning

# Abstract

Machine learning has matured to the point where much of it can be codified. Currently, procedural languages like Python are the primary tool for machine learning. But they are used as a tool to call libraries in the correct sequence. A cleaner solution is presented using a declarative language modeled after the SQL language. To make this language operational, a model of components is also defined.

# Introduction

# Example

Here are several examples to show how this can be used.

## A simple example

Assume all the servers and components are defined. Data for housing sales has been loaded. Create a model to predict housing sales prices, review the measures, infer the prices for a list of houses.

Create data MNIST as link to …

Create GAN Model MNIST1 using 7 layers and 10 epochs.

Train Model MNIST1 using data MNIST.

Test Model Housing using data …

Infer from data … using model housing

## A moderate example

## A complex example

# Functionality

Define data to be used for machine learning.

Define Machine Learning models based on any modern approach. This could include regression, random forests, neural networks, etc.

Train defined machine learning models on provided data.

Compare performance of different models on the same data and different model versions.

Presentation of data, models, models performance, etc.

Deploy trained machine learning models to a production environment.

Use a trained machine learning model to infer

# Architecture

## Components

The components are defined in the following table. All components can be installed on one computer or each can be on a separate computer. There must be at least one instance of each component. Meta Data is only updated in the meta data store. But the meta data is replicated to every other component as read only. An update to meta data is only complete when all components a=have accepted the update.

| ID | Component | Use | Instances | Notes |
| --- | --- | --- | --- | --- |
| AP | API | Allows access through APIs | 1-n |  |
| DI | Data Interface | IT can actively get data, link to data, or provide a location for data to be delivered. This can actively get data using FTP, API, etc. It also serves as a location for data to be dropped off. This is the left-hand side of the data lake. | 1-n |  |
| DL | Data Lake | The data lake has three distinct areas. On the left-hand side, the Data Interface can make data available in any format with any content at any time. In the middle, the data is cataloged and recorded. If it is recognized, it is transformed into a standard format with no change to the content. The right-hand side has deduplicated data that is in standard format and is ready for loading to the Data Warehouse if that is the plan.  Data does not stay on the left-hand side for long. It is processed as soon as possible into the middle. All data is retained in the middle for as long as storage space allows. This is the primary area used by data scientists. Data stays in the right hand side until it is processed into the data warehouse. | 1-n |  |
| DM | Data Mart |  | 0-n |  |
| DW | Data Warehouse |  | 0-n |  |
| MD | Meta Data | Information about all the other components. For example, it has information about how to get the data that the Data Get components needs to get. | 1-n |  |
| MS | Model | Stores the parameters for each model as well as versions of a model. | 1-n |  |
| PR | Processor | Can perform data movement, training, or inference. There must be at least three processors, one for each. | 3-n |  |
| UI | User Interface | Uses the API to access the components for management, operations, and monitoring. | 1-n |  |

## Data Store

The data store is a data lake with a controlled architecture. On the left any data can be provided using any mechanism available. One mechanism is Data Get. Once data arrives, it is reviewed and cataloged and placed in the proper location in the middle. If there are rules defined, any data in the middle can be processed and moved to the righthand side. Data on the right-hand side is available for training or inference.

## Environments

The meta data supports the definition of environments of components and a promotion mechanism between them. This could be used to create development, test, and production environments. It also supports the setting up of A/B and more complicated deployment schemes.

# Language User Guide

# Language Reference

This section is an alphabetical list of language elements and their semantics.

## Infrastructure

The kernel needs to be installed on each server (machine, VM, OS, or Container) to be used. Once it is installed it can be added. Note that all components can be installed on servers. Some components can use links to existing servers. For example, if there is an existing web server.

### Create Component

Used to create components on specific servers or server groups.

### Create Environment

Creates levels that can be used to promote components through.

### Create Server

Connects to a server that has the Kernel installed.

### Create Server Group

Groups servers so that they can be referred to as one.

### Promote Component

Used to promote components from one level to the next.

## Data

### Alter Data

Change the structure of data.

### Create Data

Defines the structure of data.

