Aim

Release 3.3.1

Gev Sogomonian, Gor Arakelyan et al.

Dec 18, 2021

CONTENTS

CHAPTER

ONE

WHY USE AIM?

- Modern ML development revolves around collection and analysis of AI metadata (training metrics, images, distributions etc) to analyze and explore different aspects of the model performance.
- There is both a need to manually explore and compare the metadata as well as automate for different infrastructure needs.
- Aim helps to track AI metadata and
  - Explore it manually through the most advanced open-source experiment comparison web UI.
  - Query programmatically in your favorite notebook or through script for automation.
- Use Aim to seamlessly log your ML metadata in your training environment and explore through UI and code. **Aim is free, open-source and self-hosted.**

CHAPTER

TWO

WHAT CAN YOU DO WITH AIM?

2.1 Log metrics and params

Use the Aim SDK to log as many metrics and params as you need for your training and evaluation runs. Aim users track 1000s of training runs and sometimes more than 100s of metrics per run with lots of steps.

2.2 Query metadata on Web UI

Aim enables a powerful pythonic query language to filter through metadata. It's like a python if statement over everything you have tracked. You can use this on all explorer screens.

2.3 Runs explorer

Runs explorer will help you to hollistically view all your runs, each metric last tracked values and tracked hyperparameters.

2.4 Metrics explorer

Metrics explorer helps you to compare 100s of metrics within a few clicks. It helps to save lots of time compared to other open-source experiment tracking tools.

2.5 Images explorer

Track intermediate images and search, compare them on the Images Explorer.

2.6 Params explorer

Params explorer enables a parallel coordinates view for metrics and params. Very helpful when doing hyperparameter search.

2.7 Query metadata programmatically

Use the same pythonic if statement to query the data through the Aim SDK programmatically.

CHAPTER

THREE

HOW AIM WORKS?

Aim is a python package with three main components:

- Aim Storage:
  - A rocksdb-based embedded storage where the metadata is stored locally
- Aim SDK:
  - A simple python interface that allows to track AI metadata
    - metrics
    - hyperparameters
    - images
    - distributions
- Aim UI:
  - A self-hosted web interface to deeply explore the tracked metadata

Integrated with your favorite tools

CHAPTER

FOUR

COMPARISONS TO FAMILIAR TOOLS

4.1 Tensorboard

Training run comparison

Order of magnitude faster training run comparison with Aim

- The tracked params are first class citizens at Aim. You can search, group, aggregate via params - deeply explore all the tracked data (metrics, params, images) on the UI.
- With tensorboard the users are forced to record those parameters in the training run name to be able to search and compare. This causes a super-tedius comparison experience and usability issues on the UI when there are many experiments and params. TensorBoard doesn't have features to group, aggregate the metrics.

Scalability

- Aim is built to handle 1000s of training runs with dozens of experiments each - both on the backend and on the UI.
- TensorBoard becomes really slow and hard to use when a few hundred training runs are queried / compared.

Beloved TB visualizations to be added on Aim

- Distributions / gradients visualizations.
- Embedding projector.
- Neural network visualization.

4.2 MLFlow

MLFlow is an end-to-end ML Lifecycle tool. Aim is focused on training tracking. The main differences of Aim and MLflow are around the UI scalability and run comparison features.

Run comparison

- Aim treats tracked parameters as first-class citizens. Users can query runs, metrics, images and filter using the params.
- MLFlow does have a search by tracked config, but there are no grouping, aggregation, subplotting by hyparparams and other comparison features available.

UI Scalability

- Aim UI can handle several thousands of metrics at the same time smoothly with 1000s of steps. It may get shaky when you explore 1000s of metrics with 10000s of steps each. But we are constantly optimizing!

- MLflow UI becomes slow to use when there are a few hundreds of runs.

4.3 Weights and Biases

Hosted vs self-hosted

- Weights and Biases is a hosted closed-source experiment tracker.
- Aim is self-hosted free and open-source.
  - Remote self-hosted Aim is coming soon…

CHAPTER

FIVE

COMMUNITY

If you have questions please:

1. Open a feature request or report a bug
2. Join our slack

5.1 Overview

5.1.1 Why use Aim?

- Modern ML development revolves around collection and analysis of AI metadata (training metrics, images, distributions etc) to analyze and explore different aspects of the model performance.
- There is both a need to manually explore and compare the metadata as well as automate for different infrastructure needs.
- Aim helps to track AI metadata and
  - Explore it manually through the most advanced open-source experiment comparison web UI.
  - Query programmatically in your favorite notebook or through script for automation.
- Use Aim to seamlessly log your ML metadata in your training environment and explore through UI and code. **Aim is free, open-source and self-hosted.**

5.1.2 What can you do with Aim?

Log metrics and params

Use the Aim SDK to log as many metrics and params as you need for your training and evaluation runs. Aim users track 1000s of training runs and sometimes more than 100s of metrics per run with lots of steps.

Query metadata on Web UI

Aim enables a powerful pythonic query language to filter through metadata. It's like a python if statement over everything you have tracked. You can use this on all explorer screens.

