Aim

Release 3.9.1

Gev Sogomonian, Gor Arakelyan et al.

Aug 24, 2022

CONTENTS

Jump to:

- Quick start
- Aim UI
- Using Aim
- Understanding Aim
- Examples
- API reference

If you have questions:

- Open a feature request or report a bug
- Join our slack

CHAPTER

ONE

OVERVIEW

Jump to:

- Quick start
- Aim UI
- Using Aim
- Understanding Aim
- Examples
- API reference

If you have questions:

- Open a feature request or report a bug
- Join our slack

CHAPTER

TWO

GET STARTED WITH AIM

This section shows a simple end-to-end aim setup. It starts from the installation, shows how to run Aim UI and explore the results. Use this as a starting point to get familiar with the basics of Aim while getting up and running.

2.1 Installing Aim

Aim is a python package available for Linux and MacOs for Python versions 3.6+. Install Aim using `pip3`:

```
pip3 install aim
```

Verify aim was properly installed

```
aim version
```

You should see the line listing newly installed version of Aim. For instance:

```
Aim v3.5.1
```

The installed package includes Python SDK needed for tracking training runs, UI for browsing the results and CLI for managing UI and results.

2.2 Nightly releases

Aim also provides daily dev packages with the features developed between main minor releases.

```
pip3 install --pre aim
```

Please note, that if the dependencies of `aim` are not already installed, this command will try to install the development versions of those packages as well.

Previous daily dev packages can be installed using the following command:

```
pip3 install aim==3.x.0.devyyyymmdd
```

Release history

2.3 Initializing Aim repository

Aim repository is the space where all your training runs are logged.

To initialize `aim` repo in the current working directory, run:

```
aim init
```

You should see something like this on your Command line:

```
Initialized a new Aim repository at /home/user/aim
```

Your workspace is now ready for tracking training runs with Aim.

2.4 Tracking data with Aim SDK

To start tracking, first create `aim.Run` object:

```
from aim import Run
run = Run()
```

Run class provides a dictionary-like interface for storing training hyperparameters and other dictionary-like metadata:

```
hparams_dict = {
    'learning_rate': 0.001,
    'batch_size': 32,
}
run['hparams'] = hparams_dict
```

These params can be used later on the UI to query runs, metrics, images. To track metrics with aim use the `Run.track` method:

```
run.track(3.0, name='loss')
```

The complete list of supported inputs is available in section "Track media and objects"

Here's a full example demonstrating the steps above:

```
# aim_test.py
from aim import Run

run = Run()

# set training hyperparameters
run['hparams'] = {
    'learning_rate': 0.001,
    'batch_size': 32,
}

# log metric
for i in range(10):
    run.track(i, name='numbers')
```

Run the script above

```
python3 aim_test.py
```

Congrats! Your first run with Aim is ready! Now it is time to explore results with Aim UI.

2.5 Browsing results with Aim UI

Once the script above finishes you can open Aim UI and see the results:

```
aim up
```

You should see the following output meaning Aim UI is up and running:

```
Running Aim UI on repo `<Repo#-5930451821203570655 path=/.aim read_only=None>`
Open http://127.0.0.1:43800
Press Ctrl+C to exit
```

Open your browser and navigate to `http://127.0.0.1:43800` You should be able to see the home page of Aim UI:

Click on Metrics Explorer icon

In the Search bar select a "numbers" metric and click "Search". You should be able to see line chart with tracked metric:

2.6 Running Aim UI inside notebooks

Run the following commands in the notebook to run the Aim UI:

1. Load Aim extension for notebooks:

```
%load_ext aim
```

1. Run `%aim up` to open Aim UI in the notebook:

```
%aim up
```

See integration guide with Jupyter Notebook for more details.

2.7 Running Aim UI and tracking server inside Docker container

Aim Docker images are available for running Aim UI and Aim Remote tracking server. Default command for Aim Docker image is `aim up`. To spin Docker container with Aim UI with port mapping:

```
docker run --publish 43800:43800 aimstack/aim
```

To run container with Aim Remote tracking server:

```
docker run --publish 53800:53800 aimstack/aim server
```

CHAPTER

THREE

INTEGRATE AIM INTO AN EXISTING PROJECT

3.1 Any python script

```
from aim import Run

run = Run()

# Save inputs, hparams or any other `key: value` pairs
run['hparams'] = {
    'learning_rate': 0.001,
    'batch_size': 32,
}

# ...
for step in range(10):
    # Log metrics to visualize performance
    run.track(step, name='metric_name')
# ...
```

Aim easily integrates with your favourite ML frameworks.

Aim loggers give access to the `aim.Run` object instance via the `experiment` property. The `aim.Run` instance will help you to easily track additional metrics or set any other `key: value` pairs (params) relevant to your project.

In this way you can easily extend the default integrations. More info about this is abailable on Integration guides section.

3.2 Integration with Pytorch Ignite

It only takes 2 steps to simply and easily inject Aim into pytorch ignite:

```
# import aim sdk designed for pytorch ignite
from aim.pytorch_ignite import AimLogger
```

Pytorch Ignite provides trainer objects to simplify the training process of pytorch model. We can attach the trainer object as AimLogger's output handler to use the logger function defined by aim to simplify the process of tracking experiments. This process is divided into 2 steps:

Step 1. Create `AimLogger` object

```
aim_logger = AimLogger(
    experiment='aim_on_pt_ignite',
    train_metric_prefix='train_',
    val_metric_prefix='val_',
    test_metric_prefix='test_',
)
```

Step 2. Attach output handler to the `aim_logger` object

```
aim_logger.attach_output_handler(
    trainer,
    event_name=Events.ITERATION_COMPLETED,
    tag="train",
    output_transform=lambda loss: {'loss': loss}
)
```

Adapter source can be found here. Example using Pytorch Ignite can be found here.

3.3 Integration with Pytorch Lightning

We only require 2 steps to simply and easily inject Aim into pytorch lightining:

```
# import aim sdk designed for pl
from aim.pytorch_lightning import AimLogger
```

Pytorch lighting provides trainer objects to simplify the training process of pytorch model. One of the parameters is called logger. We can use the logger function defined by aim to simplify the process of tracking experiments. This process is divided into 2 steps:

Step 1. Create `AimLogger` object

```
# track experimental data by using Aim
aim_logger = AimLogger(
    experiment='aim_on_pt_lightning',
    train_metric_prefix='train_',
    val_metric_prefix='val_',
)
```

Step 2. Pass the `aim_logger` object as the `logger` argument

```
# track experimental data by using Aim
trainer = Trainer(gpus=1, progress_bar_refresh_rate=20, max_epochs=5, logger=aim_logger)
```

Adapter source can be found here. Example using Pytorch Lightning can be found here.

3.4 Integration with Hugging Face

You only need 2 simple steps to employ Aim to collect data

Step 1: Import the sdk designed by Aim for Huggingface.

```
from aim.hugging_face import AimCallback
```

Step 2: Hugging Face has a trainer api to help us simplify the training process. This api provides a callback function to return the information that the user needs. Therefore, aim has specially designed SDK to simplify the process of the user writing callback functions, we only need to initialize `AimCallback` object as follows:

```
# Initialize aim_callback
aim_callback = AimCallback(experiment='huggingface_experiment')
# Initialize trainer
trainer = Trainer(
    model=model,
    args=training_args,
    train_dataset=small_train_dataset,
    eval_dataset=small_eval_dataset,
    compute_metrics=compute_metrics,
    callbacks=[aim_callback]
)
```

Adapter source can be found here. Example using Hugging Face can be found here.

3.5 Integration with Keras & tf.Keras

It only takes 2 steps to easily integrate aim in keras to record experimental information.

```
# call keras as the high api of tensorflow
from aim.tensorflow import AimCallback
# call keras library directly
from aim.keras import AimCallback
```

In keras, we call the `fit()` method of the model object to train the data. The callbacks are provided here. `AimCallback` inherits the usage specification of callbacks. We just need to add it to the callbacks list.

```
model.fit(x_train, y_train, epochs=5, callbacks=[
          # in case of tf.keras, we use aim.tensorflow.AimCallback
          AimCallback(experiment='aim_on_keras')
])
```

Adapter source can be found here. Example using Keras can be found here. Example using tf.Keras can be found here.

3.6 Integration with XGboost

Enjoy using aim to track xgboost experimental data which requires two simple steps:

Step 1: Explicitly import the `AimCallback` for tracking training data.

```
# call sdk aim.xgboost
from aim.xgboost import AimCallback
```

Step 2: XGboost provides the `xgboost.train` method for model training, in which the callbacks parameter can call back data information from the outside. Here we pass in aimcallbacl designed for tracking data information

```
xgboost.train(param, dtrain, num_round, watchlist,
                            callbacks=[AimCallback(experiment='xgboost_test')])
```

During the training process, you can start another terminal in the same directory, start `aim up` and you can observe the information in real time.

Adapter source can be found here. Example using XGboost can be found here.

CHAPTER

FOUR

MIGRATE FROM OTHER TOOLS

The Aim explorers add true superpowers to the AI engineer's arsenal. However not all training runs may have been tracked by Aim. So it is important to be able to port existing training run logs. There might be 1000s of training runs tracked with other tools. Aim has built-in converters to easily migrate logs from other tools. These migrations cover the most common usage scenarios. In case of custom and complex scenarios you can use Aim SDK to implement your own conversion script.

As of Aim `v3.6.0` the following converters are supported:

- *TensorBoard logs converter*
- *MLFlow logs converter*

We are working to constantly improve existing converters and implement new ones.

4.1 Show TensorBoard logs in Aim

Aim gives you a possibility to convert TensorBoard logs into native format and show them directly inside the Aim UI.

Before showing the logs in Aim, the files have to pass the conversion process.

Please note that only the following plugins are currently supported

- scalar
- image

To convert TensorBoard logs, `aim convert` command must be run on your log directory.

```
aim convert tensorboard --logdir ~/my_logdir
```

To make conversion process smooth please ensure that logs directory structure follows conventions below. Consider the following directory hierarchy:

```
~/my_logdir/
    ├> run_1/
    │    ├> <evnet_file_1>
    │    └> <evnet_file_2>
    ├> group_1/
    │    ├> <evnet_file_3> (THIS LOG WILL BE IGNORED)
    │    ├> run_2/
    │    │    ├> train/
    │    │    │    ├> <evnet_file_4>
    │    │    │    └> <evnet_file_5>
```

(continues on next page)

(continued from previous page)

```
    │       │       ├> validate/
    │       │       │   ├> <evnet_file_6>
    │       │       │   └> <evnet_file_7>
    │       │       ├> <evnet_file_8> (IGNORED IF "--flat" IS ACTIVE)
    │       │       └> <evnet_file_9> (IGNORED IF "--flat" IS ACTIVE)
    │       └> run_3/
    │           ├> <evnet_file_10>
    │           └> <evnet_file_11>
    ├> <evnet_file_12> (THIS LOG WILL BE IGNORED)
    └> <evnet_file_13> (THIS LOG WILL BE IGNORED)
```

Note that directory naming is not mandated and its up to you how to name them.

The conversion logic categorizes your hierarchy into one of `group`, `run` and `context` categories where.

- group: Is a directory which has one or more `run` directories inside it,
- run: Is a directory which has either log files or context directory inside it,
- context: Is a directory inside of `run` directory which has at least one log file inside it.

Conversion process will scan and determine `run` directories for your hierarchy and will create a distinct run for each of them.

From the hierarchy example above you can see that the following log files will be ignored since the converter treats them as unorganized log files.

- `~/my_logdir/group_1/evnet_file_3`
- `~/my_logdir/evnet_file_12`
- `~/my_logdir/evnet_file_13`

All other logs will either have `Context` or `No Context`. Context of the log is the name of the parent directory if the parent directory hasn't been categorized into neither as `run` nor `group` category.

For example:

- Log files right underneath `run_1`, `run_2` and `run_3` will have no context
- Log files in `run_2/train` and `run_2/validate` will have `train` and `validate` as context accordingly.

In case the converter finds unorganized log files in your hierarchy a warning message will be issued.

To make the converter process these logs, consider re-structuring your directories so that it matches the sample structure. (i.e. create a new directory and moving your unorganized logs there)

You can make converter treat every directory as a distinct run by supplying `--flat` option. In this case the following directories will be categorized as a `run` directory.

- `~/my_logdir/run_1/`
- `~/my_logdir/group_1/run_2/train/`
- `~/my_logdir/group_1/run_2/validate/`
- `~/my_logdir/group_1/run_3/`

The log files in all other directories will be ignored.

4.2 Show MLflow logs in Aim

Aim gives you a possibility to convert MLflow runs into native format and show them directly on Aim UI.

Before showing your MLlfow runs on Aim, they need to pass conversion process where your metrics, tags, parameters, run description/notes and *some* artifacts will be transferred into Aim storage.

Please note that as for now, only the artifacts having the following file extensions will be transferred into Aim storage!

- Images: `( 'jpg', 'bmp', 'jpeg', 'png', 'gif', 'svg' )`
- Texts: `( 'txt', 'log', 'py', 'js', 'yaml', 'yml', 'json', 'csv', 'tsv', 'md', 'rst', 'jsonnet' )`
- Sound/Audios: `( 'flac', 'mp3', 'wav' )`

To convert MLflow runs, `aim convert mlflow` command must be run on your log directory:

```
$ aim init
$ aim convert mlflow --tracking_uri 'file:///Users/aim_user/mlruns'
```

You can also set the `MLFLOW_TRACKING_URI` environment variable to have MLflow find a URI from there. In both cases, the URI can either be an HTTP/HTTPS URI for a remote server, a database connection string, or a local path to log data to a directory.

The conversion process will iterate over all your Experiments and create a distinct run for each run inside the experiment. If you want to process only a single experiment, you can provide the experiment id or name to the conversion command:

```
$ aim convert mlflow --tracking_uri 'file:///Users/aim_user/mlruns' --experiment 0
```

While converting the artifacts, the converter will try to determine file content type only based on its extension. A warning message will be issued if artifact cannot be categorized, these artifacts will not be transferred to aim! Please check the command output logs if you fail to see your artifact in Aim's web.

If you think there is problem with this conversion process please open an issue.

Once conversion process is complete - you can enjoy the power of Aim

CHAPTER

FIVE

TRACK MEDIA AND OBJECTS

Aim supports variety of data sources. Basic logging of `Run` params covers Python builtin types (such as `int`, `float` , `bool`, `bytes` and `str`) as well as composition of those into dictionaries, lists, tuples at any depth.

In addition to the builtin types, Aim provides native support for OmegaConf configs, thus simplifying integration for projects running with Hydra.

Starting from `v3.6.0` Aim provides integration with activeloop/hub datasets. Hub is the open-source dataset format for AI.

Tracking of data includes metrics, images, audio, text and chart figures. Here's the complete list of Aim objects provided by the package:

- Metrics
- *Distribution*
- *Image*
- *Audio*
- *Text*
- *Figure*

5.1 Distribution tracking with Aim

You can store distribution objects in Aim repository using our `aim.Distribution` object.

```
from aim import Distribution
```

`aim.Distribution` accepts the following parameters

- `distribution`: array-like object used to construct `aim.Distribution`.
- `bin_count`: Optional distribution bin count. 64 by default, max 512.

Your data is converted to `numpy.histogram` upon initialization of the object.

Simple example of initializing and tracking distribution

```
import random
from aim import Run, Distribution

run = Run()
d = Distribution(
```

(continues on next page)

(continued from previous page)

```
    distribution=[random.randrange(0, 10000) for _ in range(1000)],
    bin_count=250
)
run.track(d, name='dist', step=0)
```

5.2 Image tracking with Aim

Aim lets your track an image using `aim.Image` object

To get started, first import the `Image` object into your code.

```
from aim import Image
```

Our Image object uses Pillow under the hood. Image object supports the following inputs as data source.

- Path to an image file
- **PIL** (Pillow object)
- **torch.Tensor** (PyTorch tensor object)
- **tf.Tensor** (TensorFLow tensor object)
- **np.array** (Numpy array object)
- **matplotlib.figure.Figure** (matplotlib figure object)

Here's an example of tracking image from file path

```
path = "~/test_image.png"
aim_image = Image(path)
```

Image object also has the following arguments:

```
caption  (:obj: `str`,  optional): Optional image caption. '' by default.
format   (:obj: `str`,  optional): Parameter for PIL's .save() method. 'png' by default.
quality  (:obj: `int`,  optional): Parameter for PIL's .save() method. 85 by default.
optimize (:obj: `bool`, optional): Parameter for PIL's .save() method. False by default.
```

For more information on the `format`, `quality` and `optimize` parameters, refer to Pillow documentation.

