Experiment Tracking

Workshop

Hovhannes Tamoyan

31 Oct. 2024

Who am I?

Hovhannes Tamoyan

@tamohannes

tamohannes.com

Agenda

- Introduction
- 🚀 Experiment Tracking Tools - Aim
- 🍽️ Break
- 🦁 UrarTU
- 👨‍💻 Hands-On Practice Session
- Post-Mortem

Introduction

Experiment Tracking Workshop

Why?

Why track experiments?

- Experiments are **not cheap**
- Models are **hard to debug** and comprehend, **training slows down**
- Training **runs** tend to **grow in numbers**

Why?

Training runs are multi-dimensional!

- **Code**
  - Hyperparams tuning
  - Model architecture
- **Data**
  - Sources, datasets (versions)
  - Samples, annotations
- **Training environment**
  - Hardware / Driver versions
  - Packages + dependencies
  - GPU performance
  - Node performance
- Images, text, audio and other metadata

Why?

Do we even need Experiment Tracking tools?

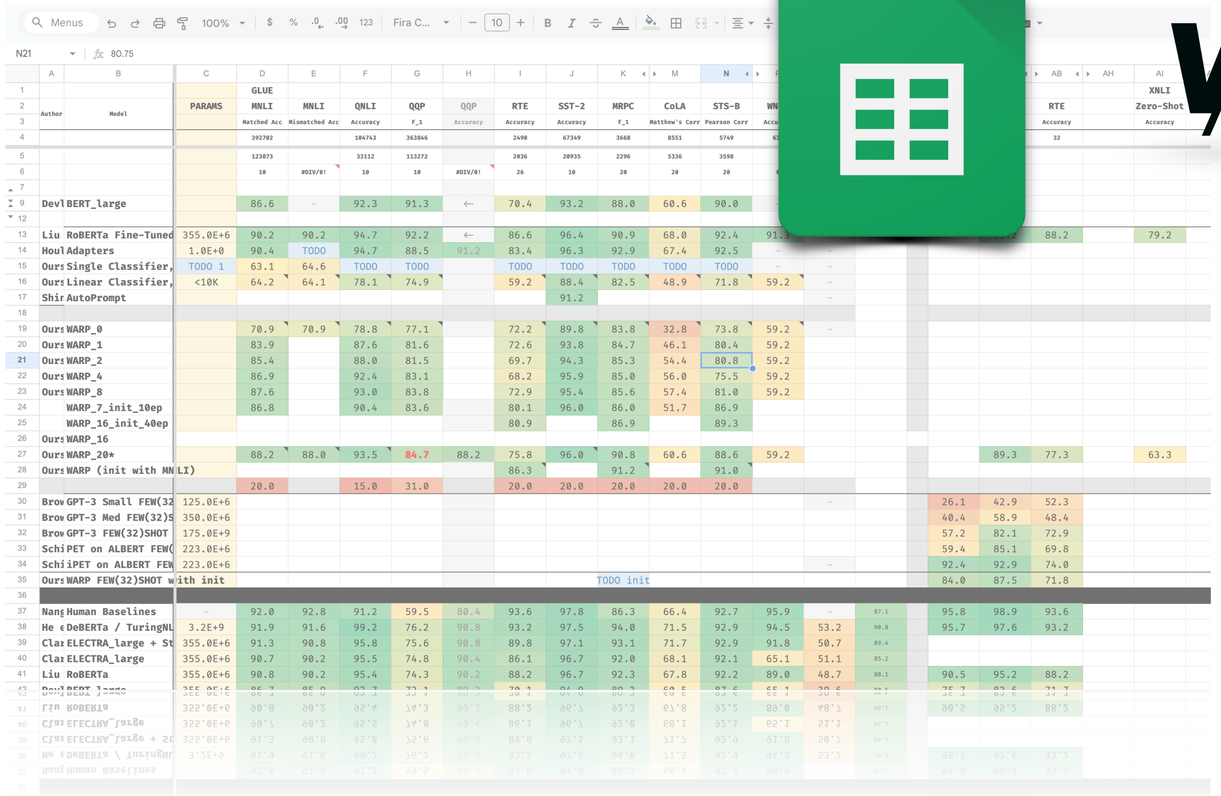
Experiment Tracking tools

Vs

Google Sheets

Why?