Runs explorer

Runs explorer will help you to hollistically view all your runs, each metric last tracked values and tracked hyperparameters.

Metrics explorer

Metrics explorer helps you to compare 100s of metrics within a few clicks. It helps to save lots of time compared to other open-source experiment tracking tools.

Images explorer

Track intermediate images and search, compare them on the Images Explorer.

Params explorer

Params explorer enables a parallel coordinates view for metrics and params. Very helpful when doing hyperparameter search.

Query metadata programmatically

Use the same pythonic if statement to query the data through the Aim SDK programmatically.

5.1.3 How Aim works?

Aim is a python package with three main components:

- Aim Storage:
  - A rocksdb-based embedded storage where the metadata is stored locally
- Aim SDK:
  - A simple python interface that allows to track AI metadata
    - metrics
    - hyperparameters
    - images
    - distributions

- Aim UI:
  - A self-hosted web interface to deeply explore the tracked metadata

Integrated with your favorite tools

5.1.4 Comparisons to familiar tools

Tensorboard

Training run comparison

Order of magnitude faster training run comparison with Aim

- The tracked params are first class citizens at Aim. You can search, group, aggregate via params - deeply explore all the tracked data (metrics, params, images) on the UI.
- With tensorboard the users are forced to record those parameters in the training run name to be able to search and compare. This causes a super-tedius comparison experience and usability issues on the UI when there are many experiments and params. TensorBoard doesn't have features to group, aggregate the metrics.

Scalability

- Aim is built to handle 1000s of training runs with dozens of experiments each - both on the backend and on the UI.
- TensorBoard becomes really slow and hard to use when a few hundred training runs are queried / compared.

Beloved TB visualizations to be added on Aim

- Distributions / gradients visualizations.
- Embedding projector.
- Neural network visualization.

MLFlow

MLFlow is an end-to-end ML Lifecycle tool. Aim is focused on training tracking. The main differences of Aim and MLflow are around the UI scalability and run comparison features.

Run comparison

- Aim treats tracked parameters as first-class citizens. Users can query runs, metrics, images and filter using the params.
- MLFlow does have a search by tracked config, but there are no grouping, aggregation, subplotting by hyparparams and other comparison features available.

UI Scalability

- Aim UI can handle several thousands of metrics at the same time smoothly with 1000s of steps. It may get shaky when you explore 1000s of metrics with 10000s of steps each. But we are constantly optimizing!
- MLflow UI becomes slow to use when there are a few hundreds of runs.

Weights and Biases

Hosted vs self-hosted

- Weights and Biases is a hosted closed-source experiment tracker.
- Aim is self-hosted free and open-source.
  - Remote self-hosted Aim is coming soon...

5.1.5 Community

If you have questions please:

1. Open a feature request or report a bug
2. Join our slack

5.2 Getting started

You only need a few steps to get started with Aim.

5.2.1 Installation

Install Aim via pip3:

```
pip3 install aim
```

Note: You need to have python3 and pip3 installed in your environment before installing Aim.

5.2.2 Integrate with your code

1. Create Run stored in the current directory:

```
from aim import Run

run = Run()
```

1. Log parameters:

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

1. Track metrics:

```
for i in range(10):
    run.track(i, name='loss', step=i, context={ "subset":"train" })
    run.track(i, name='acc', step=i, context={ "subset":"train" })
```

More details/examples here.

Congrats! Your first run is ready!

5.2.3 Run Aim UI

Start up the Aim UI to observe the run:

```
aim up
```

See more details in UI basics.

5.2.4 Query metadata via SDK

```
from aim import Repo

# Read .aim repo located at the current working directory
repo = Repo('.')

# Get collection of metrics
for run_metrics_collection in repo.query_metrics("metric.name == 'loss'").iter_runs():
    for metric in run_metrics_collection:
        # Get run params
        params = metric.run[...]
        # Get metric values
        steps, metric_values = metric.values.sparse_numpy()
```

See more details in SDK basics.

5.3 SDK Basics

5.3.1 Create a Run

`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. Tracked data is stored in `.aim` repo. If the path is not specified, the data is stored in the current working directory.

Use `Run` arguments to:

- Define where to store the data
- Define experiment name to group related runs together
- Enable system resource usage tracking (CPU, GPU, memory, etc..)

```
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
- Use integrations to automate tracking

5.3.2 Continue a Run

Specify the run hash when initializing a Run object to continue tracking.

```
from aim import Run

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

5.3.3 Track params and metrics with Run

Run provides simple and intuitive interface for:

- Tracking the metrics of your training run
- Logging the parameters of your training run

Parameters

Track nearly any python dictionaries:

```
# Log training hyper-parameters
my_run['hparams'] = {
    'learning_rate': 0.0001,
    'batch_size': 32,
}
```

Supported types of dictionaries.