Using these parameters you can manipulate image quality and/or convert the image format from `.png` to `jpeg` or to any other format (which is supported by Pillow)

```
from aim import Run, Image

# Initialize a new run
run = Run()

for step in range(1000):
    # Log image
    path = f"~/test_image_{step}.png"
    aim_image = Image(
        path,
        format='jpeg',
```

(continues on next page)

(continued from previous page)

```
        optimize=True,
        quality=50
    )

    run.track(aim_image, name='images', step=step)
```

5.3 Audio tracking with Aim

Aim lets your track an audio data using `aim.Audio` object

To get started, first import the `Audio` object into your code.

```
from aim import Audio
```

You can use `Audio` object to track MP3, WAV and FLAC audio data. Audio object supports the following data as input.

- File path
- Raw bytes
- `io.BytesIO` stream
- Numpy array (only for WAV audio format)

This object comes with the following optional arguments.

```
format  (:obj:`str`): Format of the audio source. Choices are ('flac', 'mp3', 'wav')
rate    (:obj:`int`): Only for WAV. Rate of the audio file, defaults to 22500
caption (:obj:`str`): Optional audio caption. An empty string by default.
```

Complete example of tracking WAV audio data.

```
import os.path
from aim import Run, Audio

# Initialize a new run
run = Run()

for step in range(1000):
    # Log image
    path = f"~/test_audio_{step}.mp3"
    aim_audio = Audio(
        path,
        format='mp3',
        caption=os.path.basename(path)
    )

    run.track(aim_audio, name='audios', step=step)
```

5.4 Text tracking with Aim

Aim lets your track text/string during your training process.

To get started, first import the `Text` object into your code.

```
from aim import Text
```

In order to use the `Text` object, you just need to ensure that your input data type is a string.

Here's an example of `Text` usage:

```
import random
import string
from aim import Run, Text

# Initialize a new run
run = Run()

for step in range(100):
    # Generate a random string for this example
    random_str = ''.join(random.choices(
        string.ascii_uppercase +
        string.digits, k=20)
    )
    aim_text = Text(random_str)
    run.track(aim_text, name='text', step=step)
```

5.5 Figure tracking with Aim

Aim provides a Figure object which can be used to track plotly and matplotlib figures.

To get started, first import the `Figure` object into your code.

```
from aim import Figure
```

You should pass either Plotly Figure or matplotlib Figure as input source to Aim's Figure object.

Here's an example of tracking a plotly figure

```
import plotly.express as px
from aim import Run, Figure

# Initialize a new run
run = Run()

# First we create Plotly figure
fig = px.bar(x=["a", "b", "c"], y=[1, 3, 2])

# Now we convert it to Aim Figure
aim_figure = Figure(fig)

run.track(aim_figure, name="plorly_ figures", step=0)
```

It is also easy to track `matplotlib` figure. Please note that the conversion process is done by Plotly under the hood.

```
from aim import Run, Figure
import matplotlib.pyplot as plt

# Initialize a new run
run = Run()

# define matplotlib figure
fig = plt.figure()
plt.plot([1, 2, 3])
plt.close(fig)

# Now we convert it to Aim Figure using (Plotly's functions)
aim_figure = Figure(fig)

run.track(aim_figure, name="matplotlib_figures", step=0)
```

5.6 Tracking matplotlib figures with Aim

In order to track `matplotlib` figures with Aim, either pass the `matplotlib` figure to Aim's `Image` or `Figure` object.

5.6.1 Converting matplotlib to Aim Image

```
from aim import Run, Image
import matplotlib.pyplot as plt

run = Run()

# define matplotlib figure
fig = plt.figure()
plt.plot([1, 2, 3])
plt.close(fig)

# pass it to aim Image
aim_img = Image(fig)
run.track(aim_img, step=0, name="matplotlib_images")
```

5.6.2 Converting matplotlib to Aim Figure

Please note that the conversion process is done by Plotly under the hood.

```
from aim import Run, Figure
import matplotlib.pyplot as plt

run = Run()

# define matplotlib figure
fig = plt.figure()
```

(continues on next page)

(continued from previous page)

```
plt.plot([1, 2, 3])
plt.close(fig)

aim_figure = Figure(fig)
run.track(aim_figure, step=0, name="matplotlib_figures")
```

5.7 Logging activeloop/hub dataset info with Aim

Aim provides wrapper object for `hub.dataset`. It allows to store the dataset info as a `Run` parameter and retrieve it later just as any other `Run` param. Here is an example of using Aim to log dataset info:

```
import hub

from aim.sdk.objects.plugins.hub_dataset import HubDataset
from aim.sdk import Run

# create dataset object
ds = hub.dataset('hub://activeloop/cifar100-test')

# log dataset metadata
run = Run(system_tracking_interval=None)
run['hub_ds'] = HubDataset(ds)
```

5.8 Log DVC metadata with Aim

If you are using DVC to version your datasets or track checkpoints / other large chunks of data, you can use Aim to record the info about the tracked files and datasets on Aim. This will allow to easily connect your datasets info to the tracked experiments. Here is how the code looks like

```
from aim.sdk import Run
from aim.sdk.objects.plugins.dvc_metadata import DvcData

run = Run(system_tracking_interval=None)

path_to_dvc_repo = '.'
run['dvc_info'] = DvcData(path_to_dvc_repo)
```

If we consider the following sample repo provided by DVC team:

Run the following command to list repository contents, including files and directories tracked by DVC and by Git.

```
$ git clone https://github.com/iterative/example-get-started
$ cd example-get-started
$ dvc list .
.dvcignore
.github
.gitignore
README.md
data
```

(continues on next page)

(continued from previous page)

```
dvc.lock
dvc.yaml
model.pkl
params.yaml
prc.json
roc.json
scores.json
src
```

If we apply our previous code snippet on the same repo - we can observe the same information added to Run parameters.

```
{
    'dvc_info.dataset.source': 'dvc',
    'dvc_info.dataset.tracked_files': [
        '.dvcignore', '.github', '.gitignore',
        'README.md', 'data', 'dvc.lock',
        'dvc.yaml', 'model.pkl', 'params.yaml',
        'prc.json', 'roc.json', 'scores.json', 'src'
    ]
}
```

CHAPTER

SIX

NEXT STEPS

In the Quick Start section you learned the basics of Aim functionality and how to use the Aim SDK.

In the following sections we will cover different aspects of Aim in more detail. Aim is designed and built to explore and make sense of large volumns of training run logs.

If you are interested in exploring more about Aim, please navigate to Using Aim section.

You can browse real-life Aim use-cases in Examples section.

If you want to learn more about how Aim works under the hood and get familiar with Aim terminology, navigate to Understanding Aim section.

Detailed API descriptions as well as CLI commands available at References.

CHAPTER

SEVEN

OVERVIEW

Aim UI is one of the key two interfaces to interact with the ML training runs tracked by Aim. It's super-powerful for comparing large number of experiments.

The Aim UI is mainly built around the Explorers that allow to query and compare metrics, images and params, scatterplots etc.

Explorers are built around the way to query runs and compare them. Each explorer has a way to navigate to a single run where all the run-related information is visualized and can be used to do deep-dive into runs.

Besides these, Aim UI also allows to tag the runs, delete/archive them and save Explorers state to share with the team.

7.1 Pythonic Search

Once you track your experiments with Aim, they are available to be searched. There are three objects that could be searched:

- Runs
- Metrics
- Images

This basically means that you can write a python if-statement over everything you have tracked with a few options of the output.

Explorers encompass these outputs and give you unique supwer-powers to manipulate and compare the results of the search.

7.2 Explorers

Explorers are powerful tools to query ML training runs, select specific type of data tracked (metrics, params etc), apply modifications to them and compare them.

The runs are compared by grouping and dividing them via the tracked hyperparams. **Every** param (even the system params) are available on Aim - to be used for runs comparison on all Explorers.

7.3 Runs Management

Runs Management includes the Runs Explorer. You can search through runs on the Runs Explorer and have the holistic view of your runs, their diff, the last values of the metrics, all the params etc.

This also contains the Single Run Page that will help you observer everything you have tracked for your run - all in one place. This includes params, metrics, images, distributions etc.

7.4 Other Aim UI features

Besides this, you can also save the Explorer states for reproducible experiment analysis. Aim also enables ways to tag the runs.

CHAPTER

EIGHT

HOME PAGE

Aim Home Page is a high level overview of your training activities and how to find your way around the Aim UI.

There are four main sections:

- *Statistics and Activity*
- *Integrate Aim with your code*
- *Explore Aim*
- *Get Involved*

8.1 Statistics and activity

Use the Statistics and activity to obvserve general info about your experiments and runs.

The `heatmap` shows the intensity of experiments you have made for the day - the darker the color the more experiments.

Each cell represents the set of training runs for that day.

The cells are clickable!

Once clicked, you will navigate to the Runs explorer page and automatically query the runs made on that day.

8.2 Integrate Aim with your code

A quick guide on how to get started with Aim.

Also links to the docs, a colab example and a live Aim demo.

8.3 Get Involved

The Aim community is growing rapidly. Join the Aim slack.

Ask questions! You'll be most welcome!

8.4 Explore Aim

Aim is a collection of super-effective ML experiment Explorers. Use this section to navigate through them.

CHAPTER

NINE

RUNS MANAGEMENT

9.1 Runs Explorer

9.1.1 Overview

To navigate into Runs Explorer, click on the `Runs` navigation item from the left sidebar.

Runs explorer helps you to

- *Search runs with pythonic query*
- *Observe runs in real time*
- *Delete or archive runs*
- *Export Runs report*

Search Runs

Use Search bar to query runs with Aim QL.

Follow runs in real time

Switch `Live Update` to turn on the real time mode.

Delete or archive runs

Step 1: Select runs on the runs table:

Click on the `Archive` button. Fonfirmation popup appears. Click `Archive` again and the runs are archived!

In order to batch delete the selected runs, just use the `Delete` button. In this case ase well, press `Delete` again on the confirmation popup and the runs will be hard deleted. ***Warning:**_ this operation is irreversible and the runs are deleted from the disk.

Export Runs report

Generate Runs CSV report by clicking on the `Export` button on the Runs table.

9.2 Single run page

Each training run has a dedicated page on Aim. Use the single run page to observe all the tracked metadata associated with that run.

Here are the tabs available on the single run page. Each tab visualizes respective tracked metadata or empty if not tracked.

- *Overview*
- *Params*
- *Metrics*
- *System*
- *Distributions*
- *Images*
- *Audios*
- *Texts*
- *Figures*
- *Settings*

9.2.1 Overview

Overview tab shows overall info about the run.

- These cards can contain information about `Parameters`, `Metrics`, `System Metrics`, `CLI Arguments`, `Environment Variables`, `Packages` and `Git information`. With this data, you can easily reproduce your run.
- Sidebar contains information about `Run Date`, `Run Duration`, `Run Hash`, attached `Tags` and gives the ability to navigate through tabs.

Also, you can apply advanced `searching/filtering` to those card tables.

9.2.2 Params

Params tab contains a JSON-like visualization of all the tracked params data related to a single run of interest.

9.2.3 Metrics

Metrics tab contains the visualizations of all the metrics tracked for the given run.

Note: you can track arbitrary number of runs with lots of steps with Aim!

9.2.4 System

Aim automatically tracks system metrics, so you can use them in order to detect potential resource mismanagements or anomalies.

System tab contains all the tracked system metrics for a single run.

9.2.5 Distributions

You can track the gradient, the weights and the biases distributions of all the layers for lots of steps with Aim.

The Distributions tab will allow you to observe them for a single run. You can also

- navigate between the layers
- search for distribution on specific steps

The single run distributions tab is quite powerful!

9.2.6 Images

The Images tab contains all the tracked images of a single run. You can track runs with different contexts and at different steps of training.

On the left-hand side are the names of different image-sets you have tracked along with their context unpacked.

Usually the images are tracked at diff steps and with batches. This control will allow you to quickly slice and dice the specific subset of images to view. Use these sliders to search

- which subset of steps you'd like to see (on the left-hand side)
- which indices you'd like to see (on the right-hand side)

If there is only 1 step or only 1 index, you will see the info message instead of the control

9.2.7 Audios

You can track audios with Aim. Use the Audios tab to view and play the tracked audios of a single run.

9.2.8 Texts

Use the `Texts` tab to view and search all the texts tracked for a single run.

On the left-hand side you will see the name and context of the tracked texts. You can use the search-bar on top to search the text with regexp or just match word or case.

Use the bottom controllers to control the steps and the indices of the tracked texts too.

9.2.9 Figures

Aim allows tracking Plotly and matplotlib figures. On the `Figures` tab you can view all the track figures over different contexts and steps.

Note: Aim will render figures with passed or default dimensions. There will be scrolls if the size exceeds the plotly container space of the Figures tab.

9.2.10 Settings

Use the `Settings` tab to delete or archive the single run

Delete Run

Archive Run

CHAPTER

TEN

EXPLORERS

Explorers will help you to compare 1000s of AI experiments with a few clicks. Explorers are the main tools that Aim is built around.

In this section we will go through the Aim explorers, introduce their features and how to use them.

10.1 Metrics Explorer

10.1.1 Overview

Use Metrics explorer to search and compare 1000s of ML training metrics.

The Metrics Explorer allows you to search, group and compare your metrics. Due to this and number of other visual features on the Metrics Explorer, you will save considerable amounts of time when comparing experiments- compared to other open-source experiment tracking tools.

The Metric Explorer has the following main sections:

- **Metrics Select:** to select the metrics for exploration
- **Search bar:** to query the runs for exploration
- **Charts explorer:** the space where the metrics are rendered
- **Metrics modifiers:** all the grouping, chart division and other metrics modifier tools
- **Context table:** the full information about the selected metrics is available

There is also an advanced mode of search is available too where you can use the full Aim QL (more on this further in this section).

There are two ways you can query metrics and runs

- *Select metrics and query runs*
- *Advanced Search mode*

An overview of what you can do with queried metrics - the modifiers:

- *Group by tracked parameters*
- *Aggregate grouped metrics*
- *Align metrics by time, epoch or custom metric*
- *Change scale of the axes (linear or log)*
- *Apply smoothing*
- *Ignore outliers*

- *Metric highligh modes (metric on hover, run on hover)*
- *Set chart tooltip parameters*
- *Apply zoom in/out on charts*
- *Export chart as image*

10.1.2 Select metrics and query runs

On the Metrics Explorer, there is `+ Metrics` button. Once pressed, a dropdown will appear with all your tracked metrics and their contexts flattened. The dropdown is searchable - so you can get to your metric of interest within a stroke!

The Search bar is located below the `+ Metric` button. It allows to do a pythonic query (that is eval-ed as python statement) over every param you have tracked.

Search runs with Aim QL

10.1.3 Advanced Search mode

Once you press the `Enable advanced search mode` button underneath the main `Search` button, it will enable the full Aim QL search editor - to query the metrics, the runs via full Aim QL

Here is an example:

```
((metric.name == 'bleu' and metric.context.subset == 'val') or (metric.name == 'loss' and
metric.context.subset == 'train')) and 1e-5 < run.hparams.learning_rate < 1e-2
```

10.1.4 Group by any parameter

Grouping selected metrics by any tracked params will allow you to quickly distinguish the most impactful params, decisions you have made (the preprocessing steps, the hyperparams etc).

The parameters include not only the ones you have tracked but also the native Aim objects too such as

- `metric.name`
- `metric.context.[context_key]`
- `run.hash`

There are several ways you can group the selected metrics and runs - by color, by stroke and by chart.

Group by Color

Use this to divide the selected metrics into different clusters based on selected values of params. Each cluster gets colored differently.

There are a number of options available when grouping

- **group by values** - divides into clusters as per the values of selected params)
- **reverse grouping** - divides into clusters by every param except for the chosen one.

The grouping colors are picked randomly, however it is possible to fix with the advanced coloring features.

Here are the features in the advanced mode:

- Fix the colors of the grouping
- Control the color palette to use during the grouping

Group by Stroke

Groups the metrics by a stroke style. Has all the rest of the other features available on the color grouping except the advanced mode.

Group by Chart

The end result of using this feature: divides into subplots based on the value of the selected params. Why this is a grouping mechanism? It groups the metrics belonging to the same group into separate charts.

10.1.5 Aggregate metrics

The metrics aggregation helps to quickly see the trends of each individual group of metrics. See more about *metrics grouping*.