Do we even need Experiment Tracking tools?



VS

Why?

Do we need Experiment Tracking tools?

- **Purpose-Built** for **Machine Learning**
  - **Google Sheets**: Primarily designed for **general data organization** and **lightweight data analysis**.
  - **Experiment Tracking** Tools: Tailored specifically for **the complexities** of **machine learning workflows**, including **tracking hyperparameters**, **metrics**, and **model versions**.

Why?

Do we need Experiment Tracking tools?

- **Automation** and **Integration**
  - **Google Sheets**: Requires **manual entry** and updates, which can be **error-prone** and inefficient for **large datasets** or complex experiments.
  - **Experiment Tracking** Tools: Offer **automated logging** directly from the code, seamless **integration with ML libraries**, and automatic capture of the experiment setup and results.

Why?

Do we need Experiment Tracking tools?

- **Scalability**
  - **Google Sheets**: Limited in handling very large datasets or complex queries efficiently, which can be a bottleneck in large-scale ML projects.
  - **Experiment Tracking** Tools: Built to efficiently manage large volumes of data and complex experiments across multiple projects and teams.

Why?

Do we need Experiment Tracking tools?

- **Advanced Analytics** and **Visualization**
  - **Google Sheets**: Provides **basic charting** and data analysis tools, which might not suffice for the depth of analysis required in ML experiments.
  - **Experiment Tracking** Tools: Offer **sophisticated analytics** and **visualization tools** tailored for ML outputs, such as **learning curves**, performance metrics over time, and more.

Why?

Do we need Experiment Tracking tools?

- **Reproducibility**
  - **Google Sheets**: Challenges in ensuring reproducibility when managing ML experiments due to the **manual setup and lack of integration** with ML frameworks.
  - **Experiment Tracking** Tools: Enhance reproducibility by **automatically logging all aspects of the experiment environment**, including code, data, parameters, and environment settings.

Requirements

What do we need from Experiment Tracking tools?

Requirements

What do we need from Experiment Tracking tools?

- **System Information Logging**: Automatically logs system specifications and environment settings, ensuring that experiments are reproducible and that differences in hardware or software configurations are accounted for.

Requirements

What do we need from Experiment Tracking tools?

- **Code-base** and **Dataset Version Tracking**: Tracks versions of the code (git commit hash) and datasets used in experiments, allowing to revert to previous versions and understand changes over time.

Requirements

What do we need from Experiment Tracking tools?

- **Hyperparameter** and **Metric Tracking** and **Management**: Facilitates systematic logging and comparison of hyperparameters and metrics, making it easier to identify optimal configurations and understand model performance across experiments.

Requirements

What do we need from Experiment Tracking tools?

- **Artifact Storage** and **Management**: Manages all experiment artifacts, including configuration files and model checkpoints, in an organized manner, ensuring that they are easily accessible for future use and analysis.

Requirements

What do we need from Experiment Tracking tools?

- **Fast, Easy,** and **Elegant Result Retrieval Mechanisms (UI)**: Provides a user-friendly interface that allows quick and intuitive access to experimental results, simplifying the process of analyzing data and sharing findings.

Experiment Tracking Tools

Experiment Tracking Workshop

Session 1

Popular Experiment Tracking tools

What Experiment Tracking tools are available?

- **TensorBoard** 
- **MLflow** 
- **Weights & Biases** (wandb) 
- **AimStack** (Aim) 

TensorBoard vs Aim

- **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 - 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**
  - Embedding projector.
  - Neural network visualization.

MLflow vs Aim

- 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.
- Aim and MLflow are a perfect match - check out the aimlflow - the tool that enables Aim superpowers on Mlflow.
- **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.

Wandb vs Aim

- **Hosted vs self-hosted**
  - Weights and Biases is a hosted closed-source MLOps platform.
  - Aim is self-hosted, free and open-source experiment tracking tool.

Aim

github.com/aimhubio/aim

An easy-to-use & supercharged open-source AI metadata tracker

Aim logs your training runs and any AI Metadata, enables a beautiful UI to compare, observe them and an API to query them programmatically.

SEAMLESSLY INTEGRATES WITH:

TRUSTED BY ML TEAMS FROM:

AimStack offers enterprise support that's beyond core Aim. Contact via hello@aimstack.io e-mail.