Metrics

Use `track` method to log ML metrics like 'loss', 'accuracy' or 'bleu'.

```
# Track metrics
for step in range(1000):
    value = step * 10
    my_run.track(
        value,       # Current value to track
        name='loss', # The metric name
        step=step,   # Step index (optional)
        epoch=0,     # Epoch (optional)
        context={    # Metric context (optional)
            'subset': 'train',
        },
    )
```

Run.track method full spec.

5.3.4 Track images with Run

Track images to explore model inputs, outputs, confusion matrices, weights, etc:

```
from aim import Image

for step in range(1000):
    my_run.track(
        Image(img_tensor_or_pil, img_caption), # Pass image data and/or caption
        name='generated', # The name of image set
        step=step,   # Step index (optional)
        epoch=0,     # Epoch (optional)
        context={    # Context (optional)
            'subset': 'train',
        },
    )
```

`Image` class full spec.

Tracking batches of images:

```
for step, (images, labels) in enumerate(train_loader):
    aim_images = [Image(img, lbl) for img, lbl in zip(images, labels)]

    my_run.track(
        aim_images, # List of images
        name='generated', # The name of image set
        step=step,   # Step index (optional)
        epoch=0,     # Epoch (optional)
        context={    # Context (optional)
            'subset': 'train',
        },
    )
```

Full example here.

5.3.5 Track distributions with Run

Track distributions to explore model gradients, weights, etc. To store a distribution pass an iterable of scalar values to the `Distribution` object.

```
from aim import Distribution

for step in range(1000):
    my_run.track(
        Distribution(tensor), # Pass distribution
        name='gradients', # The name of distributions
        step=step,   # Step index (optional)
        epoch=0,     # Epoch (optional)
        context={    # Context (optional)
            'type': 'weights',
        },
    )
```

`Distribution` class full spec.

5.3.6 Query Runs and saved metadata

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()
```

See more advanced usage examples here.

5.4 Search Basics

5.4.1 Introduction

Aim enables a powerful query language(AimQL) 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 *security restrictions* in place.

The data is saved as diff 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.

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

5.4.2 Searching runs

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):
   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.

Note:

The two following examples are equal:

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

Warning: 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>	

5.4.3 Searching metrics and images

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>

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

5.4.4 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.

5.5 UI Basics

Aim enables powerful UI to explore logged ML runs and metadata.

5.5.1 Runs explorer

Runs explorer will help you to hollistically view all your runs, each metric last tracked values and tracked hyperparameters.

Features:

- Full Research context at hand
- Search runs by date, experiment, hash, tag or parameters
- Search by run/experiment

5.5.2 Metrics explorer

Metrics explorer helps you to compare 100s of metrics within a few clicks. It helps to save lots of time compared to other open-source experiment tracking tools.

Features:

- Easily query any metric
- Group by any parameter
- Divide into subplots
- Aggregate grouped metrics (by conf. interval, std. dev., std. err., min/max)
- Apply smoothing
- Change scale of the axes (linear or log)
- Align metrics by time, epoch or another metric

5.5.3 Images explorer

Track intermediate images and search, compare them on the Images Explorer.

Features:

- Easily query any image
- Group by images by run parameters
- Group images by step

5.5.4 Params explorer

Params explorer enables a parallel coordinates view for metrics and params. Very helpful when doing hyperparameter search.

Features:

- Easily query any metrics and params
- Group runs or divide into subplots
- Apply chart indicator to see correlations

5.5.5 Single run page

Explore all the metadata associated with a run on the single run page. It's accessible from all the tables and tooltips.

Features:

- See all the logged params of a run
- See all the tracked metrics(including system metrics)

5.6 Integrations

Easily integrate Aim with your favorite framework / tool

5.6.1 Python script

```
import aim

# Save inputs, hparams or any other `key: value` pairs
aim.set_params(hyperparam_dict, name='hparams') # Passing name argument is optional

# ...
for step in range(10):
    # Log metrics to visualize performance
```

(continues on next page)

(continued from previous page)

```
    aim.track(metric_value, name='metric_name', epoch=epoch_number)
# ...
```

5.6.2 Hugging Face

```
from aim.hugging_face import AimCallback

# ...
aim_callback = AimCallback(repo='/path/to/logs/dir', experiment='mnli')
trainer = Trainer(
    model=model,
    args=training_args,
    train_dataset=train_dataset if training_args.do_train else None,
    eval_dataset=eval_dataset if training_args.do_eval else None,
    callbacks=[aim_callback],
    # ...
)
# ...
```

5.6.3 Pytorch Lightning

```
from aim.pytorch_lightning import AimLogger

# ...
trainer = pl.Trainer(logger=AimLogger(experiment='experiment_name'))
# ...
```

5.6.4 Keras & tf.keras

```
import aim

# ...
model.fit(x_train, y_train, epochs=epochs, callbacks=[
    aim.keras.AimCallback(repo='/path/to/logs/dir', experiment='experiment_name')

    # Use aim.tensorflow.AimCallback in case of tf.keras
    aim.tensorflow.AimCallback(repo='/path/to/logs/dir', experiment='experiment_name')
])
# ...
```

5.6.5 XGBoost