There are two aspects of aggregation you can control:

- the trend-line
- the area that the group of metricsc take

The trend-line calculation methods:

- Mean
- Median
- Min
- Max

The area calcualtion methods:

- None *(when you'd like to remove the area)*
- Min/Max
- Mean +/- Standard Deviation

- Mean +/- Standard Error
- Confidence Interval (95%)

Pls see the screenshot:

10.1.6 X-Axis properties

X-Axis properties section is for controlling density of metrics x-axis values and aligning metrics by time, epoch or another metric.

Density:

Following types of metrics density are available: *Minimum*, *Medium*, *Maximum*. By default, metrics density is the *Maximum*.

Minimum

By setting metrics density to Minimum, will query metrics by 50 point.

Medium

By setting metrics density to Medium, will query metrics by 250 point.

Maximum

By setting metrics density to Maximum, will query metrics by 500 point.

Alignment:

Following types of metrics alignment are available: *Step*, *Epoch*, *Relative Time*, *Absolute Time* and *Custom Metric*. By default, metrics aligned by *Step*.

Step

By setting metrics alignment to Step, x-axis values will represent the steps of tracked metrics.

Epoch

By setting metrics alignment to Epoch, x-axis values will represent the epochs of tracked metrics.

Relative Time

By setting metrics alignment to Relative Time, x-axis values will represent by `HH:mm:ss`, duration of tracking process.

Absolute Time

By setting metrics alignment to Absolute Time, x-axis values will represent by date `HH:mm:ss D MMM, YY`, since the start date of the first run until the last run.

Custom Metric

By setting metrics alignment to Custom Metric, x-axis values will represent selected metric values, you can detect correlations between queried metrics and selected metric.

10.1.7 Axes Scale

Axes Scale section gives ability to display axes scale's linear or logarithmic.

By default, axes scale's are *Linear*.

Linear Scale

X-axis scale: Linear, Y-axis scale: Log

X-axis scale: Log, Y-axis scale: Linear

Log Scale

10.1.8 Chart Smoothing

While smoothing the chart, the data points are modified so individual points higher than the adjacent points (presumably because of noise) are reduced, and points that are lower than the adjacent points are increased leading to a smoother signal. You can select curve interpolation methods: Linear or Cubic. By default, chart smoothing in *Exponential moving average* mode and curve interpolation method is Linear.

Exponential moving average

An exponential moving average, also known as an exponentially weighted moving average (EWMA), is a first-order infinite impulse response filter that applies weighting factors which decrease exponentially.

Centered moving average

When you center the moving averages, the data points are placed at the center of the range rather than the end of it. This is done to position the moving average values at their central positions in time.

10.1.9 Ignore outliers

An outlier is an observation that lies an abnormal distance from other values in a random sample from a population. Examination of the data for unusual observations that are far removed from the mass of data. These points are often referred to as outliers.

Excluding outliers can cause your results to become statistically significant. By default, outliers are ignored.

10.1.10 Highlight Modes

Highlighting functionality is useful for filtering metrics and highlight only hovered metric. Following types of highlighting mode are available: *Highlight Off*, *Highlight Metric on Hover*, and *Highlight Run on Hover*. By default, highlighting mode is the *Highlight Run on Hover*.

Highlight Off

By setting Highlight mode Off, there is no highlighting functionality on hover.

Highlight Metric on Hover

By setting Highlight mode Metric on Hover, mouse point closest metric highlights and other metrics displays with opacity.

Highlight Run on Hover

By setting Highlight mode Run on Hover, mouse point closest metric highlights and highlighted metric corresponding run also highlights other metrics displays with opacity.

10.1.11 Set tooltip parameters

You can select tooltip parameters to show params and values in tooltip Params section. You can select hide or show button to display or hide tooltip on hover.

10.1.12 Apply zoom on charts

Zoom In

Zoom Out

10.1.13 Export chart as image

Metric explorer also, gives ability to export your chart as image. By clicking `export button` from control panel, will be opened chart preview modal. You can change exportable chart `image width`, `single chart height`, set `image name` and `format`.

Following formats of chart export are available: `SVG`, `JPEG`, `PNG`.

10.2 Images Explorer

10.2.1 Overview

Track intermediate images search easily by using select form functional compare them on the Images Explorer by using reach controls panel.

Features:

- *Easily query any image*
- *Group images by run parameters, step and index*
- *Use controls from right control panel to configure workspace*
  - *Image properties control*
  - *Images sorting control*
  - *Images group stacking control*
  - *Images tooltip params control*

10.2.2 Query any image

Use select form to easily query any image. There are two option to query images using dropdown, by using Aim QL language and advance mode for Aim QL.

- Click on Images button
- Select options you are want to use in query
- Click on the Search button

- Click on pencel icon in the right side of select form to show input
- Type advance Aim QL query
- Click on the Search button

10.2.3 Group image by run parameters

Use select grouping dropdown which is located in the right top corner of the image explore page.

- Click on grouping button
- Select fields by which you want to groupe images

Grouping will be apply after each field selection also you can select grouping mode (Group or Reverse)

10.2.4 Image explorer right controls panel

Any change in controls will help to explore images better on the workspace

10.2.5 Images size manipulation control

- Click on image property button
- Select value from dropdown to align image. (by default dropdown value is `Height`). Use slider to configure value for scale relative to window size by default scale value is `15%`.
  - By height
  - By width
  - Original size
- Use image rendering variation by default value of this control is `Pixelated`

10.2.6 Images sorting control

- Click on image sorting button
- Select fields for sorting images. Selection ordering is meaningful and data will be sorting by selection order. Bellow is visible Ordered By list where contains all selected fields from dropdown. You can remove any already selected field by clicking on `x` icon or change sorting direction by clicking radio button Asc or Desc. Default selected direction is Asc.
- For reset all existing sorting fields you can simply click on Reset Sorting button

10.2.7 Set tooltip parameters

You can select tooltip parameters to show params and values in tooltip Params section. You can select hide or show button to display or hide tooltip on hover.

10.3 Params Explorer

10.3.1 Overview

Params explorer helps you to represent high dimensional data as a multi-dimensional visualization. Features:

- *Easily query any metrics and params*
- *Group runs by color, stroke, or chart*

- *Make the crossings easier with curve interpolation*
- *Learn patterns and colorations easier colored by the last dimension with a color indicator*

Query any metrics and params

Select params and metrics from dropdown

Search runs with Aim QL

Grouping

Group by color, stroke, or chart with selected parameters

Curve interpolation

By clicking on the Curve interpolation button in the Controls panel, it's possible to make lines from straight to curve to show correlations between non-adjacent axes.

Color indicator

By clicking on the Color indicator button in the Controls panel, it's possible to turn on lines gradient coloring by the last dimension.

10.4 Scatters Explorer

Scatter explorer gives ability to visualize correlations between metric last value data with hyper-parameter.

It represents graph where corresponding values from a set of data are placed as points on a coordinate plane. A relationship between the points is sometimes shown to be positive, negative, strong, or weak.

Abilities provided by Scatter explorer

- *Easily align metric last value data with hyper parameter*
- *Group runs by color and chart*
- *Represent the points with trend line*
- *Export chart*

Select params and metrics from `X` and `Y` axes dropdowns to align metric last value data with hyper-parameter.

- X axis

- Y axis

Also, you can search runs with Aim QL

Easily group data by color and chart with selected parameters.

- By `Color`

- By `Chart`

A trend line is a straight line that best represents the points on a `scatter plot`. The trend line may go through some points but need not go through them all.

From trend line options popover you can change regression from `Linear` (by default) to `LOESS`(locally weighted smoothing), which creates a smooth line through a `scatter plot` to help you to see relationship between variables and foresee trends. Also, you can change the `bandwidth` with `slider`

Scatter explorer also, gives ability to `export` your chart as `image`.

By clicking `export button` from control panel, will be opened chart preview modal. You can change exportable chart `image width`, `single chart height`, set `image name` and `format`.

Following image formats are available export: `SVG`, `JPEG`, `PNG`.

CHAPTER

ELEVEN

BOOKMARKS

11.1 Overview

Use the Bookmarks to save the Aim Explorer state. This includes search query, aggregations and any other modifications applied to the explorer. The Bookmarks page is a list of *cards* to quickly access the explorer state with one click.

There are bookmark buttons available on all Explorer pages - on the top right . In order to create a bookmark, just press the bookmark button on your preferred Explorer page.

This will open a `create bookmark` form with fields for `title` and `description`.

The created bookmark will contain all the current configuration of the explorer.

11.2 The Bookmark Card

Each of the bookmark cards contains the following:

- Explorer Icon
- Title
- View Bookmark button
- Delete Bookmark button
- Search query
- Selected metrics and params

11.3 Delete Bookmark

Clicking the delete bookmark button will open a modal, where you can confirm or cancel the deletion.

CHAPTER

TWELVE

TAGS PAGE

12.1 Overview

Tags functionality intended to mark a runs. A tag can be attached to the runs to distribute by segments and then find it quickly.

12.2 Create tag

How to create tag? There are two options for creating a tag.

12.2.1 First option

- Go to the tags page by clicking on the Tags from the left sidebar.
- Click on the create tag button to open the create tag form modal. In this modal there is the form that has two input fields first one for tag name the second one for tag comment and also there are exist color pallets for selecting tag color.

- Type name for a tag. Name filed is mandatory and can't be empty for tag creation form and has maximum 50 symbol limit validation.
- Type comment for a tag. Comment field is optional for tag creation form and has max 200 symbol limit validation.
- Select color for a tag from the color pallet.
- Click to the create button for saving a the tag then. After successful saving should appear toaster approving the create on the right top corner of the window. Optional there are default selected colors for tag.

12.2.2 Second option

- Go to any explorer page (metrics, params, images, scatters).
- Click to one of the sequence unit to open popover where is exist tag section with attach button.
- Click on attach button to open the select tag popover where you will see all your previously created tags.

- Click create tag button and you will be redirected to the tags page than the actual first option.

12.3 Attach tag

- Go to any explorer page (metrics, params, images, scatters).
- Click to one of the sequence unit to open popover where is exist tag section with attach button.

- Click on attach button to open the select tag popover where will be visible all tags.
- Select a tag you want to attach to the sequence unit. You can select more then on tag for each point.

12.4 Update attached tags

How to update attached tags?

- Go to any explorer page (metrics, params, images, scatters).
- Select point which you want to attach tag and click on it to open popover where is the exist tag section. In tags section will be visible already attached tags.
- Click on attach button for adding new tag to open the select tag popover where will be visible all existing tags.

- Click on x icon in the right end of the each tag for removing the tag from the point.

12.5 Edit tag

- Go to the tags page
- Click to edit icon in the right side in the tag row

- Then should appear the edit modal. In the edit modal there is the form that has two input fields first one for tag name the second one for tag comment and also there are exist color pallets for selecting tag color. In this modal is possible to make changes for tag.
- Make changes you need.

- Then you have three possible actions close modal, save changes and reset changes. After closing the modal you will lose all changes, after clicking the reset button modal form fields will be reset to initial values and by clicking the save button you will save all changes for the tag. After successful saving should appear toaster approving the update on the right top corner of the window.

12.6 Delete tag

- Go to the tags page.
- Click on the trash icon in the right side in the tag row to open the delete modal.

- In the delete modal there is a tag name input field and a tag name label at the top of the tag name input. You need to type the tag name for approving you are want to delete that tag.
- Then you have two possible actions delete the tag or close the modal by canceling the delete operation. If you want to delete a tag please double-check the tag name and click to delete button. After tag deletion, there are no possibilities to recover it. Also if you are deleting the tag it will be removed from all relations too.

12.7 Used in overlay

In the tags page you can select tag by clicking on circle icon then will opened overlay in the right side of window. Here is visible that runs which are use the tag. By clicking a run hash you will be redirected to single run page.

CHAPTER

THIRTEEN

MANAGE RUNS

13.1 Create runs

`Run` is the main object that tracks and stores ML training metadata(e.g. metrics or hyperparams).

When initializing a `Run` object, Aim creates a `.aim` repository at the specified path. The tracked data is stored in the `.aim` repo. If the repo path is not specified, the run data is stored in the current working directory.

Use the following `Run` arguments to:

- `repo`: define where to store the data
- `experiment`: define experiment name to group related runs together
- `system_tracking_interval`: Enable system resource usage tracking (CPU, GPU, memory, etc..). By default enabled. Set to `None` to disable

```
from aim import Run

my_run = Run(
    repo='/repo/path/to/store/runs',
    experiment='experiment_name'
)
```

`Run` class full spec.

Additionally, Aim SDK also gives a flexibility to:

- Use multiple `Runs` in one training script to store multiple runs at once. Usually handy when doing hyperparameter search.
- Use integrations to automate tracking

13.2 Continue runs

Each Run object has a `hash` associated with it which could be looked up at `aim runs ls` (check out the Aim CLI here). Specify the run hash when initializing a `Run` object to continue tracking.

```
from aim import Run

run = Run(run_hash='run_hash')
```

13.3 Delete runs

There are cases when Run data is not needed. Examples of such cases are, failed training runs or simple disk space cleanup. Aim provides SDK and CLI interfaces to delete Runs.

To remove Runs via the SDK:

```
from aim import Repo

repo = Repo.from_path('aim_repo_path')
repo.delete_run('run_hash')
repo.delete_runs(['run_hash_1', 'run_hash_2'])
```

Repo class full spec.

To remove Runs using command line:

```
aim runs rm run_hash_1 run_hash_2 run_hash_3
```

More details on `aim runs` in CLI reference.

CHAPTER

FOURTEEN

CONFIGURE RUNS

You can configure Aim run instance to change the default repository location, retrieve a particular run instance, or store custom parameters for your run. You can configure your Run upon instantiation.

Here you can see all available options you can use to configure your Run instance.

- `run_hash`: Run's hash. Can be used to restore/retrieve a particular run form the repo. This is handy if you want to update something or add new metrics in your already existing run. If skipped, the hash will be generated automatically. Example covered in *Reusing Run in your Repo* section.
- `repo`: Aim repository path or Repo object to which Run object is bound. If skipped, default Repo is used. More covered in *Defining custom location for your Repo* section.
- `read_only`: Run creation mode. Default is False, meaning Run object can be used to track metrics.
- `experiment`: Sets Run's `experiment` property. 'default' if not specified. Can be used later to query runs/sequences.
- `system_tracking_interval`: Sets the tracking interval in seconds for system usage metrics (CPU, Memory, etc.). Set to `None` to disable system metrics tracking.
- `log_system_params`: Enable/Disable logging of system params such as installed packages, git info, environment variables, etc. Checkout *Training Run Reproducibility* section for more info.

14.1 Reusing Run in your Repo

Sometimes you might want to add/update parameters and tracked values in your existing Run instance. This can be achieved trough using `run_hash` parameter. Aim generates a unique id (uuid) for your Run instance only if `run_hash` parameter is not supplied. Example of re-using generated run.

```
from aim import Run

# You can also use ID shown in Aim Web after
# creating a run without supplying run_hash parameter
uid = '508c5b29-02c7-4875-a157-f099ea193bfa'
run = Run(run_hash=uid)
for i in range(100):
    run.track(i, step=i, name='test')
```

In the example above, no matter how often this piece of code will be executed, only a single run entry will be created or updated.

14.2 Defining custom location for your Repo

When you use aim, the library automatically creates `.aim` directory which we call "Repo" in your current directory. Aim stores data about your runs inside the Repo. Sometimes creating a Repo in your current directory is not convenient due to various reasons (like lack of permissions), and in such scenarios you can define your own path where a new `.aim` directory will be created or just reused.

```
from aim import Run

run = Run(repo='~/repo')
...
```

The code above will create a new dir entry `~/repo/.aim`

14.3 Organizing Runs in Experiments

Aim allows you to group runs under experiments, which can be useful for comparing runs intended to tackle a particular task.

```
from aim import Run

run = Run(experiment="fraud-detection")
```

14.4 Adding tags and params to Run

Aim lets you add tags and parameters on your Run object. Example of this would be if you want to tag your run with a specific version or keyword. Or you might need to log parameters, so you have some insight on what params have been used to process that particular Run.

Example of adding and removing tags

```
from aim import Run

run = Run()
run.add_tag('v1.0')
run.add_tag('some-awesome-tag')
```

Or you can modify tags on your existing run, but first you have to restore Run object by using it's hash value

```
from aim import Run

uid = '508c5b29-02c7-4875-a157-f099ea193bfa'
run = Run(run_hash=uid)
run.remove_tag('some-awesome-tag')
run.add_tag('another-awesome-tag')
```

Almost same approach goes for parameters

```
from aim import Run
```

(continues on next page)

(continued from previous page)

```
run = Run()
var = 'some value'

run['var'] = var
```

Also, you can restore the Run object using its hash key and overwrite value of the previously stored parameter.

```
from aim import Run

uid = '508c5b29-02c7-4875-a157-f099ea193bfa'
run = Run(run_hash=uid)
var = 'another value'

run['var'] = var
```

14.5 Training Run Reproducibility

When running multiple training jobs it is crucial to understand the factors affecting the trained models performance. While the hyperparameters diff is an obvious place to look at, the training script environment itself can change the collected metadata in unexpected ways. It is important to be able to reproduce your runs' environment to presumably get the same results. Sometimes even minor version change in your script dependencies or a small tweak in the training code itself can affect training results. Thus, it's important to collect and log information such as package versions, environment variables, input arguments, etc. with each run.

