# GitHub - aimhubio/aim: Aim — a super-easy way to record, search and compare 1000s of ML training runs

https://github.com/aimhubio/aim

None Tue Jul, 20 19:18


**A super-easy way to record, search and compare 1000s of ML training runs**



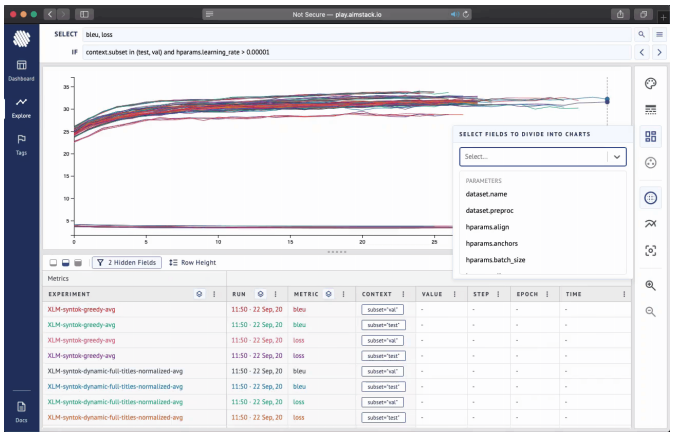




---

🔥 Try out Aim at play.aimstack.io 🔥
Watch the tutorial video

Join the Aim community on Slack




Integrate seamlessly with your favorite tools


Aim is an open-source comparison tool for AI experiments. With more resources and complex models more experiments are ran than ever. Use Aim to deeply inspect thousands of hyperparameter-sensitive training runs at once.

# Getting Started in 3 Steps

Follow the steps below to get started with Aim.

### 1. Install Aim on your training environment

*Prerequisite: You need to have python3 and pip3 installed in your environment before installing Aim*

### 2. Integrate Aim with your code

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

*See documentation here.*

- Integrate PyTorch Lightning

```
from aim.pytorch_lightning import AimLogger

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

*See documentation here.*

- Integrate 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],
    # ...
)
# ...
```

*See documentation here.*

- Integrate 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')
])
# ...
```

*See documentation here.*

- Integrate 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])
# ...
```

*See documentation here.*

**3. Run the training as usual and start Aim UI**

Jump to [Overview] [SDK Specifications] [Use Cases]

# Overview

Aim helps you to compare 1000s of training runs at once through its framework-agnostic python SDK and performant UI.

While using Aim SDK you create a Session object. It handles the tracking of metrics and parameters.

When the training code is instrumented with Aim SDK's Python Library and ran, Aim creates the `.aim` repository in your specified path and stores the data. Otherwise the data is created and stored in working directory.

Additionally, Aim SDK also gives you flexibility to:

- use multiple sessions in one training script to store multiple runs at once. When not initialized explicitly, Aim creates a default session.
- use experiments to group related runs together. An experiment named `default` is created otherwise.
- use integrations to automate tracking

Jump to [Getting Started] [SDK Specifications] [Use Cases]

# Democratizing AI Dev tools

# The mission...

Aim's mission is to democratize AI dev tools. We believe that the best AI tools need to be:

- open-source, open-data-format, community-driven, extensible
- have great UI/UX, CLI and other interfaces for automation
- performant both on UI and data

# Our motivation...

Existing open-source tools (TensorBoard, MLFlow) are super-inspiring.

However we see lots of improvements to be made. Especially around issues like:

- ability to handle 1000s of large-scale experiments
- actionable, beautiful and performant visualizations
- extensibility - how easy are the apis for extension/democratization?

These problems are a huge motivation.

We are inspired to build beautiful, scalable AI dev tools with great APIs. That's what unites the Aim community.

Join us, help us build the future of AI tooling!

# SDK Specifications

# Session

Session is the main object that tracks and stores the ML training metadata (metrics and hyperparams).

Use Session arguments to define:

- custom path for `.aim` directory
- experiment name: each session is associated with an experiment
- run name/message

Use Session methods to specify:

- the metrics of your training run(s)
- the hyperparameters of your training run(s)

*Class* aim.**Session**() *source*

*Arguments*

- **repo** - Full path to parent directory of Aim repo - the `.aim` directory. By default current working directory.
- **experiment** - A name of the experiment. By default `default`. Use experiments to group related runs together.

- **flush_frequency** - The frequency per step to flush intermediate aggregated values of metrics to disk. By default per `128` step.
- **block_termination** - If set to `True` process will wait until all the tasks are completed, otherwise pending tasks will be killed at process exit. By default `True`.
- **run** - A name of the run. If run name is not specified, universally unique identifier will be generated.
- **system_tracking_interval** - System resource usage tracking interval in seconds. By default 10 seconds. In order to disable system tracking set `system_tracking_interval=0`.