```
from aim.xgboost import AimCallback

# ...
aim_callback = AimCallback(repo='/path/to/logs/dir', experiment='experiment_name')
bst = xgb.train(param, xg_train, num_round, watchlist, callbacks=[aim_callback])
# ...
```

5.6.6 Jupyter Notebook

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.

5.7 Aim SDK

5.7.1 aim.sdk.repo module

class aim.sdk.repo.**Repo**(*path*, *, *read_only=None*, *init=False*)

Aim repository object.

Provides methods for repositories creation/opening/cleanup. Provides APIs for accessing Runs. Provides API for querying Runs/Metrics based on a given expression.

Parameters

- **path** (*str*) – Path to Aim repository.
- **read_only** (bool, optional) – Flag for opening Repo in readonly mode. False by default.
- **init** (bool, optional) – Flag used to initialize new Repo. False by default. Recommended to use `aim init` command instead.

collect_params_info()

Utility function for getting run meta-parameters.

Returns All runs meta-parameters.

Return type dict

collect_sequence_info(*sequence_types*)

Utility function for getting sequence names and contexts for all runs by given sequence types.

Parameters

- **sequence_types** (tuple[str], optional) – Sequence types to get tracked sequence names/contexts for.

- **'metric'.** (*Defaults to*) –

Returns Tree of sequences and their contexts groupped by sequence type.

Return type dict

classmethod default_repo(*init=False*)
Named constructor for default repository.

Searches nearest *.aim* directory from current directory to roo directory. If not found, return Repo for current directory.

Parameters **init** (bool, optional) – Flag used to initialize new Repo. False by default. Recommended to use *aim init* command instead.

Returns *Repo* object.

classmethod exists(*path*)
Check Aim repository existence.

Parameters **path** (*str*) – Path to Aim repository.

Returns True if repository exists, False otherwise.

classmethod from_path(*path*, *read_only=None*, *init=False*)
Named constructor for Repo for given path.

Parameters

- **path** (*str*) – Path to Aim repository.
- **read_only** (bool, optional) – Flag for opening Repo in readonly mode. False by default.
- **init** (bool, optional) – Flag used to initialize new Repo. False by default. Recommended to use `aim init` command instead.

Returns *Repo* object.

get_run(*run_hash*)
Get run if exists.

Parameters **run_hash** (*str*) – Run hash.

Returns Run object if hash is found in repository. *None* otherwise.

iter_runs()
Iterate over Repo runs.

Yields next Run in readonly mode .

query_audios(*query=''*)
Get audio collections satisfying query expression.

Parameters **query** (*str*) – query expression.

Returns Iterable for audio sequences matching query expression.

Return type SequenceCollection

query_distributions(*query=''*)
Get distribution collections satisfying query expression.

Parameters **query** (*str*) – query expression.

Returns Iterable for distribution sequences matching query expression.

Return type SequenceCollection

query_figure_objects(*query=''*)
Get Figures collections satisfying query expression.

Parameters **query** (*str*) – query expression.

Returns Iterable for Figure sequences matching query expression.

Return type SequenceCollection

query_images(*query=''*)
Get image collections satisfying query expression.

Parameters **query** (*str*) – query expression.

Returns Iterable for image sequences matching query expression.

Return type SequenceCollection

query_metrics(*query=''*)
Get metrics satisfying query expression.

Parameters **query** (*str*) – query expression.

Returns Iterable for metrics matching query expression.

Return type MetricCollection

query_runs(*query=''*, *paginated=False*, *offset=None*)
Get runs satisfying query expression.

Parameters

- **query** (str, optional) – query expression. If not specified, query results will include all runs.
- **paginated** (bool, optional) – query results pagination flag. False if not specified.
- **offset** (str, optional) – *hash* of Run to skip to.

Returns Iterable for runs/metrics matching query expression.

Return type SequenceCollection

query_texts(*query=''*)
Get text collections satisfying query expression.

Parameters **query** (*str*) – query expression.

Returns Iterable for text sequences matching query expression.

Return type SequenceCollection

classmethod rm(*path*)
Remove Aim repository.

Parameters **path** (*str*) – Path to Aim repository.

5.7.2 aim.sdk.run module

class aim.sdk.run.**Run**(*run_hash=None*, ***, *repo=None*, *read_only=False*, *experiment=None*, *system_tracking_interval=10*)

Run object used for tracking metrics.

Provides method *track* to track value and object series for multiple names and contexts. Provides dictionary-like interface for Run object meta-parameters. Provides API for iterating through tracked sequences.

Parameters

- **run_hash** (str, optional) – Run's hash. If skipped, generated automatically.
- **(** (*repo*) – obj:`Union[Repo,str], optional): Aim repository path or Repo object to which Run object is bound. If skipped, default Repo is used.
- **read_only** (bool, optional) – Run creation mode. Default is False, meaning Run object can be used to track metrics.
- **experiment** (str, optional) – Sets Run's *experiment* property. 'default' if not specified. Can be used later to query runs/sequences.
- **system_tracking_interval** (int, optional) – Sets the tracking interval in seconds for system usage metrics (CPU, Memory, etc.). Set to *None* to disable system metrics tracking.

__delitem__(*key*)

Remove key from run meta-params. :param key: meta-parameter path

__getitem__(*key*)

Get run meta-parameter by key.

Parameters **key** – path to Run meta-parameter.

Returns Collected sub-tree of Run meta-parameters.

Examples