Doing this manually requires a lot of code to be added to your training script. This is where Aim's logging of system parameters can come in handy!

14.5.1 What data is logged automatically?

Aim lets you enable system params logging for your Run which in result will log the following parameters

- Environment Variables
- Executables
- CLI arguments
- Installed packages and their versions
- Git information such as current branch, commit hash, author, etc. (if applicable)

14.5.2 How to enable system params automatic logging?

To enable logging of the parameters listed above, your Run instance must be supplied with `log_system_params=True` option, by default it is disabled!

```
run = Run(log_system_params=True)
```

In addition, these logged parameters can be used in the search box to filter runs based on the supplied parameters. Everything is searchable at Aim!

Here is an example of what you can do with it:

```
run.__system_params.git_info.branch == 'feature/testing'
```

CHAPTER

FIFTEEN

QUERY RUNS AND OBJECTS

Use Repo object to query and access saved Runs.

Initialize a Repo instance:

```
from aim import Repo

my_repo = Repo('/path/to/aim/repo')
```

Repo class full spec.

Query logged metrics and parameters:

```
query = "metric.name == 'loss'" # Example query

# Get collection of metrics
for run_metrics_collection in my_repo.query_metrics(query).iter_runs():
    for metric in run_metrics_collection:
        # Get run params
        params = metric.run[...]
        # Get metric values
        steps, metric_values = metric.values.sparse_numpy()
```

Besides querying Runs and metrics, you can also query logged Image objects:

```
query = "images.name == 'mnist_dataset' and images.context.subset == 'val"

# Get collection of Image sequences
for image_seq in my_repo.query_images(query).iter():
    # Get first tracked batch of each sequence
    image_batch = image_seq.values.first_value()
    # Get Image metadata
    image_meta = map(Image.json, image_batch)
    # Convert to PILImage
    pil_images = map(Image.to_pil_image, image_batch)
```

Image class full spec.

See more advanced usage examples here.

CHAPTER

SIXTEEN

QUERY LANGUAGE BASICS

16.1 Introduction

Aim enables a powerful query language(Aim QL) to filter through all the stored metadata.

AimQL filters the tracked metadata using **python expression**. Think of it as a python if statement over everything you have tracked. Hence, nearly any python compatible expression is available with *some restrictions* in place.

The data is saved as different types of entities (e.g. `run`, `metric`). The search queries are written against these entities. When iterating over entities the python expression is evaluated in a Boolean context. When the value is *"truthy"*, then the current entity is yielded. Otherwise the entity is skipped over.

Currently, AimQL is only used for filtering data, and has no role in sorting or aggregating the data.

16.2 Searching

Let's track several Runs via Aim SDK:

```
# Initialize run_1
# Define its params and track loss metric within test and train contexts
run_1 = Run()
run_1['learning_rate'] = 0.001
run_1['batch_size'] = 32
for i in range(10):
    run_1.track(i, name='loss', context={ 'subset':'train' })
    run_1.track(i, name='loss', context={ 'subset':'test' })

# Initialize run_2
run_2 = Run()
run_2['learning_rate'] = 0.0007
run_2['batch_size'] = 64
for i in range(10):
   run_2.track(i, name='loss', context={ 'subset':'train' })
   run_2.track(i, name='loss', context={ 'subset':'test' })

# Initialize run_3
run_3 = Run()
run_3['learning_rate'] = 0.005
run_3['batch_size'] = 16
for i in range(10):
```

(continues on next page)

(continued from previous page)

```
run_2.track(i, name='loss', context={ 'subset':'train' })
run_2.track(i, name='loss', context={ 'subset':'test' })
```

Aim SDK will collect and store the above metadata in `.aim` repo.

Run	Parameters	Metrics
run_1 <hash=a32c910>		
run_2 <hash=a32c911>		
run_3 <hash=a32c912>		

When searching runs, use the `run` keyword which represents the `Run` object. It has the following properties:

Property	Description
`name`	Run name
`hash`	Run hash
`experiment`	Experiment name
`tags`	List of run tags
`archived`	`True` if run is archived, otherwise `False`
`creation_time`	Run creation timestamp
`end_time`	Run end timestamp

Run parameters could be accessed both via chained properties and attributes.

The two following examples are equal:

- run.hparams.learning_rate == 32
- run["hparams", "learning_rate"] == 32

AimQL has been designed to be highly performant. Only the params that are used in the query will be loaded into memory.

If you use the **['hparams']['learning_rate']** syntax Aim will load the whole dictionary into memory. The search performance will be impacted.

We recommend to use either **['hparams', 'learning_rate']** or **hparams.learning_rate** syntax which are equivalent to each other in terms of the performance.

Query examples:

1. Get runs where `learning_rate` is greater than `0.0001` and `batch_size` is greater than `32`.

```
run.learning_rate > 0.0001 and run.batch_size > 32
```

Result:

Run	Parameters
run_2 <hash=a32c911>	

1. Get runs where `learning_rate` is either `0.0001` or `0.005`.

```
run.learning_rate in [0.0001, 0.005]
```

Result:

Run	Parameters
run_1 <hash=a32c910>	
run_3 <hash=a32c912>	

16.3 Searching metrics and images

16.3.1 Searching metrics

When iterating over metrics, use the `metric` keyword which represents the tracked metric. While searching metrics, you can also refer to the related runs via the `run` keyword.

`metric` has the following default properties.

Property	Description
`name`	Metric name
`context`	Metric context dictionary

Query examples

1. Query metrics by name:

```
metric.name == "loss"
```

Result:

Metric	Related run
loss { "subset":"train" }	run_1 <hash=a32c910>
loss { "subset":"test" }	run_1 <hash=a32c910>
loss { "subset":"train" }	run_2 <hash=a32c911>
loss { "subset":"test" }	run_2 <hash=a32c911>
loss { "subset":"train" }	run_3 <hash=a32c912>
loss { "subset":"test" }	run_3 <hash=a32c912>

1. Query metrics by name and context

```
metric.name == "loss" and metric.context.subset == "train"
```

Result:

Metric	Related run
loss { "subset":"train" }	run_1 <hash=a32c910>
loss { "subset":"train" }	run_2 <hash=a32c911>
loss { "subset":"train" }	run_3 <hash=a32c912>

1. Query metrics by name and run parameters

```
metric.name == "loss" and run.learning_rate >= 0.001
```

Result:

Metric	Related run
loss { "subset":"train" }	run_1 <hash=a32c910>
loss { "subset":"test" }	run_1 <hash=a32c910>
loss { "subset":"train" }	run_3 <hash=a32c912>
loss { "subset":"test" }	run_3 <hash=a32c912>

16.3.2 Searching images

Images search works in the same way as metrics. When iterating over images, use the `images` keyword which represents the tracked images sequence. While searching images, you can also refer to the related runs via the `run` keyword.

`images` keyword has the following default properties.

Property	Description
`name`	Image sequence name
`context`	Image sequence context dictionary

Query examples:

- images.name == "generated" and run.learning_rate >= 0.001
- images.name == "generated" and images.context.ema == 0

16.4 Security restrictions

AimQL expression is evaluated with RestrictedPython.

RestrictedPython is a tool that helps to define a subset of the Python language which allows to provide a program input into a trusted environment.

We have followed these restrictions to avoid security risks such as executing a non-safe function via AimQL.

CHAPTER

SEVENTEEN

TRACK EXPERIMENTS WITH AIM REMOTE SERVER (EXPERIMENTAL FEATURE)

17.1 Overview

Aim remote tracking server allows running experiments in a multi-host environment and collect tracked data in a centralized location. It provides SDK for client-server communications and utilized gRPC protocol as its core transport layer.

In this guide we will show you how to setup Aim remote tracking server and how to integrate it in client-side code.

17.2 Prerequisites

Remote tracking server assumes multi-host environments used to run multiple training experiments. The machine running the server have to accept incoming TCP traffic on a dedicated port (default is 53800).

17.3 Server-side setup

1. Make sure aim `3.4.0` or upper installed:

```
$ pip install "aim>=3.4.0"
```

2. Initialize `aim` repository (optional):

```
$ aim init
```

1. Run aim server with dedicated `aim` repository:

```
$ aim server --repo <REPO_PATH>
```

You will see the following output:

```
> Server is mounted on 0.0.0.0:53800
> Press Ctrl+C to exit
```

The server is up and ready to accept tracked data.

1. Run aim UI

```
$ aim up --repo <REPO_PATH>
```

17.4 Client-side setup

With the current architecture there is almost no change in aim SDK usage. The only difference from tracking locally is that you have to provide the remote tracking URL instead of local aim repo path. The following code shows how to create Run with remote tracking URL and how to use it.

```
from aim import Run

aim_run = Run(repo='aim://172.3.66.145:53800')  # replace example IP with your tracking␣
↪server IP/hostname

# Log run parameters
aim_run['params'] = {
    'learning_rate': 0.001,
    'batch_size': 32,
}
...
```

You are now ready to use aim_run object to track your experiment results. Below is the full example using pytorch + aim remote tracking on MNIST dataset.

```
from aim import Run

import torch
import torch.nn as nn
import torchvision
import torchvision.transforms as transforms


# Initialize a new Run with remote tracking URL
aim_run = Run(repo='aim://172.3.66.145:53800')  # replace example IP with your tracking␣
↪server IP/hostname

# Device configuration
device = torch.device('cpu')

# Hyper parameters
num_epochs = 5
num_classes = 10
batch_size = 16
learning_rate = 0.01

# aim - Track hyper parameters
aim_run['hparams'] = {
    'num_epochs': num_epochs,
    'num_classes': num_classes,
    'batch_size': batch_size,
    'learning_rate': learning_rate,
}
```

(continues on next page)

(continued from previous page)

```
# MNIST dataset
train_dataset = torchvision.datasets.MNIST(root='./data/',
                                           train=True,
                                           transform=transforms.ToTensor(),
                                           download=True)

test_dataset = torchvision.datasets.MNIST(root='./data/',
                                          train=False,
                                          transform=transforms.ToTensor())

# Data loader
train_loader = torch.utils.data.DataLoader(dataset=train_dataset,
                                           batch_size=batch_size,
                                           shuffle=True)

test_loader = torch.utils.data.DataLoader(dataset=test_dataset,
                                          batch_size=batch_size,
                                          shuffle=False)


# Convolutional neural network (two convolutional layers)
class ConvNet(nn.Module):
    def __init__(self, num_classes=10):
        super(ConvNet, self).__init__()
        self.layer1 = nn.Sequential(
            nn.Conv2d(1, 16, kernel_size=5, stride=1, padding=2),
            nn.BatchNorm2d(16),
            nn.ReLU(),
            nn.MaxPool2d(kernel_size=2, stride=2))
        self.layer2 = nn.Sequential(
            nn.Conv2d(16, 32, kernel_size=5, stride=1, padding=2),
            nn.BatchNorm2d(32),
            nn.ReLU(),
            nn.MaxPool2d(kernel_size=2, stride=2))
        self.fc = nn.Linear(7 * 7 * 32, num_classes)

    def forward(self, x):
        out = self.layer1(x)
        out = self.layer2(out)
        out = out.reshape(out.size(0), -1)
        out = self.fc(out)
        return out


model = ConvNet(num_classes).to(device)

# Loss and optimizer
criterion = nn.CrossEntropyLoss()
optimizer = torch.optim.Adam(model.parameters(), lr=learning_rate)

# Train the model
```

(continues on next page)

(continued from previous page)

```
total_step = len(train_loader)
for epoch in range(num_epochs):
    for i, (images, labels) in enumerate(train_loader):
        images = images.to(device)
        labels = labels.to(device)

        # Forward pass
        outputs = model(images)
        loss = criterion(outputs, labels)

        # Backward and optimize
        optimizer.zero_grad()
        loss.backward()
        optimizer.step()

        if i % 30 == 0:
            print('Epoch [{}/{}], Step [{}/{}], '
                  'Loss: {:.4f}'.format(epoch + 1, num_epochs, i + 1,
                                        total_step, loss.item()))

            # aim - Track model loss function
            aim_run.track(loss.item(), name='loss', epoch=epoch,
                          context={'subset':'train'})

            correct = 0
            total = 0
            _, predicted = torch.max(outputs.data, 1)
            total += labels.size(0)
            correct += (predicted == labels).sum().item()
            acc = 100 * correct / total

            # aim - Track metrics
            aim_run.track(acc, name='accuracy', epoch=epoch, context={'subset': 'train'})

            if i % 300 == 0:
                aim_run.track(loss.item(), name='loss', epoch=epoch, context={'subset':
↪'val'})
                aim_run.track(acc, name='accuracy', epoch=epoch, context={'subset': 'val
↪'})


# Test the model
model.eval()
with torch.no_grad():
    correct = 0
    total = 0
    for images, labels in test_loader:
        images = images.to(device)
        labels = labels.to(device)
        outputs = model(images)
        _, predicted = torch.max(outputs.data, 1)
        total += labels.size(0)
```

(continues on next page)

(continued from previous page)

```
        correct += (predicted == labels).sum().item()

    print('Test Accuracy: {} %'.format(100 * correct / total))
```

17.5 Message size limitations

Aim Remote Tracking server uses gRPC as a transport layer. gRPC imposes message size limits on sending/receiving messages from/to server. Aim is configured to use 16MB message size limit by default. If you want to specify a different limit, use `__AIM_RT_MAX_MESSAGE_SIZE__` environment variable.

```
# max message size 128MB
export __AIM_RT_MAX_MESSAGE_SIZE__=134217728
```

Note: The message max size should be the same for both Aim Remote Tracking server and client code.

17.6 Conclusion

As you can see, aim remote tracking server allows running experiments on multiple hosts with simple setup and minimal changes to your training code.

CHAPTER

EIGHTEEN

HOST AIM ON KUBERNETES (K8S)

Since Aim can run as a local server through FastAPI, it can be deployed to a K8S cluster! Hosting Aim on K8S comes with several advantages:

- multiple users of your organization can access Aim in a single spot, which removes the need for ML practitioners to run Aim themselves
- Aim runs can be centralized on a remote volume, which provides additional support and encouragement for remote model training and monitoring
- a deployment to K8S abstracts away the Aim CLI, which empowers users to focus on the value provided by Aim (visualizations/applications vs. CLI `up` and `repo` understanding)

The following sections illustrate how to deploy and serve Aim on K8S. The sections assume:

- access to a cloud provider, such as GCP, AWS, or Azure
- a repository that can host Dockerfiles, such as Google Artifact Registry or Dockerhub
- ability/permissions to provision a `ReadWriteMany` volume, or bind an existing one to a K8S deployment

18.1 Dockerfile

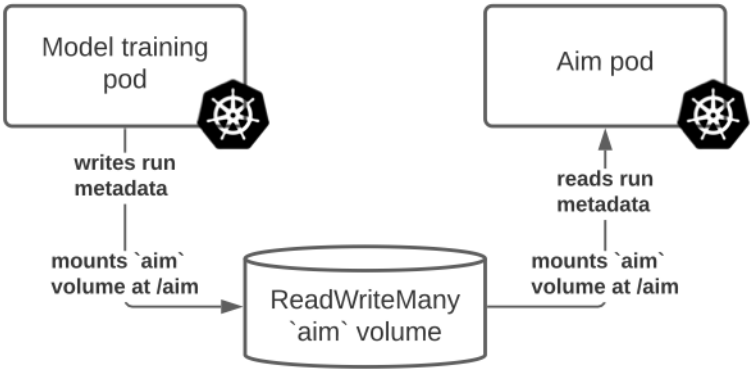
The following Dockerfile image should suffice for getting Aim running in a container:

```
# python3.7 should be sufficient to run Aim
FROM python:3.7

# install the `aim` package on the latest version
RUN pip install --upgrade aim

# make a directory where the Aim repo will be initialized, `/aim`
RUN mkdir /aim

ENTRYPOINT ["/bin/sh", "-c"]

# have to run `aim init` in the directory that stores aim data for
# otherwise `aim up` will prompt for confirmation to create the directory itself.
# We run aim listening on 0.0.0.0 to expose all ports. Also, we run
# using `--dev` to print verbose logs. Port 43800 is the default port of
# `aim up` but explicit is better than implicit.
CMD ["echo \"N\" | aim init --repo /aim && aim up --host 0.0.0.0 --port 43800 --workers␣
↪2 --repo /aim"]
```

Assuming you store the above in your current directory, the container can be built using `docker build . -t my-aim-container:1` and pushed to your repository with `docker push my-docker-repository.dev/deployments/aim:1`.