### Create Data Get

Creates an outline of a way to get data. It can be one time or repetitive.

### Create Data Receive

Defines a general way to receive data.

### Delete Data

Removes a data definition. This is a soft delete.

## Model

### Alter Model

### Create Model

### Delete Model

## Training

### Train Model {model} on data {data}

## Inference

Uses a model and data to make an inference. The data can be provided on the inference statement or come from the data store component.

### Infer from {data} using {model}

Infers from the data provided using the model provided. The model can specify a version.

## Authentication

Use an authentication provider to allow users to access the system.

## Authorization

Use an authorization provider to allow users to use the system.

# API User Guide

# API Reference

All APIs are authenticated. All APIs support GET, HEAD, POST, PUT, PATCH, DELETE, and OPTIONS.

## components

Work with the set of components.

## component

## models

## model

## servers

## server

## system

## systems

## users

## user

# Installation

There are multiple types of installation. There will only be a few actual installation methods. This section describes the types of installation from the user’s perspective. The general approach is to install a small kernel that can be expanded. The initial installation will install the kernel and additional services needed to start the UI. For additional machines, only the kernel needs to be installed. Any additional services can then be added through the UI once the kernel is installed, and the machine is added to the complex.

## Home

The expectation is that for home use, the user would want everything to run on one machine. It is expected that there would only be one environment which would most likely be for development.

What will be used for Authentication and Authorization providers?

### Bare Metal

The preferred approach. Take a small machine and completely replace an OS with the installation of MLL. It will install Linux, Kubernetes, create multiple docker images for the needed servers, and present a web UI.

### Linux

Installation of Linux will be done from the distribution’s repositories. By default, the installation will start the services needed to run the UI. From the UI, additional services can be started.

### Windows

The installation will be from the Windows store. Note that a user could install WSL and perform the installation there.

### Virtual Machine

The virtual machine can be running Linux or Windows. Each virtual machine will need to be created, and the kernel installed on it.

### Docker

A plain docker setup can be used in Docker for Linux or docker for Windows.

## In House

The expectation is that a company would test on a Linux or Windows install. But they would officially install on bare metal or Kubernetes. It is expected that each service will run on its own server or container. It is expected that there will be multiple servers for reliability and for increased performance.

It is expected that there will be multiple environments. At least Development, test, and production.

### Bare Metal

The preferred approach. Take a small machine and completely replace an OS with the installation of MLL. It will install Linux, Kubernetes, create multiple docker images for the needed servers, and present a web UI.

### Linux

Installation of Linux will be done from the distribution’s repositories. By default, the installation will start the services needed to run the UI. From the UI, additional services can be started.

### Windows

The installation will be from the Windows store. Note that a user could install WSL and perform the installation there.

### Kubernetes

This is on a customer-supplied Kubernetes cluster.

## Cloud

This is where a cloud provider wants to make it available to their customers. They may use the Bare Metal or Kubernetes approach. A cloud provided can mix both Multi Homed and Single Homed installations.

### Multi Homed

Clients would share servers and need to be kept separate. Capacity would be shared and there would need to be the ability to define limits. For example, a customer could pay for up to three servers or up to 10GB of disk space.

### Single Homed

A setup would be created for each client.

# Development

The initial implementation could be done by converting this language to Java and executing Java. If successful and useful, more direct implementation can be done.

## Directory Structure

{Where is the code for the UI, processing, etc.?}

Root

Installation

Paser

Src

Main

Java

Resources

Test

Java

resources

Servers {may not be needed if we are embedding servers such as mariadb}

API

DataGet

DataLake

DataMart

DataReceiver

DataWarehouse

MetaData

Model

Processor

UI

Common

## Tools

Git

Maven

Visual Studio Code

## Parser

Define the syntax in ANTLR and generate in Java.

See <https://tomassetti.me/antlr-mega-tutorial/> for a tutorial on ANTLR.

# Language Syntax

Syntax diagrams of the language.

Notes

There are language components for:

* Infrastructure
* Data Definition
* Model Definition
* Model Training
* Model Inference