```
>>> run = Run('3df703c')
>>> run['hparams']  # -> {'batch_size': 42}
>>> run['hparams', 'batch_size']  # -> 42
```

__setitem__(*key*, *val*)

Set Run top-level meta-parameter.

Parameters

- **key** (str) – Top-level meta-parameter name. Use ellipsis to reset run's all meta-parameters.
- **val** – Meta-parameter value.

Examples

```
>>> run = Run('3df703c')
>>> run[...] = params
>>> run['hparams'] = {'batch_size': 42}
```

add_tag(*value*)
Add tag to run

Parameters **value** (*str*) – Tag to add.

collect_sequence_info(*sequence_types*, *skip_last_value=False*)
Retrieve Run's all sequences general overview.

Parameters

- **sequence_types** – Type names of sequences for which to collect name/context pairs.
- **skip_last_value** (bool, optional) – Boolean flag to include tracked sequence last value in
- **default.** (*sequence info. False by*) –

Returns list of sequence's *context*, *name* and optionally last tracked value triplets.

Return type list

dataframe(*include_props=True*, *include_params=True*)
Get run properties and params as pandas DataFrame

Parameters

- **include_props** – (int, optional): If true, include run structured props
- **include_params** – (int, optional): If true, include run parameters

get_audio_sequence(*name*, *context*)
Retrieve audios sequence by its name and context.

Parameters

- **name** (*str*) – Tracked audios sequence name.
- **context** (Context) – Tracking context.

Returns Audios object if exists, *None* otherwise.

get_distribution_sequence(*name*, *context*)
Retrieve distributions sequence by it's name and context.

Parameters

- **name** (*str*) – Tracked distribution sequence name.
- **context** (Context) – Tracking context.

Returns Distributions object if exists, *None* otherwise.

get_figure_sequence(*name*, *context*)
Retrieve figure sequence by its name and context.

Parameters

- **name** (*str*) – Tracked figure sequence name.
- **context** (Context) – Tracking context.

Returns `Figures` object if exists, *None* otherwise.

`get_image_sequence`(*name*, *context*)
Retrieve images sequence by it's name and context.

Parameters

- **name** (*str*) – Tracked image sequence name.
- **context** (`Context`) – Tracking context.

Returns `Images` object if exists, *None* otherwise.

`get_metric`(*name*, *context*)
Retrieve metric sequence by it's name and context.

Parameters

- **name** (*str*) – Tracked metric name.
- **context** (`Context`) – Tracking context.

Returns `Metric` object if exists, *None* otherwise.

`get_text_sequence`(*name*, *context*)
Retrieve texts sequence by it's name and context.

Parameters

- **name** (*str*) – Tracked text sequence name.
- **context** (`Context`) – Tracking context.

Returns `Texts` object if exists, *None* otherwise.

`iter_metrics_info`()
Iterator for all run metrics info.

Yields tuples of (name, context, run) where run is the Run object itself and name, context defines Metric type sequence (with values of *float* and *int*).

`iter_sequence_info_by_type`(*dtypes*)
Iterator for run sequence infos for the given object data types

Parameters **dtypes** – The objects data types list.

Yields tuples of (name, context, run) where run is the Run object itself and name, context defines sequence for one of *dtypes* types.

`metrics`()
Get iterable object for all run tracked metrics.

Returns Iterable for run metrics.

Return type `MetricCollection`

Examples