18.2 Volume

The core advantage of using a K8S volume to store Aim runs is that other K8S deployments can mount the same volume and store their runs on it! This way, the core Aim K8S deployment can read the new runs and display them to users who want to visualize their results. For example, one can have a deployment that performs model training and records Aim runs on the same volume that is mounted to the Aim deployment! This model is illustrated by the following diagram:



Generally, volumes that support have the `ReadWriteMany` property are manually provisioned, such as Filestore instances on Google Cloud or, generally, GlusterFS volumes. Once a disk is provisioned, it can be bound to a persistent volume via an IP. Assuming you can provision a disk like this on your cloud provider and obtain an IP, we can create a volume representation, along with a claim for it. The persistent volume (`aim-pv.yaml`) can be formulated as:

```
apiVersion: v1
kind: PersistentVolume
metadata:
  name: aim-runs
spec:
  capacity:
    storage: 1Ti # or whatever size disk you provisioned
  accessModes:
    - ReadWriteMany
  nfs:
    path: /aim
    server: 123.12.123.12  # add your own IP here
```

The persistent volume claim (`aim-pvc.yaml`) is:

```
apiVersion: v1
kind: PersistentVolumeClaim
metadata:
  name: aim-runs-claim
spec:
  accessModes:
```

(continues on next page)

(continued from previous page)

```
    - ReadWriteMany
  storageClassName: "" # if you have a custom storage class, use it! Otherwise, it's␣
↪`default`
  volumeName: aim-runs
  resources:
    requests:
      storage: 1Ti
```

These can be provisioned via:

```
> kubectl apply -f aim-pv.yaml
> kubectl apply -f aim-pvc.yaml
```

Once the volume is provisioned, we can mount it to our deployments!

18.3 Deployment

The main Aim deployment will have a single container that runs Aim. This deployment will mount the volume that was provisioned previously, and the main Aim repository will be initialized at the path the volume is mounted to. For example, if the volume is mounted to `/aim`, then the deployment will initialize and read Aim runs from that path. The K8S deployment is:

```
apiVersion: apps/v1
kind: Deployment
metadata:
  labels:
    app: my-aim-deployment
  name: my-aim-deployment
  namespace: default
spec:
  selector:
    matchLabels:
      app: my-aim-deployment
  strategy:
    rollingUpdate:
      maxSurge: 25%
      maxUnavailable: 25%
    type: RollingUpdate
  template:
    metadata:
      labels:
        app: my-aim-deployment
    spec:
      containers:
        image: my-docker-repository.dev/deployments/aim:1
        name: my-aim-deployment
        ports:
          - containerPort: 43800
            protocol: TCP
        resources:
          limits:
```

(continues on next page)

(continued from previous page)

```
            cpu: "1"
            memory: 2Gi
          requests:
            cpu: 100m
            memory: 1Gi
        volumeMounts:
          - mountPath: /aim
            name: aim-runs
      volumes:
        - name: aim-runs
          persistentVolumeClaim:
            claimName: aim-runs-claim
```

This K8S deployment:

- defines a pod with a single replica that runs the Aim server defined by the Dockerfile
- mounts the persistent volume `aim-run` through the `aim-run-claim` persistent volume claim
- the Dockerfile initializes the `/aim` directory as the Aim repo. Note that the Dockerfile already passes `N` to the confirmation prompt in case the repo is already initialized (this will be the case after the initial deployment creation, since the repo has to be initialized only once, but it's nice to avoid manual work)
- starts up the Aim server on port 43800, which reads all the runs stored at `/aim`

18.4 Service

Now that a deployment is deployed, the Aim server can be exposed through a K8S service! Depending on your cluster setup, you have several options for exposing the deployment. One option is to run:

```
> kubectl expose deployment my-aim-deployment --type=LoadBalancer --name=my-aim-service
```

Another alternative is to create the service definition yourself, and apply it. The definition (`aim-svc.yaml`) can be:

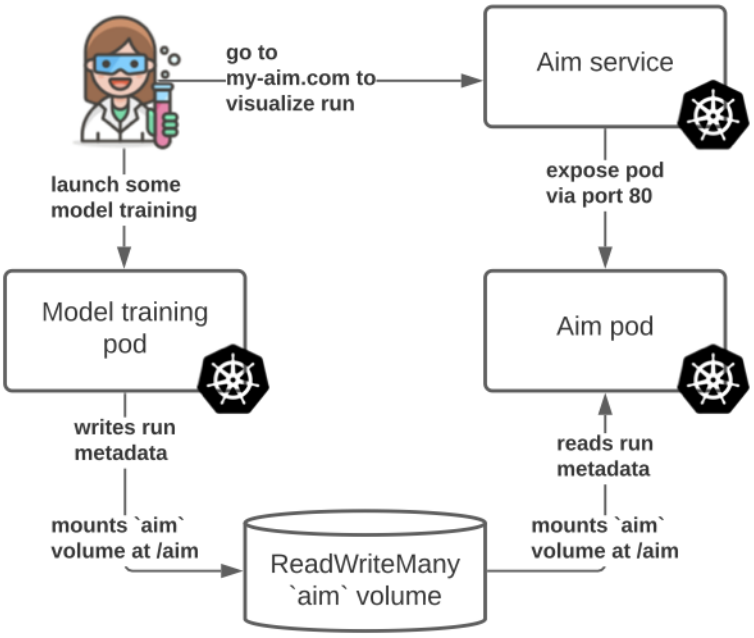
```
apiVersion: v1
kind: Service
metadata:
  name: my-aim-service
spec:
  selector:
    app: my-aim-deployment
  ports:
    - protocol: TCP
      port: 80
      targetPort: 43800
```

The service definition can be applied via:

```
> kubectl apply -f aim-svc.yaml
```

18.5 Conclusion

That's it! Now you have the following structure serving your users' Aim runs:



Assuming your users can submit a model training run to *some* pod/deployment that runs model training and has the right `aim` code to record a run at path `/aim`, your Aim deployment will be able to display the run the next time it performs a live update!

CHAPTER

NINETEEN

RUN AIM UI ON JUPYTER NOTEBOOK

Start your notebook with the following code to install Aim:

```
!pip install aim
```

Next, initialize a new run and save some hyperparameters:

```
from aim import Run

run = Run()

run['hparams'] = {
    'learning_rate': 0.001,
    'batch_size': 32,
}
```

Note: Do not forget to call run.finalize() once the training is over.

After tracking runs with `Run`, run the following commands in the notebook to run the Aim UI:

1. Load Aim extension for notebooks:

```
%load_ext aim
```

1. Run `%aim up` to open Aim UI in the notebook:

```
%aim up
```

CHAPTER

TWENTY

RUN AIM UI ON SAGEMAKER NOTEBOOK INSTANCE

In this guide you will learn how to run Aim UI on your Sagemaker Jupyter instance

Aim can be installed and used on Jupyter notebooks. With SageMaker notebook instance, there are difficulties with establishing POST requests which prevents some Aim functionalities to work. So we have built a script to update the proxying package on SageMaker notebook instance. In order to work with Aim UI, please follow the steps below:

- Create a lifecycle configuration on your SageMaker Service.
- Copy and paste the script in your configuration's `Start Notebook` phase.

For more information how to create a lifecycle configuration on AWS SageMaker Service, please go through the AWS SageMaker Documentation.

After creating a lifecycle configuration, attach it to your SageMaker instance and restart the instance.

It is possible to run this script manually from the terminal of your jupyter instance as well.

Note: Why this script is used for?

The script above will uptade the *jupyter-server-proxy* package on your notebook instance. To access a port on your instance (i.e. open the Aim UI) API requests and static files need to go through the proxy, which is created by the package (see the script). It will replace the existing proxy and make sure Aim and other such requests go through. Fore more information please go through the https://github.com/jupyterhub/jupyter-server-proxy/pull/328#issue-1145874348.

Once your notebook instance is successfully restarted you will be able to use Aim UI on the instance.

Next, install aim.

```
$ pip install aim
```

Initialize a new run and save some hyperparameters.

```
from aim import Run

run = Run()

run['hparams'] = {
    'learning_rate': 0.001,
    'batch_size': 32,
}
```

20.1 Using Terminal

In order to run Aim UI from a terminal, execute the following command:

```
$ aim up --base-path=/proxy/absolute/<your-port>/aim-sage
```

After running this command you will be able to open `<sagemker_instance>/proxy/absolute/<your-port>/aim-sage/` in your browser. The default port is `43800`.

It is possible to set `__AIM_PROXY_URL__` env variable, and `aim up` command will print out the generated url for Aim UI.

To find your proxy url, just copy your SageMaker URL and remove `/lab` postfix.

20.2 Using Notebook Extension

1. Load Aim extension for notebooks:

```
%load_ext aim
```

1. Run `%aim up` command to open Aim UI in the notebook:

```
%aim up --proxy-url=https://<instance>.notebook.<region>.sagemaker.aws
```

Will load the Aim UI in the notebook cell.

Will print out the proxy url under the loaded Aim UI. Use that URL to open Aim UI on your browser.

The default port is `43801` for notebook extension to prevent confusions. The `--port=<your-port>` argument is supported as well.

Note: In notebook extension, the only mandatory argument is `--proxy-url` when using it on SageMaker.

CHAPTER

TWENTYONE

INTEGRATION GUIDES

Aim integrates seamlessly with your favorite ML frameworks - Pytorch Ignite, Pytorch Lightning, Hugging Face and others. Basic integration guides can be found at Quick Start section.

In this section we're going to deep-dive into the ways we can extend the basic loggers, manipulate them to track a lot more. The basic loggers can track specific metrics and hyper-params only.

There are two ways Aim callbacks/adapters/loggers can be extended:

- by deriving and overriding the main methods that are responsible for logging.
- by using public property called `experiment` which gives access to underlying `aim.Run` object to easily track new metrics, params and other metadata that would benefit your project.

21.1 Pytorch Ignite

Both callback extension mechanisms are available with Pytorch Ignite. In the example below you'll see how to use the experiment property to track confusion matrix as an image using `aim.Image` after the training is completed.

Here is an example colab notebook.

```
from aim import Image
from aim.pytorch_ignite import AimLogger

import matplotlib.pyplot as plt
import seaborn as sns

# Create a logger
aim_logger = AimLogger()
...
@trainer.on(Events.COMPLETED)
def log_confusion_matrix(trainer):
    metrics = val_evaluator.state.metrics
    cm = metrics['cm']
    cm = cm.numpy()
    cm = cm.astype(int)
    classes = ['T-shirt/top','Trouser','Pullover','Dress','Coat','Sandal','Shirt',
↪'Sneaker','Bag','Ankle Boot']
    fig, ax = plt.subplots(figsize=(10,10))
    ax= plt.subplot()
    sns.heatmap(cm, annot=True, ax = ax,fmt="d")
    # labels, title and ticks
```

(continues on next page)

(continued from previous page)

```
    ax.set_xlabel('Predicted labels')
    ax.set_ylabel('True labels')
    ax.set_title('Confusion Matrix')
    ax.xaxis.set_ticklabels(classes,rotation=90)
    ax.yaxis.set_ticklabels(classes,rotation=0)
    aim_logger.experiment.track(Image(fig), name='cm_training_end')
```

With Pytorch Ignite there's also a 3rd approach to extend the integration. For example Pytorch Ignite's Tensorboard logger provides a possibility to track model's gradients and weights as histograms. Same can be achieved with Aim

```
from typing import Optional, Union

import torch.nn as nn
from ignite.contrib.handlers.base_logger import BaseWeightsHistHandler
from ignite.engine import Engine, Events

from aim.pytorch_ignite import AimLogger
from aim import Distribution


class AimGradsHistHandler(BaseWeightsHistHandler):
    def __init__(self, model: nn.Module, tag: Optional[str] = None):
        super(GradsHistHandler, self).__init__(model, tag=tag)

    def __call__(self, engine: Engine, logger: AimLogger, event_name: Union[str,␣
↪Events]) -> None:
        global_step = engine.state.get_event_attrib_value(event_name)
        context = {'subset': self.tag} if self.tag else {}
        for name, p in self.model.named_parameters():
            if p.grad is None:
                continue
            name = name.replace(".", "/")
            logger.experiment.track(
                Distribution(p.grad.detach().cpu().numpy()),
                name=name,
                step=global_step,
                context=context
            )

# Create a logger
aim_logger = AimLogger()

# Attach the logger to the trainer to log model's weights norm after each iteration
aim_logger.attach(
    trainer,
    event_name=Events.ITERATION_COMPLETED,
    log_handler=AimGradsHistHandler(model)
)
```

21.2 Pytorch Lightning

In the example provided in the Aim GitHub repo using PL + Aim there's already a reference how to customize an integration.

```
def test_step(self, batch, batch_idx):
    ...
    # Track metrics manually
    self.logger.experiment.track(1, name='manually_tracked_metric')
```

So you can track lots of metadata at each iteration of test step: images, texts, whatever is needed by you and supported by Aim.

21.3 Hugging Face

Here is how to extend the basic Hugging Face logger. Below is an example of a `CustomCallback` that's derived from the `AimCallback`. The main HF method here is the `on_log()` that's overriden.

This allows us to track any `str` object that is passed to `on_log()` method as `aim.Text`.

```
from aim.hugging_face import AimCallback
from aim import Text


class CustomCallback(AimCallback):
    def on_log(self, args, state, control,
               model=None, logs=None, **kwargs):
        super().on_log(args, state, control, model, logs, **kwargs)

        context = {
            'subset': self._current_shift,
        }
        for log_name, log_value in logs.items():
            if isinstance(log_value, str):
                self.experiment.track(Text(log_value), name=log_name, context=context)
```

21.4 TF/keras

Here is how to track confusion matrices with Aim while extending the default callback provided for `tf.keras`. We have taken and adapted this example. to Aim. Here is how it looks:

```
from aim.tensorflow import AimCallback

class CustomImageTrackingCallback(AimCallback):
    def __init__(self, data):
        super().__init__()
        self.data = data

    def on_epoch_end(self, epoch, logs=None):
        super().on_epoch_end(epoch, logs)
```

(continues on next page)

(continued from previous page)

```
        from aim import Image
        # Use the model to predict the values from the validation dataset.
        test_pred_raw = self.model.predict(test_images)
        test_pred = np.argmax(test_pred_raw, axis=1)

        # Calculate the confusion matrix.
        cm = sklearn.metrics.confusion_matrix(test_labels, test_pred)
        # Log the confusion matrix as an image summary.
        figure = plot_confusion_matrix(cm, class_names=class_names)
        cm_image = Image(figure)

        # Log the confusion matrix as an Aim image.
        self.experiment.track(cm_image,"Confusion Matrix", step=epoch)

aim_callback = CustomImageTrackingCallback()

model.fit(
    train_images,
    train_labels,
    epochs=5,
    verbose=0, # Suppress chatty output
    callbacks=[aim_callback],
    validation_data=(test_images, test_labels),
)
```

21.5 XGBoost

Here is how to override the `AimCallback` for XGBoost.

```
from aim import Text
from aim.xgboost import AimCallback


class CustomCallback(AimCallback):

    def after_iteration(self, model, epoch, evals_log):
        for data, metric in evals_log.items():
            for metric_name, log in metric.items():
                self.experiment.track(Text(log), name=metric_name)

        return super().after_iteration(model, epoch, evals_log)
```

CHAPTER

TWENTYTWO

OVERVIEW

Aim is built around several concepts allowing to make sure that it meets the following criteria:

- **Run data isolation**. Each training run process isolated in terms of data and do not require additional services to run.
- **Scalability**. Aim web app is able to handle 1000s of training runs. Starting from v3.4.0 Aim provides a Remote Tracking server allowing to run multiple parallel experiments in a distributed multi-host environment.
- **Flexibility**. Aim UI and query language allow users to select, group and filter the tracked data any way they want.

22.1 Aim Components

In order to understand how Aim works, lets take a quick look on a different components it has.