Aim

github.com/aimhubio/aim

README | Code of conduct | Apache-2.0 license

Drop a star to support Aim ⭐ | Join Aim discord community

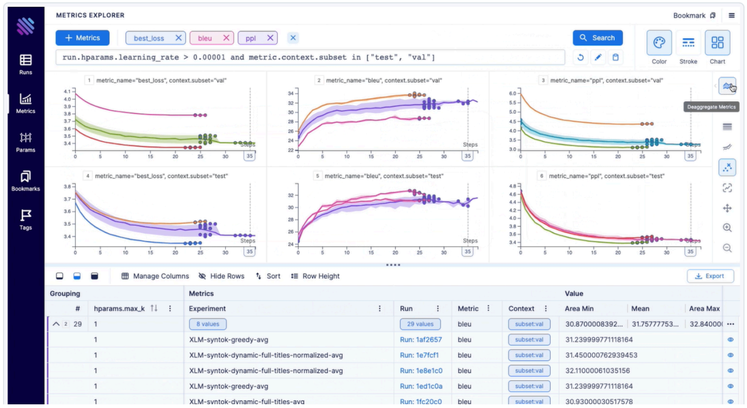
Aim

An easy-to-use & supercharged open-source experiment tracker

Aim logs your training runs and any AI Metadata, enables a beautiful UI to compare, observe them and an API to query them programmatically.

Aim | Follow @aimstackio | Medium

platform Linux | macOS | python >= 3.7 | pypi v3.25.0 | License Apache 2.0 | downloads 210k/week | issues 361 open



+ 4 deployments

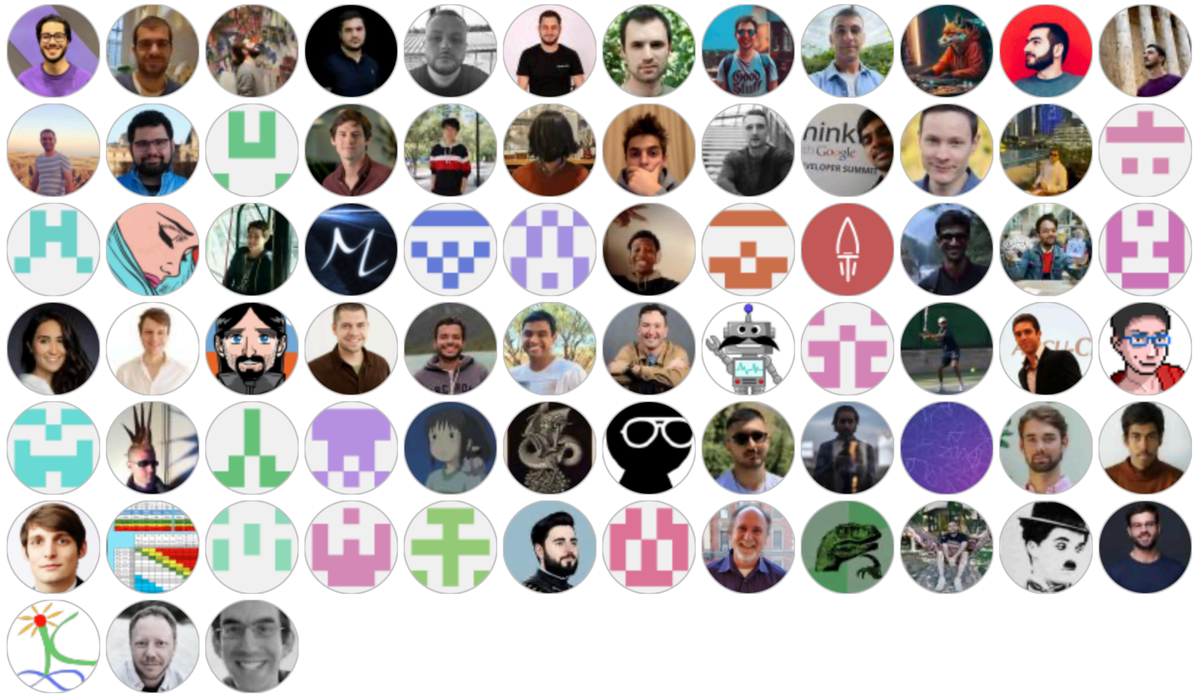
Languages

- Python 91.5%
- Cython 4.9%
- JavaScript 1.8%
- HTML 0.7%
- CSS 0.7%
- Shell 0.2%
- Other 0.2%

Contributing to Aim

Considering contibuting to Aim? To get started, please take a moment to read the CONTRIBUTING.md guide.