```
>>> run = Run('3df703c')
>>> for metric in run.metrics():
>>>     metric.values.sparse_numpy()
```

remove_tag(*tag_id*)
Remove run tag.

Parameters **tag_id** (*str*) – uuid of tag to be removed.

track(*value*, *name*, *step=None*, *epoch=None*, *, *context=None*)
Main method for tracking numeric value series and object series.

Parameters

- **value** – The tracked value.
- **name** (*str*) – Tracked sequence name.
- **step** (int, optional) – Sequence tracking iteration. Auto-incremented if not specified.
- **epoch** (int, optional) – The training epoch.
- **context** (dict, optional) – Sequence tracking context.

Appends the tracked value to sequence specified by *name* and *context*. Appended values should be of the same type, in other words, sequence is a homogeneous collection.

property archived
Check is run archived or not.

Getter Returns run's archived state.

Setter Archive/un-archive run.

Type bool

property created_at
Run object creation time [UTC] as datetime.

Getter Returns run creation time.

property creation_time
Run object creation time [UTC] as timestamp.

Getter Returns run creation time.

property description
Run description, set by user.

Getter Returns run's description.

Setter Sets run's description.

Type string

property end_time
Run finalization time [UTC] as timestamp.

Getter Returns run finalization time.

property experiment
Run experiment.

Getter Returns run's experiment name.

Setter Sets run's experiment.

Type string

property finalized_at
Run finalization time [UTC] as datetime.

Getter Returns run finalization time.

property name
Run name, set by user.

Getter Returns run's name.

Setter Sets run's name.

Type string

property tags
List of run tags.

Getter Returns run's tag list.

5.7.3 aim.sdk.objects.image

class aim.sdk.objects.image.**Image**(**args*, ***kwargs*)
Image object used to store image objects in Aim repository..

Parameters

- **(** (*image*) – obj:): pillow *Image* object or *torch.Tensor* or *numpy.array* used to construct *aim.Image*.
- **caption** (str, optional) – Optional image caption. '' by default.

json()
Dump image metadata to a dict

to_pil_image()
Method to convert aim.Image to pillow Image

property caption
Image caption, set by user.

Getter Returns image caption.

Setter Sets image caption.

Type string

property format
Stored image format.

Getter Returns image format.

Type string

property height
Stored image height.

Getter Returns image height.

Type string

property size
Stored image size.

Getter Returns image (width, height) pair.

Type string

property width
Stored image width.

Getter Returns image width.

Type string

5.7.4 aim.sdk.objects.distribution

class aim.sdk.objects.distribution.**Distribution**(**args*, ***kwargs*)
Distribution object used to store distribution objects in Aim repository.

Parameters

- **(** (*distribution*) – obj:): array-like object used to construct *aim.Distribution*.
- **bin_count** (int, optional) – Optional distribution bin count. 64 by default, max 512.

json()
Dump distribution metadata to a dict

to_np_histogram()
Return *np.histogram* compatible format of the distribution

property bin_count
Stored distribution bin count

Getter Returns distribution bin_count.

Type string

property range
Stored distribution range

Getter Returns distribution range.

Type List

property ranges
Stored distribution ranges

Getter Returns distribution ranges as *np.array*.

Type np.ndarray

property weights
Stored distribution weights

Getter Returns distribution weights as *np.array*.

Type np.ndarray

5.7.5 aim.sdk.sequence module

class aim.sdk.sequence.**Sequence**(*name*, *context*, *run*)
Class representing single series of tracked value.

Objects series can be retrieved as Sequence regardless the object's type, but subclasses of Sequence might provide additional functionality. Provides interface to access tracked values, steps, timestamps and epochs. Values, epochs and timestamps are accessed via *aim.storage.arrayview.ArrayView* interface.

classmethod allowed_dtypes()
classmethod to get allowed object types for particular sequence

For example, numeric sequences a.k.a. Metric allow float and integer numbers. The base Sequence allows any value, and to indicate that, *allowed_dtypes* returns '*'.

classmethod sequence_name()
classmethod to get retrieve sequence's registered name

property epochs
Tracked epochs array as `ArrayView`.

Getter Returns epochs ArrayView.

property indices
Metric tracking steps as `list`.

Getter Returns steps list.

property timestamps
Tracked timestamps array as `ArrayView`.

Getter Returns timestamps ArrayView.

property values
Tracked values array as `ArrayView`.

Getter Returns values ArrayView.

5.7.6 aim.sdk.metric module

5.7.7 aim.sdk.image_sequence module

5.7.8 aim.sdk.distribution_sequence module

5.7.9 aim.sdk.sequence_collection module

class aim.sdk.sequence_collection.**SequenceCollection**
Abstract interface for collection of tracked series/sequences.

Typically represents sequences of a same run or sequences matching given query expression.

abstract iter()
Get Sequence iterator for collection's sequences.

Yields Next sequence object based on implementation.

abstract iter_runs()
Get SequenceCollection iterator for collection's runs.

Yields Next run's SequenceCollection based on implementation.

class aim.sdk.sequence_collection.**SingleRunSequenceCollection**(*run*, *seq_cls=<class 'aim.sdk.sequence.Sequence'>*, *query=''*)

Bases: *aim.sdk.sequence_collection.SequenceCollection*

Implementation of SequenceCollection interface for a single Run.

Method *iter()* returns Sequence iterator which yields Sequence matching query from run's sequences. Method *iter_runs()* raises StopIteration, since the collection is bound to a single Run.

Parameters

- **run** (Run) – Run object for which sequences are queried.
- **seq_cls** (type) – The collection's sequence class. Sequences not matching to seq_cls.allowed_dtypes will be skipped. *Sequence* by default, meaning all sequences will match.
- **query** (str, optional) – Query expression. If specified, method *iter()* will return iterator for sequences matching the query. If not, method *iter()* will return iterator for run's all sequences.

class aim.sdk.sequence_collection.**QuerySequenceCollection**(*repo*, *seq_cls=<class 'aim.sdk.sequence.Sequence'>*, *query=''*)

Bases: *aim.sdk.sequence_collection.SequenceCollection*

Implementation of SequenceCollection interface for repository's sequences matching given query.

Method *iter()* returns Sequence iterator, which yields Sequence matching query from currently iterated run's sequences. Once there are no sequences left in current run, repository's next run is considered. Method *iter_runs()* returns SequenceCollection iterator for repository's runs.

Parameters

- **repo** (Repo) – Aim repository object.
- **seq_cls** (type) – The collection's sequence class. Sequences not matching to seq_cls.allowed_dtypes will be skipped. *Sequence* by default, meaning all sequences will match.
- **query** (str, optional) – Query expression. If specified, method *iter()* will skip sequences not matching the query. If not, method *iter()* will return iterator for all sequences in repository (that's a lot of sequences!).

class aim.sdk.sequence_collection.**QueryRunSequenceCollection**(*repo*, *seq_cls=<class 'aim.sdk.sequence.Sequence'>*, *query=''*, *paginated=False*, *offset=None*)

Bases: *aim.sdk.sequence_collection.SequenceCollection*

Implementation of SequenceCollection interface for repository's runs matching given query.

Method *iter()* returns Sequence iterator which yields Sequence for current run's all sequences. Method *iter_runs()* returns SequenceCollection iterator from repository's runs matching given query.

Parameters

- **repo** (Repo) – Aim repository object.
- **seq_cls** (type) – The collection's sequence class. Sequences not matching to seq_cls.allowed_dtypes will be skipped. *Sequence* by default, meaning all sequences will match.

- **query** (str, optional) – Query expression. If specified, method *iter_runs()* will skip runs not matching the query. If not, method *iter_run()* will return SequenceCollection iterator for all runs in repository.

5.8 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.

5.8.1 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.

5.8.2 version

Display the Aim version installed.