- **Aim Storage**. At its core Aim uses a custom-built storage, based on rocksdb. More details in 'Where is data collected?'. Data tracked by different training runs collected and indexed in an aim repository (`.aim`). The storage itself is generic; it allows accessing the data as collection of dictionaries and arrays.
- **Aim SDK**. On top of the storage Aim SDK provides functionality to track/select/query data. Additionally, SDK is a layer used by Web APIs and CLI.
- **Aim UI**. Web app allowing to browse run metadata, metrics, images and other tracked data.
- **Aim CLI**. A collection of command line utilities for running Aim web server, managing aim repositories, runs, etc.
- **Remote Tracking server**. A gRPC-based service accepting incoming traffic and storing data on a centralized server.

The next sections will describe various concepts Aim introduces and provide more detailed look on individual components introduced above.

CHAPTER

TWENTYTHREE

DATA STORAGE - WHERE AIM DATA IS COLLECTED

This section provides a deep-dive into Aim storage structure. It is important to know the internal storage organization in order to understand how it affects queries performance.

23.1 Storage structure

The core foundation for Aim storage is rocksdb. It is a fast, embedded key-value store maintained by facebook. The aim repository is a collection of individual rocksdb databases with abstraction layers added to manage the collection as one database. The abstraction of a single KV store called `Container`. Below is the directory structure for a typical Aim project:

```
.aim
    run_metadata.sqlite
    meta/
        index/
        chunks/
            aacf48e769534c32a9cc5a3c/
            80483ab611a24bf5bd8fc288/
            16f83a2c2f50477f8446f322/
        progress/
            aacf48e769534c32a9cc5a3c
            80483ab611a24bf5bd8fc288
        locks/
            aacf48e769534c32a9cc5a3c
            80483ab611a24bf5bd8fc288
    seqs/
        chunks/
            aacf48e769534c32a9cc5a3c/
            80483ab611a24bf5bd8fc288/
            16f83a2c2f50477f8446f322/
        progress/
            aacf48e769534c32a9cc5a3c
            80483ab611a24bf5bd8fc288
        locks/
            aacf48e769534c32a9cc5a3c
            80483ab611a24bf5bd8fc288
```

There are two main parts of the storage:

- `run_metadata.sqlite`: SQLite database for storing `Run` structured data, such as, creation time, name, `Experiment` it is attached to, `Tag`s etc.

- `meta/` and `seqs/` directories: a collection of rocksdb storages. Used to write `Run`s tracked data, such as params, metrics and objects.

In the tree above the hash-strings (i.e. `aacf48e769534c32a9cc5a3c`) represent a single `Run`. When the new `Run` is started, aim will create two Containers:

- **Meta** container for logged params as well as metadata about collected sequences, contexts, etc.
- **Sequence** container for the value series.

The reason the actual sequence data separated from metadata is the necessity for fast queries, regardless sequence size.

Additionally, per each container two files will be created to properly manage the container state

- **lock** file, indicating that the container opened in write-mode.
- **progress** file, indicating that container potentially has un-indexed data.

This setup allows implementing concurrent training jobs setup without requiring additional synchronization routines, and without the risk of losing or corrupting the data. For example, two jobs might run on different hosts where aim repo mounted on a shared NFS location.

23.2 What is the index container?

Each run writes data into its own isolated containers. The aim queries require reading the Run metadata from **meta** container. However, with 1000s of runs opening each meta container database will slow-down the queries. Here the indexing of metadata becomes crucial.

`Run` object maintains lock for both **meta** and **sequence** containers during the training script execution. Run will continue to write its data into its own containers. Once the execution finishes, **meta** container data indexed, the container locks released, and the progress file removed. The **sequence** container data is not indexed, since the individual points of a sequence are not queryable, and the sequence info is available in **meta** container.

23.3 How data written to/read from the storage?

`Run` object provides interface for logging the Run parameters dict-like data and tracking series of scalars and objects. Aim has a custom encoding layer which translates this hierarchical data into the sets of key-value pairs to be written into rocksdb. The same encoding layer is responsible for re-constructing the tracked data/objects. During query execution, aim SDK will walk through all runs in `index` container + the **meta** containers for chunks which have progress file (remember that progress file indicates potentially un-indexed data for the Run). If the run/sequence match the query expression, the appropriate run/sequence will be yielded. Notice that till this point no data was accessed from the **sequence** container. The sequence data itself is read upon request.

CHAPTER

TWENTYFOUR

STORAGE INDEXING - HOW AIM DATA IS INDEXED

24.1 Background

When tracking experiment metadata with Aim, each run creates its own isolated space in aim repository. This allows to run multiple concurrent experiments without setting-up additional services responsible for data writes synchronization. Once run is complete, all the data it tracked is being indexed. We call this step run finalization. When the training script terminated with `SIGTERM` signal, Aim will handle this and make sure that run properly finalized and data is indexed. However, there are cases when training terminated abnormally and data remains unindexed.

24.2 How things worked before?

Due to the chunks of data being unindexed, chunks of data would remain in the runs' separate storage but not in index storage. This means that queries had to open multiple files to read the repo data. Once failed runs started to accumulate, queries will slow down. In order to mitigate this `aim reindex` command has been introduced. The command will scan the aim repo and index all stalled runs.

24.3 Automatic indexing

Though `aim reindex` command will address the performance issues it is not the most convenient way to do. The questions such as "When should I run `aim reindex`?" or "How frequent should I run `aim reindex`?" depend on the actual aim repository and use-case. Thus, we need to automate the indexing of aim repository. Each time `aim up` command is ran, Aim will spawn a background thread along with the web server. The thread will check for the unindexed runs and reindex them one at the time. This will keep queries performance high without locking the index storage for too long.

24.4 Conclusion

With the new automatic indexing logic in place, users don't have to manually run `aim reindex` command. It is still in place for cases when all the runs data should be indexed at once. The combination of automatic (implicit) and manual (explicit) reindexing makes sure aim repo has good performance in a long-term usage screnarios and provides good overall user experience.

CHAPTER

TWENTYFIVE

CONCEPTS

All the functionality in Aim is build around several key concepts. This chapter will give a brief overview of these core concepts. For more details please check the Reference section or Glossary.

25.1 Aim Run

Run is an abstraction representing the tracked data for a single experiment. Its in memory model is SDK class `aim.Run`. It is a core class used in your training script for tracing metrics and objects, as well as storing training hyperparams and other data. Run object are queryable and UI provides a rich functionality for exploring runs and browsing single run details.

25.2 Aim Repo

While you do multiple training experiments, multiple runs data stored in a single directory called Aim repository (repo for short). You can think of aim repo as an application centralized database. SDK provides an in memory model for repo `aim.Repo` class. It is responsible for repository resources management and might be used to query and/or iterate over the stored data.

25.3 Run Params

Each run has a set of parameters associating with it. This might include the training script hyperparameters, dataset information, etc. The Run object provides dictionary-like interface to set and access run params. Run parameters are also available in the context of queries. You can set the whole configuration at once with the syntax like this:

```
run['hparams'] = conf
```

At this moment `Run` supports setting configuration from Python dictionaries and `OmegaConf` configs. Support of popular configuration formats constantly added. You can check the full list in Supported Data types section.

25.4 Run Sequence

The sequence is a set of homogeneous ordered objects. In aim sequence must be bound to the Run object. When the value is tracked in aim, it is appended to an existing or newly created Sequence object. The entire sequences can be queried using aim QL and each sequence can be sliced further down. Sequence object is agnostic to the element type it holds. The way how the sequence represented in UI, and the set of additional operations it might have depends on the element type. For example Metric is a sequence of scalars. It can be represented as a value chart in UI, and SDK provides methods to convert it to `numpy.ndarray`.

25.5 Sequence Context

The sequence context provides a mechanism to query/group multiple sequences beyond simple string comparison on sequence name. Sequences with the same name but with different context can perfectly coexist in the scope of one Run. In other words, sequence defined by its Run, name and context. The example usage of this is tracking the same metric 'loss' for different stages of training (train, validation, test). The resulting Run will have 3 metrics: 1) `'loss' {'subset': 'train'}` 2) `'loss' {'subset': 'val'}` 3) `'loss' {'subset': 'test'}` Here is a small code example demonstrating how to specify context for a metric sequence:

```
from aim import Run

aim_run = Run()
for i in range(100):
    if i % 2 == 0:
        aim_run.track(i, name=r'numbers', context={'odds': True})
    else:
        aim_run.track(i, name=r'numbers', context={'odds': False})
```

CHAPTER

TWENTYSIX

RUNNING AIM WITH PROFILING

Aim comes with profiling feature which is logging all api requests to the backend into a directory inside your repository. We use pyinstrument as underlying profiler. To toggle profiling, run `aim up` with `--profiler` flag.

```
$ aim up --profiler
```

This will instruct backend to create a new directory inside your repository (`.aim/profler`). On every api call, profiler will create an `.html` file containing whole run trace of that api. Basically you can navigate into that directory and open the file in your browser to see where's the performance bottleneck.

26.1 Why would you need to enable the profiling.

Well you don't, unless we explicitly ask you to in case you have any performance issues while using Aim.

Please note that no data is sent to us when profiling is toggled. Everything is stored locally and managed by the end user.

CHAPTER

TWENTYSEVEN

TRACK AND COMPARE GANS WITH AIM

27.1 Overview

Generative Adversarial Networks, or GANs, are deep-learning-based generative models.

Generative modeling is an unsupervised learning task in machine learning that involves automatically discovering and learning the patterns of input data in such a way that the model can be used to generate new examples that plausibly could have been drawn from the original dataset.

In this guide we will show you how to integrate Aim with your GAN and GAN with EMA to compare the generated images from both experiments to compare their performances.

27.2 Experiment

We will train and compare a regular GAN vs GAN with EMA. EMA is a technique for parameter averaging in GAN training, which computes an exponentially discounted sum of weights.

We will use lightweight-gan model implemented by lucidrains and MetFaces Dataset as a training dataset.

To be able to analyze the results we will fix random 64 points and track them during the training both for a regular GAN and GAN w/ EMA.

27.3 Track images with Aim

1. Initialize a new run in the trainer class to collect and store sequences of images:

```
class Trainer():
    def __init__(
        self,
        name = 'default',
        results_dir = 'results',
        models_dir = 'models',
                ...
        ):
    ...
    self.run = aim.Run()           # Initialize aim.Run
    self.run['hparams'] = hparams  # Log hyperparams
    ...
```

Code on GitHub

1. Track images generated by a regular GAN:

```
# Regular GAN

# Get generated images
generated_images = self.generate_(self.GAN.G, latents)

aim_images = []
for idx, image in enumerate(generated_images):
    ndarr = image.mul(255).add_(0.5).clamp_(0, 255).permute(1, 2, 0).to('cpu', torch.
↪uint8).numpy()
    im = PIL.Image.fromarray(ndarr)
    aim_images.append(aim.Image(im, caption=f'#{idx}'))

# Store with Aim (name="generated" and context.ema=0)
self.run.track(value=aim_images, name='generated', step=self.steps, context={'ema':␣
↪False})
```

Code on GitHub

1. Track images generated by a GAN with enabled EMA:

```
# GAN with moving averages

# Get generated images
generated_images = self.generate_(self.GAN.GE, latents)

aim_images = []
for idx, image in enumerate(generated_images):
    ndarr = image.mul(255).add_(0.5).clamp_(0, 255).permute(1, 2, 0).to('cpu', torch.
↪uint8).numpy()
    im = PIL.Image.fromarray(ndarr)
    aim_images.append(aim.Image(im, caption=f'EMA #{idx}'))

# Store with Aim (name="generated" and context.ema=1)
self.run.track(value=aim_images, name='generated', step=self.steps, context={'ema': True}
↪)
```

Code on GitHub

27.4 Explore the results with Aim UI

1. Visualize images generated by a regular GAN:

1. Visualize images generated by GAN with EMA:

As you may notice GAN with EMA converges in an exponential fashion and has better results at the end.

1. Let's compare the final step of the two methods side by side:

27.5 Conclusion

As you can see GAN with EMA performed much better compared to the regular one.

With Aim you can easily compare diff groups of tracked images from diff runs.

Group them by the run hash, other parameters available to slice and dice and observe the difference between the runs.

CHAPTER

TWENTYEIGHT

AIM CLI

Aim CLI offers a simple interface to easily organize and record your experiments. Paired with the Python Library, Aim is a powerful utility to record, search and compare AI experiments. Here are the set of commands supported:

Command	Description
`init`	Initialize the `aim` repository.
`version`	Displays the version of aim cli currently installed.
`up`	Runs Aim web UI for the given repo.
`upgrade`	Upgrades legacy Aim repository from `2.x` to `3.0`.
`reindex`	Process runs left in 'in progress' state and optimized finished runs.
`server`	Run `aim` remote tracking server accepting incoming RPC requests. *Experimental feature.*
`runs`	Manage run data for the given repo.
`convert`	Tool-set for converting 3rd party data into Aim readable format.

28.1 init

*******This step is optional.*** Initialize the aim repo to record the experiments.

```
$ aim init
```

Creates `.aim` directory to save the recorded experiments to. Running `aim init` in an existing repository will prompt the user for re-initialization.

Args	Description
`--repo <repo_path>`	Path to parent directory of `.aim` repo. *Current working directory by default.*

*******Beware:*** Re-initialization of the repo clears `.aim` folder from previously saved data and initializes new repo.
*******Note:*** This command is not necessary to be able to get started with Aim as aim is automatically initializes with the first aim function call.

28.2 version

Display the Aim version installed.

```
$ aim version
```

28.3 up

Start the Aim web UI locally.

```
$ aim up [ARGS]
```

Args	Description
-h	--host <host>	Specify host address.
-p	--port <port>	Specify port to listen to.
--repo <repo_path>	Path to parent directory of .aim repo. *Current working directory by default.*
--dev	Run UI in development mode.
--profiler	Enables API profiling which logs run trace inside .aim/profiler directory.
--log-level	Specifies log level for python logging package. *``WARNING``by default, ``DEBUG``when ``–dev``option is provided.*

28.4 upgrade

Upgrade Aim repository containing data logged with older version of Aim.

```
$ aim upgrade [ARGS] SUBCOMMAND
```

Args	Description
--repo <repo_path>	Path to parent directory of .aim repo. *Current working directory by default.*

upgrade subcommands

Upgrade aim repository from 2.x to 3.0.

```
$ aim ugrade 2to3 [ARGS]
```

Args	Description
--skip-failed-runs	Use this flag to skip runs which are failed/have missing or incomplete data.
--skip-checks	Use this flag to skip new repository consistency checks.
--drop-existing	Use this flag to clear old .aim directory. By default old data is kept in .aim_legacy.

28.5 reindex

Update index to include all runs in Aim repo which are left in progress.

```
$ aim reindex [ARGS]
```

Args	Description
`--repo <repo_path>`	Path to parent directory of `.aim` repo. *Current working directory by default.*
`--finalize-only`	Only finalize runs left in 'in progress' state. Do not attempt runs optimization.

28.6 server

Run a gRPC server to collect tracked data from remote clients.

```
$ aim server [ARGS]
```

Args	Description
`--repo <repo_path>`	Path to parent directory of `.aim` repo. *Current working directory by default.*
`-h &#124; --host <host>`	Specify host address.
`-p &#124; --port <port>`	Specify port to listen to. *Default is 53800.*
`-w &#124; --workers <N>`	Specify number of gPRC workers. *Default is 1 worker.*
`--ssl-keyfile`	Specify path to keyfile for secure connection.
`--ssl-certfile`	Specify path to cert. file for secure connection.
`--log-level`	Specifies log level for python logging package. *``WARNING`` by default.*

28.7 runs

Upgrade Aim repository runs data.

```
$ aim runs [ARGS] SUBCOMMAND
```

Args	Description
`--repo <repo_path>`	Path to parent directory of `.aim` repo. *Current working directory by default.*

runs subcommands

Sub-command	Description
`ls`	List runs in `aim` repository.
`rm`	Remove run data for given runs hashes. At lease one run should be specified.
`cp`	Copy run data for given runs hashes. At lease one run should be specified.
`mv`	Move run data for given runs hashes. At lease one run should be specified.

Global expression (*) support is available for run hashes. If hash contains *, it must be enclosed within quotes (`''`) as bash resolves the expression before passing it to `aim runs` command.

```
$ aim runs ls
```

```
$ aim runs rm [HASH] ...
```

```
$ aim runs cp [ARGS] [HASH] ...
```

Args	Description
`--destination <dest_repo_path>`	Path to destination repo. **Required.**

```
$ aim runs mv [ARGS] [HASH] ...
```

Args	Description
`--destination <dest_repo_path>`	Path to destination repo. **Required.**

28.8 convert

Tool-set for converting 3rd party data into Aim readable format.

```
$ aim convert [ARGS] SUBCOMMAND
```

Args	Description
`--repo <repo_path>`	Path to parent directory of `.aim` repo. *Current working directory by default.*

convert subcommands

Sub-command	Description
`tf`	Convert from TensorFlow events.
`mlflow`	Convert from MLFlow logs.