*Methods*

- `track()` - Tracks the training run metrics associated with the Session
- `set_params()` - Sets the params of the training run associated with the Session
- `flush()` - Flushes intermediate aggregated metrics to disk. This method is called at a given frequency and at the end of the run automatically.
- `close()` - Closes the session. If not invoked, the session will be automatically closed when the training is done.

*Returns*

Session object to attribute recorded training run to.

*Example*

- Here are a few examples of how to use the `aim.Session` in code.

*The Default Session*

When no session is explicitely initialized, a default Session object is created by Aim.

When `aim.track` or `aim.set_params` are invoked, underneath the default session object's `track` and `set_param` are called.

# track

Session.**track**(*value, name='metric_name' [, epoch=epoch] [, **context_args]*) source

*Parameters*

- **value** - the metric value of type `int` / `float` to track/log

- **name** - the name of the metric of type `str` to track/log (preferred divider: `snake_case`)
- **epoch** - an optional value of the epoch being tracked
- **context_args** - any set of other parameters passed would be considered as key-value context for metrics

*Example*

```
session_inst = aim.Session()

session_inst.track(0.01, name='loss', epoch=43, subset='train', dataset='train_1')
session_inst.track(0.003, name='loss', epoch=43, subset='val', dataset='val_1')
```

Once tracked this way, the following search expressions are enabled:

```
loss if context.subset in (train, val) # Retrieve all losses in both train and val phase
loss if context.subset == train and context.dataset in (train_1) # Retrieve all losses in
train phase with given datasets
```

Please note that any key-value could be used to track this way and enhance the context of metrics and enable even more detailed search.

Search by context example here:

## set_params

Session.**set_params**(*dict_value, name*) *source*

*Parameters*

- **dict_value** - Any dictionary relevant to the training
- **name** - A name for dictionaries

*Example*

```
session_inst = aim.Session()

 # really any dictionary can go here
hyperparam_dict = {
  'learning_rate': 0.0001,
  'batch_siz': 32,
}
session_inst.set_params(hyperparam_dict, name='params')
```

The following params can be used later to perform the following search experssions:

```
loss if params.learning_rate < 0.01 # All the runs where learning rate is less than 0.01
loss if params.learning_rate == 0.0001 and params.batch_size == 32 # all the runs where
learning rate is 0.0001 and batch_size is 32
```

***Note:*** If the `set_params` is called several times with the same name all the dictionaries will add up in one place on the UI.

# flush

Session.**flush**() *source*

Aim calculates intermediate values of metrics for aggregation during tracking. This method is called at a given frequency(see Session) and at the end of the run automatically. Use this command to flush those values to disk manually.

# Instrumentation

Use Python Library to instrument your training code to record the experiments.

The instrumentation only takes 2 lines:

```
# Import aim
import aim

# Initialize a new session
session_inst = Session()
```

Afterwards, simply use the two following functions to track metrics and any params respectively.

```
session_inst.set_params(hyperparam_dict, name='dict_name')

for iter, sample in enumerate(train_loader):
  session_inst.track(metric_val, name='metric_name', epoch=current_epoch)
```

Jump to [Getting Started] [Overview] [Use Cases]

# Integrations

We have integrated Aim to Tensorflow, Keras and Pytorch Lightning to enable automatic tracking. It allows you to track metrics without the need for explicit track statements.

# TensorFlow and Keras

Pass an instance of `aim.tensorflow.AimCallback` to the trainer callbacks list.

***Note:*** Logging for pure `keras` is handled by `aim.keras.AimCallback`

*Parameters*

- **repo** - Full path to parent directory of Aim repo - the `.aim` directory (optional)
- **experiment** - A name of the experiment (optional)

*Example*

```
import aim

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

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

TensorFlow v1 full example here
TensorFlow v2 full example here
Keras full example here

# PyTorch Lightning

Pass `aim.pytorch_lightning.AimLogger` instance as a logger to the `pl.Trainer` to log metrics and parameters automatically.

*Parameters*

- **repo** - Full path to parent directory of Aim repo - the `.aim` directory (optional)
- **experiment** - A name of the experiment (optional)
- **train_metric_prefix** - The prefix of metrics names collected in the training loop. By default `train_` (optional)
- **test_metric_prefix** - The prefix of metrics names collected in the test loop. By default `test_` (optional)
- **val_metric_prefix** - The prefix of metrics names collected in the validation loop. By default `val_` (optional)
- **flush_frequency** - The frequency per step to flush intermediate aggregated values of metrics to disk. By default per `128` step. (optional)

- **system_tracking_interval** - System resource usage tracking interval in seconds. By default 10 seconds. In order to disable system tracking set `system_tracking_interval=0`. (optional)

*Example*

```
from aim.pytorch_lightning import AimLogger

...
# Initialize Aim PL logger instance
aim_logger = AimLogger(experiment='pt_lightning_exp')