```
$ aim version
```

5.8.3 up

Start the Aim web UI locally.

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

Args	Description
`-h &#124; --host <host>`	Specify host address.
`-p &#124; --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.

5.8.4 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`.

5.8.5 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.

5.9 Aim Storage

5.9.1 aim.storage.arrayview module

class aim.storage.arrayview.**ArrayView**(**args*, ***kwds*)

Array of homogeneous elements with sparse indices. Interface for working with array as a non-sparse array is available for cases when index values are not important.

first()

First index and value of the array.

first_idx()

First index of the array.

first_value()

First value of the array.

indices()

Return sparse indices iterator.

Yields Array's next sparse index.

indices_list()
Get sparse indices as a `list`.

indices_numpy()
Get sparse indices as numpy array.

items()
Return items iterator.

Yields Tuple of array's next sparse index and value.

keys()
Return sparse indices iterator.

Yields Array's next sparse index.

last()
Last index and value of the array.

last_idx()
Last index of the array.

last_value()
Last value of the array.

sparse_list()
Get sparse indices and values as :obj:`list`s.

sparse_numpy()
Get sparse indices and values as numpy arrays.

tolist()
Convert to values list

values()
Return values iterator.

Yields Array's next value.

values_list()
Get values as a `list`.

values_numpy()
Get values as numpy array.

5.10 Track and compare GANs with Aim

5.10.1 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.

5.10.2 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.

5.10.3 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)
```

(continues on next page)

(continued from previous page)

```
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

5.10.4 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:

5.10.5 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.

5.11 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 aim.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
```

5.12 Integration with Huggingface

In this guide, we will show you how to integrate Aim with Huggingface. The work we are going to do together is sentiment classification problem, which is the most common text classification task. We choose the IMDB movie review dataset as an experimental dataset, which classifies movie reviews as positive or negative. During the training process, we will show the use of aim to record effective information.

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: Huggingface 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 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]
)
```

5.13 Integration with Keras & tf.Keras

This tutorial leverages the well-known handwritten digit recognition task to describe how to integrate Aim with Keras & tf.Keras to train a digital image classification model based on the mnist dataset.

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')
])
```

5.14 Integration with Pytorch Lightning

The work is designed to build a image classifier to solve a famous real world problem ——handwritten digit recognition. In this work, we will introduce how to introduce aim logger to manage output information.