Sub-command: tf

Options	Description
`--flat`	Disregard context directory and treat them as distinct run directories. Inactive by default.

Sub-command: mlflow

Options	Description
`--tracking_uri`	MLFlow logs URI. Can be either an HTTP/HTTPS URI for a remote server, a database connection string, or a local path.
`-e &#124; --experiment`	MLFlow Experiment name. If specified, only runs for `exp_name` will be converted.

CHAPTER

TWENTYNINE

AIM SDK

CHAPTER

THIRTY

AIM STORAGE

30.1 aim.storage.arrayview module

CHAPTER

THIRTYONE

ANONYMIZED TELEMETRY

We constantly seek to improve Aim for the community. Telemetry data helps us immensely by capturing anonymous usage analytics and statistics. You will be notified when you run `aim up`. The telemetry is collected only on the UI. The python package **does not** have any telemetry associated with it.

31.1 Motivation

Aim UI uses segment's analytics toolkit to collect basic info about the usage:

- Anonymized stripped-down basic usage analytics;
- Anonymized number of experiments and run. We constantly improve the storage and UI for performance in case of many experiments. This type of usage analytics helps us to stay on top of the performance problem.

Note: No analytics is installed on the Aim Python package.

31.2 How to opt out

You can turn telemetry off by setting the `AIM_UI_TELEMETRY_ENABLED` environment variable to `0`.

CHAPTER

THIRTYTWO

CHANGELOG

32.1 3.9.1 Apr 29, 2022

32.1.1 3.9.1 Apr 29, 2022 - Enhancements:

- Add `Notes Tab` to single run page (arsengit)
- Add the run name to the batch delete and the batch archive modals (VkoHov)
- Increase the scalability of rendering lines in charts (KaroMourad)
- Increase live update requests delay to prevent performance issues (rubenaprikyan)
- Change font-family to monospace in the Table component (arsengit)
- Add info massage for single value sliders (VkoHov)
- Add `--log-level` argument for aim up/server commands (mihran113)
- Add notes backend api interface (devfox-se)

32.1.2 3.9.1 Apr 29, 2022 - Fixes:

- Fix LineChart y-dimension margin calculation (KaroMourad)
- Fix HighPlot lines partially rendering issue (KaroMourad)
- Fix HighPlot axis ticks overlapping issue (KaroMourad)
- Fix sorting Params/Scatters explorer axis ticks (KaroMourad)
- Fix compatibility with pytorch-lightning v1.6.0 (mihran113)
- Fix the image's original size cropping (VkoHov)
- Fix `PATH` related issues for `alembic` and `uvicorn` (mihran113)
- Fix queries for custom object APIs (mihran113)
- Fix chart height updating when resize mode changed (VkoHov)
- Fix HuggingFace callback context capturing (mihran113)
- Fix Params/Scatters explorers' row hiding functionality (VkoHov)
- Fix Profiler logs are saved outside repo directory (devfox-se)

32.2 3.8.1 Apr 6, 2022

- Encode run hash before including in CSS selectors (Hamik25)
- Fix displaying incorrect metric values for large range scale in LineChart (KaroMourad)
- Fix issue with rendering lines for large range scale in LineChart (KaroMourad)
- Fix issue with URL state sync for bookmarks (roubkar)
- Fix issue with displaying negative param values on Aim UI (roubkar)
- Fix row hiding functionality (roubkar)
- Tune RunOverviewTab container styles (arsengit)
- Update documentations links on UI (rubenaprikyan)
- Fix `RepoIndexManager` run's reference cleanup (mihran113)
- Fix remote run finalization (mihran113)
- Fix issue with fetch on load more (infinite scroll) functionality in Runs Explorer (rubenaprikyan)

32.3 3.8.0 Mar 26, 2022

32.3.1 3.8.0 Mar 26, 2022 - Enhancements:

- Hugging Face adapter refactoring (mihran113)
- Add run description columns to all run specific tables (VkoHov, mihran113)
- Change images rendering optimization default value to smoother (VkoHov)
- Set default steps ordering to desc in single run tabs (VkoHov, devfox-se)
- Add run name to grouping, ordering and run navigation popovers (VkoHov)
- Add ability to apply color scale on columns with numeric values (VkoHov)
- Refactored XGBoost AimCallback (devfox-se)
- Reopenable callbacks for integrations (mihran113)
- Add DVC integration (devfox-se)
- Add API profiler and unified API error response (devfox-se)
- Add API to retrieve N'th step of sequence (devfox-se)

32.3.2 3.8.0 Mar 26, 2022 - Fixes:

- Fix issue with calculation of active point on mouse hover in the LineChart (KaroMourad)
- Fix issue with wrong URL caching for Explorer pages (roubkar)
- Fix issue with focusing on the chart active point while moving the cursor (KaroMourad)
- Fix the image full view toggle icon visibility if the image has a white background (VkoHov)
- Fix scroll to the end of the audio tab (VkoHov)
- Add scrollbar to image full view mode content (VkoHov)

- Fix issues with run name/description not being set (mihran113)
- Fix issue with run single page tabs result caching (mihran113)
- Fix git system param tracking (devfox-se)
- Fix runs manual closing (mihran113)
- Fix Docker image creation step in packaging workflow (alberttorosyan)
- Fix Jinja2 template rendering with starlette==0.14.2 (alberttorosyan)

32.4 3.7.5 Mar 18, 2022

- Add request aborting functionality in single run page tabs (arsengit)
- Render plotly figures properly in single run page (arsengit)

32.5 3.7.4 Mar 15, 2022

- Fix density min and max validation calculation (VkoHov)

32.6 3.7.3 Mar 14, 2022

- Add missing names for dynamically imported files in single run page (arsengit)

32.7 3.7.2 Mar 10, 2022

- Fix issue with rendering UI re keeping long URL (KaroMourad)
- Split code in the single run page to optimize chunk size (arsengit)

32.8 3.7.1 Mar 10, 2022

- Fix metric queries with epoch=None (alberttorosyan)

32.9 3.7.0 Mar 9, 2022

32.9.1 3.7.0 Mar 9, 2022 - Enhancements:

- Add Run overview tab in run single page (arsengit, VkoHov, KaroMourad, rubenaprikyan)
- Custom max message size for Aim Remote tracking (alberttorosyan)
- Docker images for aim up/server (alberttorosyan)
- TF/Keras adapters refactoring (mihran113)
- Remote tracking client-side retry logic (aramaim)

- Add record_density to initial get-batch request for figures (VkoHov)

32.9.2 3.7.0 Mar 9, 2022 - Fixes:

- Fix rendering new lines in texts visualizer (arsengit)

32.10 3.6.3 Mar 4, 2022

- Fix UI rendering issue on colab (rubenaprikyan)

32.11 3.6.2 Mar 2, 2022

- Fix chart interactions issue in the Single Run Page Metrics tab (roubkar)
- Fix `resolve_objects` in remote tracking client subtree (alberttorosyan)
- Reject `0` as step/record count (alberttorosyan, VkoHov)
- Fix error on mlflow conversion by experiment id (devfox-se)

32.12 3.6.1 Feb 25, 2022

- Fix issue with aligning x-axis by custom metric (KaroMourad)
- Add `__AIM_PROXY_URL__` env variable to see full proxy url when running `aim up` command(rubenaprikyan)
- Add `--proxy-url` argument to notebook extension's `%aim up` to render UI correctly if there is a proxy server (rubenaprikyan)
- Add SageMaker integration, `jupyter-server-proxy` s bug-fix script (rubenaprikyan, mahnerak)
- Fix animation support in Plotly visualization and figure loading performance (Hamik25, mihran113)
- Display `None` values in group config column (VkoHov, Hamik25)
- Fix rendering issue on `Select` form search suggestions list (arsengit)
- Fix PL.AimLogger save_dir AttributeError (GeeeekExplorer)
- Remove `__example_type__` substring from param name (VkoHov)

32.13 3.6.0 Feb 22 2022

32.13.1 3.6.0 Feb 22 2022 - Enhancements:

- Sort params columns in alphabetical order (arsengit)
- Add illustrations for indicating explorer search states (arsengit)
- Ability to export chart as image (KaroMourad)
- Ability to group by metric.context (VkoHov)
- Tune manage columns items highlighting styles (VkoHov)

- Set active style on table actions popover buttons with applied changes (arsengit)
- Unification of Run Custom Object APIs (alberttorosyan, VkoHov)
- Aim repo runs data automatic indexing (alberttorosyan)
- Pytorch Lightning adapter refactoring (mihran113)
- Add Pytorch Ignite integration (mihran113)
- Add wildcard support for `aim runs` subcommands (mihran113)
- Add MLflow logs conversion command (devfox-se)
- Add CustomObject implementation for `hub.dataset` (alberttorosyan)

32.13.2 3.6.0 Feb 22 2022 - Fixes:

- Fix live updated data loss after triggering endless scroll (VkoHov)
- Fix system metric columns pinning functionality and grouping column order (arsengit)
- Fix system metrics search in manage columns popover (VkoHov)
- Fix queries on remote repos (mihran113)
- Fix incorrect boolean value formatting (VkoHov)

32.14 3.5.4 Feb 15 2022

- Fix batch archive functionality (VkoHov)
- Add repo lock/release feature (devfox-se)

32.15 3.5.3 Feb 11 2022

- Fix rendering issue in runs explorer page (arsengit)

32.16 3.5.2 Feb 10 2022

- Fix issue with displaying current day activity cell on week's first day (rubenaprikyan)
- Fix issue with filtering options while typing in input of autocomplete in Tooltip and Grouping popovers (rubenaprikyan)

32.17 3.5.1 Feb 4 2022

- Fix folder creation when tracking with remote tracker (aramaim)

32.18 3.5.0 Feb 3 2022

32.18.1 3.5.0 Feb 3 2022 - Enhancements:

- Ability to hide system metrics from table (arsengit)
- Add input validations to range selectors (Hamik25)
- Improve media panel rendering performance on hovering over images (KaroMourad)
- Add ability to parse and import TensorFlow events into aim (devfox-se)
- Add system parameter logging: CLI, Env, Executable, Git, Installed packages (devfox-se)
- Convert nested non-native objects (e.g. OmegaConf config instance) upon storing (devfox-se)
- Add cli subcommands cp and mv for aim runs command (mihran113)
- Add handler for matplotlib figures in Image and Figure custom objects (devfox-se)
- Improve highlighting of table focused/hovered/selected row (VkoHov)

32.18.2 3.5.0 Feb 3 2022 - Fixes:

- Fix stalled runs deletion (mihran113)
- Fix background transparency in colab when using dark mode of system (rubenaprikyan)
- Fix Grouping and Tooltip popovers states' resetting issue when live-update is on (rubenaprikyan)
- Fix table column's sort functionality issue in Params and Scatters Explorers (rubenaprikyan)

32.19 3.4.1 Jan 23 2022

- Fix issue with displaying experiment name in Images Explorer table (VkoHov)

32.20 3.4.0 Jan 22 2022

- Add ability to apply group stacking on media elements list (KaroMourad)
- Add ability to apply sorting by run creation_time on table rows (roubkar)
- Add ability to filter texts table with keyword matching (roubkar, rubenaprikyan)
- Add ability to delete run from settings tab (Hamik25)
- Enhance controls states of explorer pages (arsengit)
- Add –repo, –host arguments support for notebook extension (VkoHov, rubenaprikyan)
- Add trendline options to ScatterPlot (roubkar)

- Add ability to display images in original size and align by width (arsengit)
- Add version, docs and slack links to sidebar (arsengit)
- Enhance AudioPlayer component (arsengit)
- Recover active tab in run details page after reload (roubkar)
- Add ability to archive or delete runs with batches (VkoHov)
- Remote tracking server [experimental] (alberttorosyan, mihran113, aramaim)
- Add ability to change media elements order (VkoHov)
- Add ability to hard delete runs (alberttorosyan)
- Lossy format support for aim.Image (devfox-se)
- Timezone issues fix for creation and end times (mihran113)

32.21 3.3.5 Jan 14 2022

- Add non-strict write mode to replace not-yet-supported types with their string representations. (mahnerak)
- Log pytorch_lightning hyperparameters in non-strict mode. (mahnerak)

32.22 3.3.4 Jan 10 2022

- Fix issue with WAL files flushing (alberttorosyan)
- Support for omegaconf configs in pytorch_lightning adapter (devfox-se)

32.23 3.3.3 Dec 24 2021

- Fix issue with showing range panel in Images Explorer (roubkar)

32.24 3.3.2 Dec 20 2021

- Fix issue with not providing point density value to live-update query (rubenaprikyan)

32.25 3.3.1 Dec 18 2021

- Fix getValue function to show correct chart title data (KaroMourad)

32.26 3.3.0 Dec 17 2021

- Add ability to track and explore audios in run detail page (arsengit, VkoHov, devfox-se)
- Add ability to track and visualize texts (mihran113, roubkar)
- Fix boolean values encoding (mahnerak)
- Add Scatter Explorer to visualize correlations between metric last value and hyperparameter (KaroMourad)
- Add ability to track and visualize plotly objects (devfox-se, Hamik25, rubenaprikyan)
- Add ability to query distributions by step range and density (VkoHov, rubenaprikyan)
- Add colab notebook support (mihran113, rubenaprikyan)
- Implement images visualization tab in run detail page (VkoHov, KaroMourad)
- Add custom URL prefix support (mihran113, Hamik25, roubkar)
- Enhance metric selection dropdowns to see lists in alphabetical order (rubenaprikyan)

32.27 3.2.2 Dec 10 2021

- Fix Run finalization index timeout issue (alberttorosyan)

32.28 3.2.1 Dec 8 2021

- Add ability to provide custom base path for API (mihran113, roubkar)
- Fix table groups column default order (arsengit)
- Fix table panel height issue in runs explorer page (arsengit)

32.29 3.2.0 Dec 3 2021

- Add ability to cancel pending request (roubkar, arsengit)
- Add support for secure protocol for API calls (mihran113, roubkar)
- Implement image full size view (VkoHov)
- Add ability to manipulate with image size and rendering type (arsengit)
- Enhance Table column for selected grouping config options (arsengit)
- Implement suggestions list for AimQL search (arsengit, rubenaprikyan)
- Add ability to track and visualize distributions (mihran113, rubenaprikyan)
- Add notebook extension, magic functions (rubenaprikyan)

32.30 3.1.1 Nov 25 2021

- Apply default ordering on images set (VkoHov)
- Ability to show image data in a tooltip on hover (KaroMourad)
- Support of Image input additional data sources (alberttorosyan)
- Ability to export run props as pandas dataframe (gorarakelyan)
- Slice image sequence by index for the given steps range (alberttorosyan)
- Improve Images Explorer rendering performance through better images list virtualization (roubkar)

32.31 3.1.0 Nov 20 2021

- Add ability to explore tracked images (VkoHov)
- Improve rendering performance by virtualizing table columns (roubkar)
- Add ability to apply grouping by higher level param key (roubkar)
- Add ability to specify repository path during `aim init` via `--repo` argument (rubenaprikyan)

32.32 3.0.7 Nov 17 2021

- Fix for missing metrics when numpy.float64 values tracked (alberttorosyan)

32.33 3.0.6 Nov 9 2021

- Fix for blocking container optimization for in progress runs (alberttorosyan)

32.34 3.0.5 Nov 9 2021

- Add tqdm package in setup.py required section (mihran113)

32.35 3.0.4 Nov 8 2021

- Switch to aimrocks 0.0.10 - exposes data flushing interface (mihran113)
- Optimize stored data when runs finalized (mihran113)
- Update `aim reindex` command to run storage optimizations (alberttorosyan)
- Storage partial optimizations on metric/run queries (alberttorosyan)

32.36 3.0.3 Nov 4 2021

- Bump sqlalchemy version to 1.4.1 (alberttorosyan)

32.37 3.0.2 Oct 27 2021

- Switch to aimrocks 0.0.9 - built on rocksdb 6.25.3 (alberttorosyan)
- Remove grouping select options from Params app config (VkoHov)
- Sort metrics data in ascending order for X-axis (KaroMourad)

32.38 3.0.1 Oct 22 2021

- Check telemetry_enabled option on segment initialization (VkoHov)
- Draw LineChart Y-axis (horizontal) tick lines on zooming (KaroMourad)
- Sort select options/params based on input value (roubkar)
- Fix query construction issue for multiple context items (roubkar)
- Fix issue with making API call from Web Worker (VkoHov)