# Log parameters (optional)
aim_logger.log_hyperparams({
  'max_epochs': 10,
})

trainer = pl.Trainer(logger=aim_logger)
trainer.fit(model, train_loader, val_loader)
...
```

Full example here

# Hugging Face

Pass `aim.hugging_face.AimCallback` instance as a callback to the `transformers.Trainer` to log metrics and parameters automatically.

*Parameters*

- **repo** - Full path to parent directory of Aim repo - the `.aim` directory (optional)
- **experiment** - A name of the experiment (optional)
- **system_tracking_interval** - System resource usage tracking interval in seconds. By default 10 seconds. In order to disable system tracking set `system_tracking_interval=0`. (optional)

*Example*

```
from aim.hugging_face import AimCallback

# ...
# Initialize Aim callback instance
aim_callback = AimCallback(repo='/path/to/logs/dir', experiment='mnli')

# Initialize trainer
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,
    compute_metrics=compute_metrics,
    tokenizer=tokenizer,
    data_collator=data_collator,
    callbacks=[aim_callback]
)
# ...
```

Full example here

# XGBoost

Pass `aim.xgboost.AimCallback` instance as a callback to the `xgboost.train` to log metrics automatically.

*Parameters*

- **repo** - Full path to parent directory of Aim repo - the `.aim` directory (optional)
- **experiment** - A name of the experiment (optional)
- **system_tracking_interval** - System resource usage tracking interval in seconds. By default 10 seconds. In order to disable system tracking set `system_tracking_interval=0`. (optional)
- **flush_frequency** - The frequency per step to flush intermediate aggregated values of metrics to disk. By default per `128` step. (optional)

*Example*

```
from aim.xgboost import AimCallback

# ...
# Initialize Aim callback instance
aim_callback = AimCallback(repo='/path/to/logs/dir', experiment='experiment_name')

# Initialize trainer
bst = xgb.train(param, xg_train, num_round, watchlist, callbacks=[
  aim_callback,
])
# ...
```

Full example here

Jump to [Getting Started] [Overview] [Use Cases]

# Command Line Interface

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. |
| `experiment` | Creates a new experiment to group similar training runs into. |
| `up` | Runs Aim web UI for the given repo |

## init

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

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

***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.

## version

Display the Aim version installed.

## experiment

Create new experiments to organize the training runs. Here is how it works:

```
$ aim experiment COMMAND [ARGS]
```

| Command | Args | Description |
|---|---|---|
| `add` | `-n` \| `--name <exp_name>` | Add new experiment with a given name. |
| `checkout` | `-n` \| `--name <exp_name>` | Switch/checkout to an experiment with given name. |
| `ls` | | List all the experiments of the repo. |
| `rm` | `-n` \| `--name <exp_name>` | Remove an experiment with the given name. |

***Disclaimer:*** Removing the experiment also removes the recorded experiment runs data.

## up

Start the Aim web UI locally.

| 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* |
| `--tf_logs <logs_dir_path>` | Use Aim to search and compare TensorBoard experiments. More details in TensorBoard Experiments |
| `--dev` | Run UI in development mode. |

***Please make sure to run `aim up` in the directory where `.aim` is located.***

Jump to [Getting Started] [Overview] [Use Cases]

# Use Cases

# Searching Experiments

AimQL is a super simple, python-like search that enables rich search capabilities to search experiments. Here are the ways you can search on Aim:

- **Search by experiment name** - `experiment == {name}`
- **Search by run** - `run.hash == '{run_hash}'` or `run.hash in ('{run_hash_1}', '{run_hash_2}')` or `run.archived is True`

- **Search by param** - `params.{key} == {value}`
- **Search by context** - `context.{key} == {value}`

## Search Examples

- Display the losses and accuracy metrics of experiments whose learning rate is 0.001:
  - `loss, accuracy if params.learning_rate == 0.001`
- Display the train loss of experiments whose learning rate is greater than 0.0001:
  - `loss if context.subset == train and params.learning_rate > 0.0001`

Check out this demo project deployment to play around with search.

Jump to [Getting Started] [Overview] [SDK Specifications]

# TensorBoard Experiments

Easily run Aim on experiments visualized by TensorBoard. Here is how:

```
$ aim up --tf_logs path/to/logs
```

This command will spin up Aim on the TensorFlow summary logs and load the logs recursively from the given path. Use `tf:` prefix to select and display metrics logged with tf.summary in the dashboard, for example `tf:accuracy`.

Tensorboard search example here

Jump to [Getting Started] [Overview] [Specifications]

# Anonymized Telemetry

We constantly seek to improve Aim for the community. Telementry 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.

## 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.*

## How to opt out

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

Contributor Guide

Jump to [Getting Started] [Overview] [SDK Specifications]