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

```
# call 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 experimential 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 to the logger variable

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

5.15 Integration with XGboost

In the real world, there is a well-known handwritten digit recognition problem. In this article, we use the machine learning framework xgboost to help us train an image classification model. In this process, we will use Aim to track our experimental data.

Enjoy using aim to track xgboost experimental data only 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, you can observe the information in real time.

5.16 Changelog

5.16.1 3.3.1 Dec 18 2021

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

5.16.2 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)

5.16.3 3.2.2 Dec 10 2021

- Fix Run finalization index timeout issue (alberttorosyan)

5.16.4 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)

5.16.5 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)

5.16.6 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)

5.16.7 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)

5.16.8 3.0.7 Nov 17 2021

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

5.16.9 3.0.6 Nov 9 2021

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

5.16.10 3.0.5 Nov 9 2021

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

5.16.11 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)

5.16.12 3.0.3 Nov 4 2021

- Bump sqlalchemy version to 1.4.1 (alberttorosyan)

5.16.13 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)

5.16.14 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)

5.16.15 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

5.16.16 2.7.1 Jun 30 2021

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

5.16.17 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)

5.16.18 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)

5.16.19 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)

5.16.20 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)

5.16.21 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)

5.16.22 2.2.1 Mar 31 2021

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

5.16.23 2.2.0 Mar 24 2021

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

5.16.24 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)

5.16.25 2.1.5 Jan 7 2021

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

5.16.26 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)

5.16.27 2.1.3 Nov 24 2020

- Support comparison of list type contexts (gorarakelyan)

5.16.28 2.1.2 Nov 24 2020

- Fix empty contexts comparison issue (gorarakelyan)

5.16.29 2.1.1 Nov 22 2020

- Return only selected params in SelectResult (gorarakelyan)

5.16.30 2.1.0 Nov 19 2020

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

5.16.31 2.0.27 Nov 13 2020

- Fix issue with artifact step initializer (gorarakelyan)

5.16.32 2.0.26 Nov 10 2020

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

5.16.33 2.0.25 Nov 9 2020

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

5.16.34 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)

5.16.35 2.0.23 Nov 5 2020

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

5.16.36 2.0.22 Nov 5 2020

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

5.16.37 2.0.21 Nov 1 2020

- Resolve compatibility issues with python 3.5.0 (gorarakelyan)

5.16.38 2.0.20 Oct 26 2020

- Enable pypi aim package name (gorarakelyan)

5.16.39 2.0.19 Oct 25 2020

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

5.16.40 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)

5.16.41 2.0.17 Oct 5 2020

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

5.16.42 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)

5.16.43 2.0.15 Sep 21 2020

- Set Map artifact default namespace (gorarakelyan)

5.16.44 2.0.14 Sep 21 2020

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

5.16.45 2.0.13 Sep 21 2020

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

5.16.46 2.0.12 Sep 12 2020

- Update Session to handle exceptions gracefully (gorarakelyan)

5.16.47 2.0.11 Sep 11 2020

- Add alias to keras adapter (gorarakelyan)

5.16.48 2.0.10 Sep 10 2020

- Show progress bar when pulling AimDE image (gorarakelyan)

5.16.49 2.0.9 Sep 10 2020

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

5.16.50 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)

5.16.51 2.0.7 Aug 21 2020

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

5.16.52 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)

5.16.53 2.0.5 Jul 18 2020

- Fix issue with PyPI reStructuredText format compatibility (gorarakelyan)

5.16.54 2.0.4 Jul 18 2020

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

5.16.55 2.0.3 Jul 8 2020

- Pass project path to development environment container (gorarakelyan)

5.16.56 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)

5.16.57 2.0.1 Jun 24 2020

- Fix inconsistent DE naming (gorarakelyan)

5.16.58 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)

5.16.59 1.2.17 May 8 2020

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

5.16.60 1.2.16 Apr 29 2020

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

5.16.61 1.2.15 Apr 29 2020

- Add basic image list tracking (gorarakelyan)

5.16.62 1.2.14 Apr 27 2020

- Optimize segmentation tracking insight to load faster (gorarakelyan)

5.16.63 1.2.13 Apr 25 2020

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

5.16.64 1.2.12 Apr 20 2020

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

5.16.65 1.2.11 Apr 16 2020

- Make epoch property optional for Metric (gorarakelyan)

5.16.66 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)

5.16.67 1.2.9 Mar 16 2020

- Alert prerequisites installation message for running board (gorarakelyan)

5.16.68 1.2.8 Mar 15 2020

- Update profiler interface for keras (gorarakelyan)

5.16.69 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)

5.16.70 1.2.6 Feb 28 2020

- Add learning rate update tracking (gorarakelyan)

5.16.71 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)

5.16.72 1.2.4 Feb 20 2020

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

5.16.73 1.2.3 Feb 13 2020

- Set minimum required python version to 3.5.2 (gorarakelyan)

5.16.74 1.2.2 Feb 13 2020

- Downgrade required python version (gorarakelyan)

5.16.75 1.2.1 Feb 13 2020

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

5.16.76 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)

5.16.77 1.1.1 Jan 14 2020

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

5.16.78 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)

5.16.79 1.0.2 Jan 7 2020

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

5.16.80 1.0.1 Dec 26 2019

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

5.16.81 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)

5.16.82 0.2.9 Nov 30 2019

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

5.16.83 0.2.8 Nov 30 2019

- Update aim auth public key search algorithm (gorarakelyan)

5.16.84 0.2.7 Nov 14 2019

- Update dependencies torch and torchvision versions (sgevorg)

5.16.85 0.2.6 Nov 5 2019

- Update aim track logger (gorarakelyan)

5.16.86 0.2.5 Nov 4 2019

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

5.16.87 0.2.4 Nov 3 2019

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

5.16.88 0.2.3 Nov 3 2019

- Update package requirements (gorarakelyan)

5.16.89 0.2.2 Nov 1 2019

- Update package requirements (sgevorg)

5.16.90 0.2.1 Nov 1 2019

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

5.16.91 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)

5.16.92 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

SIX

INDICES AND TABLES

- genindex
- modindex

PYTHON MODULE INDEX

a

INDEX

J

K

L

M

N

Q

R

S

T

V

W