32.39 3.0.0 Oct 21 2021

- Completely revamped UI:
  - Runs, metrics and params explorers
  - Bookmarks, Tags, Homepage
  - New UI works smooth with ~500 metrics displayed at the same time with full Aim table interactions
- Completely revamped storage:
  - 10x faster embedded storage based on Rocksdb
  - Average run query execution time on ~2000 runs: 0.784s
  - Average metrics query execution time on ~2000 runs with 6000 metrics: 1.552s

32.40 2.7.1 Jun 30 2021

- Fix bookmark navigation issue (roubkar)
- Empty metric select on X-axis alignment property change (roubkar)

32.41 2.7.0 Jun 23 2021

- Add ability to export table data as CSV (KaroMourad)
- Add ability to bookmark explore screen state (roubkar)
- Add dashboards and apps API (mihran113)

32.42 2.6.0 Jun 12 2021

- Resolve namedtuple python 3.5 incompatibility (gorarakelyan)
- Add ability to align X-axis by a metric (mihran113, roubkar)
- Add tooltip popover for the chart hover state (roubkar)

32.43 2.5.0 May 27 2021

- Set gunicorn timeouts (mihran113)
- Remove redundant deserialize method (gorarakelyan)
- Move the Flask server to main repo to support 'docker'less UI (mihran113)

32.44 2.4.0 May 13 2021

- Bump up Aim UI to v1.6.0 (gorarakelyan)
- Add xgboost integration (khazhak)
- Update keras adapter interface (khazhak)
- Convert tensors to python numbers (gorarakelyan)

32.45 2.3.0 Apr 10 2021

- Bump up Aim UI to v1.5.0 (gorarakelyan)
- Set default interval of sys tracking to 10 seconds (gorarakelyan)
- Add ability to track system metrics (gorarakelyan)

32.46 2.2.1 Mar 31 2021

- Bump up Aim UI to v1.4.1 (gorarakelyan)

32.47 2.2.0 Mar 24 2021

- Bump up Aim UI to v1.4.0 (gorarakelyan)
- Add Hugging Face integration (Khazhak)
- Reorganize documentation (Tatevv)

32.48 2.1.6 Feb 26 2021

- Add ability to opt out telemetry (gorarakelyan)
- Remove experiment name from config file when calling repo.remove_branch method (gorarakelyan)

32.49 2.1.5 Jan 7 2021

- Handle NaN or infinite floats passed to artifacts (gorarakelyan)

32.50 2.1.4 Dec 2 2020

- Add ability to specify session run hash (gorarakelyan)
- Initialize repo if it was empty when opening session (gorarakelyan)
- Add validation of map artifact parameters (gorarakelyan)

32.51 2.1.3 Nov 24 2020

- Support comparison of list type contexts (gorarakelyan)

32.52 2.1.2 Nov 24 2020

- Fix empty contexts comparison issue (gorarakelyan)

32.53 2.1.1 Nov 22 2020

- Return only selected params in SelectResult (gorarakelyan)

32.54 2.1.0 Nov 19 2020

- Add AimRepo select method (gorarakelyan)
- Implement SelectResult class (gorarakelyan)

32.55 2.0.27 Nov 13 2020

- Fix issue with artifact step initializer (gorarakelyan)

32.56 2.0.26 Nov 10 2020

- Add `block_termination` argument to aim.Session (gorarakelyan)
- Convert infinity parameter to string in artifacts (gorarakelyan)

32.57 2.0.25 Nov 9 2020

- Reconstruct run metadata file when running close command (gorarakelyan)

32.58 2.0.24 Nov 8 2020

- Add SIGTERM signal handler (gorarakelyan)
- Run `track` function in a parallel thread (gorarakelyan)
- Add SDK session flush method (gorarakelyan)
- Flush aggregated metrics at a given frequency (gorarakelyan)
- Update run metadata file only on artifacts update (gorarakelyan)

32.59 2.0.23 Nov 5 2020

- Make experiment name argument required in SDK close command (gorarakelyan)

32.60 2.0.22 Nov 5 2020

- Add SDK `close` method to close dangling experiments (gorarakelyan)

32.61 2.0.21 Nov 1 2020

- Resolve compatibility issues with python 3.5.0 (gorarakelyan)

32.62 2.0.20 Oct 26 2020

- Enable pypi aim package name (gorarakelyan)

32.63 2.0.19 Oct 25 2020

- Add PyTorch Lightning logger (gorarakelyan)
- Add TensorFlow v1 and v2 keras callbacks support (gorarakelyan)

32.64 2.0.18 Oct 7 2020

- Add ability to run Aim UI in detached mode (gorarakelyan)
- Add ability to specify repo path when running Aim UI (gorarakelyan)

32.65 2.0.17 Oct 5 2020

- Rename `AimDE` to `Aim UI` (gorarakelyan)

32.66 2.0.16 Oct 2 2020

- Add ability to specify host when running AimDE (gorarakelyan)
- Disable `AimContainerCommandManager` (gorarakelyan)
- Remove `aimde` command entry point (gorarakelyan)
- Remove `de` prefix from development environment management commands (gorarakelyan)

32.67 2.0.15 Sep 21 2020

- Set Map artifact default namespace (gorarakelyan)

32.68 2.0.14 Sep 21 2020

- Set Metric hashable context to None if no kwarg is passed (gorarakelyan)

32.69 2.0.13 Sep 21 2020

- Add ability to query runs by metric value (gorarakelyan)
- Add ability to query runs via SDK (gorarakelyan)

32.70 2.0.12 Sep 12 2020

- Update Session to handle exceptions gracefully (gorarakelyan)

32.71 2.0.11 Sep 11 2020

- Add alias to keras adapter (gorarakelyan)

32.72 2.0.10 Sep 10 2020

- Show progress bar when pulling AimDE image (gorarakelyan)

32.73 2.0.9 Sep 10 2020

- Add ability to start multiple sessions (gorarakelyan)
- Add Aim adapter for keras (gorarakelyan)

32.74 2.0.8 Aug 26 2020

- Set SDK to select only unarchived runs by default (gorarakelyan)
- Add ability to archive/unarchive runs (gorarakelyan)
- Enable search by run attributes (gorarakelyan)
- Add `is not` keyword to AimQL (gorarakelyan)

32.75 2.0.7 Aug 21 2020

- Validate Artifact values before storing (gorarakelyan)
- Add sessions to SDK (gorarakelyan)

32.76 2.0.6 Aug 13 2020

- Add ability to retrieve metrics and traces from repo (gorarakelyan)
- Add SDK `select` method to select runs and artifacts (gorarakelyan)
- Implement search query language (gorarakelyan)

32.77 2.0.5 Jul 18 2020

- Fix issue with PyPI reStructuredText format compatibility (gorarakelyan)

32.78 2.0.4 Jul 18 2020

- Add ability to attach tf.summary logs to AimDE (gorarakelyan)

32.79 2.0.3 Jul 8 2020

- Pass project path to development environment container (gorarakelyan)

32.80 2.0.2 Jul 7 2020

- Make `epoch` argument optional for `Metric` artifact (gorarakelyan)
- Add ability to automatically commit runs after exit (gorarakelyan)
- Add `aim up` shortcut for running development environment (gorarakelyan)
- Remove first required argument(artifact name) from sdk track function (gorarakelyan)
- Add general dictionary artifact for tracking `key: value` parameters (gorarakelyan)

32.81 2.0.1 Jun 24 2020

- Fix inconsistent DE naming (gorarakelyan)

32.82 2.0.0 Jun 18 2020

- Tidy up aim and remove some artifacts (gorarakelyan)
- Update AimContainerCMD to open connection on custom port (gorarakelyan)
- Save passed process uuid to commit configs (gorarakelyan)
- Ability to query processes (gorarakelyan)
- Execute process and store logs into a commit of specific experiment (gorarakelyan)
- Kill running process and its children recursively (gorarakelyan)
- Keep executed processes for monitoring and management (gorarakelyan)
- Add container command handler to exec commands on the host (gorarakelyan)
- Refactor Text artifact to store sentences using protobuf and aimrecords (jamesj-jiao)
- Add ability to pass aim board port as an argument (gorarakelyan)

32.83 1.2.17 May 8 2020

- Add config command (gorarakelyan)
- Tune artifacts: images, metric_groups, params (gorarakelyan)

32.84 1.2.16 Apr 29 2020

- Add ability to pass numpy array as a segmentation mask (gorarakelyan)

32.85 1.2.15 Apr 29 2020

- Add basic image list tracking (gorarakelyan)

32.86 1.2.14 Apr 27 2020

- Optimize segmentation tracking insight to load faster (gorarakelyan)

32.87 1.2.13 Apr 25 2020

- Remove GitHub security alert (gorarakelyan)
- Add image semantic segmentation tracking (gorarakelyan)

32.88 1.2.12 Apr 20 2020

- Add missing init file for aim.artifacts.proto (@mike1808)

32.89 1.2.11 Apr 16 2020

- Make epoch property optional for Metric (gorarakelyan)

32.90 1.2.10 Apr 16 2020

- Serialize and store `Metric` records using protobuf and aimrecords (gorarakelyan)
- Create RecordWriter factory which handles artifact records saving (gorarakelyan)
- Extract artifact serialization to ArtifactWriter (mike1808)

32.91 1.2.9 Mar 16 2020

- Alert prerequisites installation message for running board (gorarakelyan)

32.92 1.2.8 Mar 15 2020

- Update profiler interface for keras (gorarakelyan)

32.93 1.2.7 Mar 14 2020

- Add board pull command (gorarakelyan)
- Change board ports to 43800,1,2 (gorarakelyan)
- Add ability to profile graph output nodes (gorarakelyan)
- Remove issue with autograd inside while loop (gorarakelyan)
- Add aim board development mode (gorarakelyan)
- Update board name hash algorithm to md5 (gorarakelyan)
- Add board CLI commands: up, down and upgrade (gorarakelyan)
- Add ability to tag version as a release candidate (gorarakelyan)

32.94 1.2.6 Feb 28 2020

- Add learning rate update tracking (gorarakelyan)

32.95 1.2.5 Feb 25 2020

- Add autocommit feature to push command: `aim push -c [-m <msg>]` (gorarakelyan)
- Add cli status command to list branch uncommitted artifacts (gorarakelyan)
- Add an ability to aggregate duplicated nodes within a loop (gorarakelyan)
- Remove gradient break issue when profiling output nodes (gorarakelyan)

32.96 1.2.4 Feb 20 2020

- Enable profiler to track nodes inside loops (gorarakelyan)
- Ability to disable profiler for evaluation or inference (gorarakelyan)

32.97 1.2.3 Feb 13 2020

- Set minimum required python version to 3.5.2 (gorarakelyan)

32.98 1.2.2 Feb 13 2020

- Downgrade required python version (gorarakelyan)

32.99 1.2.1 Feb 13 2020

- Edit README.md to pass reStructuredText validation on pypi (gorarakelyan)

32.100 1.2.0 Feb 13 2020

- Make aim CLI directly accessible from main.py (gorarakelyan)
- Add disk space usage tracking (gorarakelyan)
- Add profiler support for Keras (gorarakelyan)
- Add TensorFlow graph nodes profiler (gorarakelyan)
- Add command to run aim live container mounted on aim repo (gorarakelyan)
- Update profiler to track GPU usage (gorarakelyan)
- Add machine resource usage profiler (gorarakelyan)

32.101 1.1.1 Jan 14 2020

- Remove aim dependencies such as keras, pytorch and etc (gorarakelyan)

32.102 1.1.0 Jan 12 2020

- Update code diff tracking to be optional (gorarakelyan)
- Add default False value to aim init function (gorarakelyan)
- Update aim repo to correctly identify cwd (gorarakelyan)
- Update push command to commit if msg argument is specified (gorarakelyan)
- Add ability to initialize repo from within the sdk (gorarakelyan)

32.103 1.0.2 Jan 7 2020

- Remove objects dir from empty .aim branch index (gorarakelyan)

32.104 1.0.1 Dec 26 2019

- Add cil command to print aim current version (gorarakelyan)

32.105 1.0.0 Dec 25 2019

- Add aim version number in commit config file (gorarakelyan)
- Update push command to send username and check storage availability (gorarakelyan)
- Add hyper parameters tracking (gorarakelyan)
- Update push command to print shorter file names when pushing to remote (gorarakelyan)
- Update tracking artifacts to be saved in log format (gorarakelyan)
- Add pytorch cuda support to existing sdk artefacts (gorarakelyan)
- Add cli reset command (gorarakelyan)
- Add nested module tracking support to aim sdk (gorarakelyan)
- Add code difference tracking to aim sdk (gorarakelyan)
- Update aim push command to send commits (gorarakelyan)
- Add commit structure implementation (gorarakelyan)
- Add aim commit command synchronized with git commits (gorarakelyan)
- Add version control system factory (gorarakelyan)
- Update all insights example (gorarakelyan)
- Add model gradients tracking (gorarakelyan)

- Add model weights distribution tracking (gorarakelyan)
- Add aim correlation tracking (gorarakelyan)

32.106 0.2.9 Nov 30 2019

- Update push tolerance when remote origin is invalid (gorarakelyan)

32.107 0.2.8 Nov 30 2019

- Update aim auth public key search algorithm (gorarakelyan)

32.108 0.2.7 Nov 14 2019

- Update dependencies torch and torchvision versions (sgevorg)

32.109 0.2.6 Nov 5 2019

- Update aim track logger (gorarakelyan)

32.110 0.2.5 Nov 4 2019

- Add branch name validation (gorarakelyan)
- Add single branch push to aim push command (gorarakelyan)

32.111 0.2.4 Nov 3 2019

- Update aim auth print format (gorarakelyan)
- Update setup.py requirements (gorarakelyan)

32.112 0.2.3 Nov 3 2019

- Update package requirements (gorarakelyan)

32.113 0.2.2 Nov 1 2019

- Update package requirements (sgevorg)

32.114 0.2.1 Nov 1 2019

- Add paramiko to required in setup.py (sgevorg)

32.115 0.2.0 Nov 1 2019

- Update the repo to prep for open source pypi push (sgevorg)
- Add error and activity logging (sgevorg)
- Add push command robustness (gorarakelyan)
- Add cli auth command (gorarakelyan)
- Add public key authentication (gorarakelyan)
- Update push to send only branches (gorarakelyan)
- Add branching command line interface (gorarakelyan)
- Update skd interface (gorarakelyan)
- Add pytorch examples inside examples directory (gorarakelyan)
- Add model load sdk method (gorarakelyan)
- Add model checkpoint save tests (gorarakelyan)
- Update file sending protocol (gorarakelyan)
- Add model tracking (gorarakelyan)

32.116 0.1.0 - Sep 23 2019

- Update setup py to build cython extensions (gorarakelyan)
- Update tcp client to send multiple files through one connection (gorarakelyan)
- Update tcp client to send images (gorarakelyan)
- Update sdk track functionality to support multiple metrics (gorarakelyan)
- Update push command for sending repo to a given remote (gorarakelyan)
- Add cli remote commands (gorarakelyan)
- Update cli architecture from single group of commands to multiple groups (gorarakelyan)
- Add testing env first skeleton and versions (sgevorg)
- Add dummy exporting files from .aim-test (sgevorg)
- Add description for Testing Environment (sgevorg)
- Update metadata structure and handling (sgevorg)

- Add support for seq2seq models (sgevorg)
- Update the output of doker image build to be more informative and intuitive (sgevorg)
- Update README.MD with changed Aim messaging (sgevorg)
- Remove setup.cfg file (maybe temporarily) (sgevorg)
- Update the location for docker build template files, move to data/ (sgevorg)
- Update the `docs/cli.md` for aim-deploy docs (sgevorg)
- Add docker deploy `.aim/deploy_temp/<model>` cleanup at the end of the build (sgevorg)
- Add Docker Deploy via `aim-deploy` command (sgevorg)
- Add Docker image generate skeleton (sgevorg)
- Add AimModel.load_mode static function to parse `.aim` files (sgevorg)
- Update exporter to decouple from specifics of exporting and framework (sgevorg)
- Add model export with `.aim` extension (sgevorg)
- Remove pack/unpack of the metadata (sgevorg)
- Add pack/unpack to add metadata to model for engine processing (sgevorg)
- Add aim-deploy command configuration in cli (sgevorg)
- Add basic cli (sgevorg)
- Update setup.py for cli first version (sgevorg)
- Add initial cli specs (sgevorg)
- Add directories: the initial skeleton of the repo (sgevorg)
- Add gitignore, license file and other basics for repo (sgevorg)

CHAPTER

THIRTYTHREE

INDICES AND TABLES

- genindex
- modindex