Join Aim contributors by submitting your first pull request. Happy coding! 😊



Made with contrib.rocks.

github.com/aimhubio/aim

- Recommended sources:
  - The Docs: https://aimstack.readthedocs.io/en/latest/overview.html
- Medium blogsposts:
  - https://medium.com/aimstack
- Aim tutorials:
  - https://aimstack.io/blog/tutorials
- Demos:
  - https://aimstack.io/#demos

Core Components

- Aim **SDK**:
  - Python interface to define and track any object
  - Query tracked metadata with fully supported pythonic expressions
  - Integrations with essential ML tools and frameworks
- Aim **Storage**:
  - Modular (runs isolation - easily copy, move, delete runs)
  - Extendable (easily store any python object)
- Aim **UI**:
  - Metadata management and visualization
  - Deep comparison and exploration of multi-dimensional metadata

Deep dive into Aim

- **Setup**: Local and remote tracking, the Run class
- **Tracking**: Tracking objects such as Metric, Text, Audio, and Image
- **Artifacts**: Storing objects
- **Adapters**: Integrating Aim into an existing project
- **Migrate**: Importing runs from other trackers into Aim
- **UI**
  - **Runs Management**: Run explorer, bookmarks and tags
  - **Explorers**: Metrics, Parameters, and Text explorers

Setup: Local and remote tracking, the Run class

- **Initializing Aim repository**:
- The space where all your training runs are logged

```
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.

Setup: Local and remote tracking, the Run class

- **Browsing results with Aim UI**:
- Open Aim UI and see the results using:

```
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:

Setup: Local and remote tracking, the Run class

- **Tracking data with Aim SDK**:
- To start tracking, first create aim.Run object:

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

Setup: Local and remote tracking, the Run class

- **Tracking data with Aim SDK**:
- 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
```

Setup: Local and remote tracking, the Run class

- **Tracking data with Aim SDK**:
- 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')
```

Setup: Local and remote tracking, the Run class

- **Server-side setup**:
- 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 utilizes http/ws protocols as its core transport layer.

```
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.

Setup: Local and remote tracking, the Run class

- **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.

```
from aim import Run

aim_run = Run(repo='aim://172.3.66.145:53800')

...
```

- Replace example IP with your tracking server IP/hostname

Tracking: Tracking objects such as Metric, Text, Audio, and Image

- 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**.

Tracking: Tracking objects such as Metric, Text, Audio, and Image

- 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**

Tracking: Tracking objects such as Metric, Text, Audio, and Image

- **Metric**: Tracking multiple values at once:

```
from aim import Run

aim_run = Run()

aim_run.track({'accuracy': 0.72, 'f1': 0.99}, context={'subset': 'train'})
```

Tracking: Tracking objects such as Metric, Text, Audio, and Image

- **Text**: 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
```

Tracking: Tracking objects such as Metric, Text, Audio, and Image

- 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:

```
from aim import Run, Text

run = Run()

aim_text = Text(‘some text')
run.track(aim_text, name=‘input prompt’, step=step)
```

Tracking: Tracking objects such as Metric, Text, Audio, and Image

- **Audio**: 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
```

Tracking: Tracking objects such as Metric, Text, Audio, and Image

- 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)

Tracking: Tracking objects such as Metric, Text, Audio, and Image

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

run = Run()

for step in range(1000):
    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)
```

Tracking: Tracking objects such as Metric, Text, Audio, and Image

- **Image**: 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
```

- The 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)

Tracking: Tracking objects such as Metric, Text, Audio, and Image

- Here’s an example of tracking image from file path

```
from aim import Run, Image

run = Run()

for step in range(1000):
    path = f"~/test_image_{step}.png"
    aim_image = Image(
        path,
        format='jpeg',
        optimize=True,
        quality=50
    )

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

Artifacts: Storing objects

- There are only two steps for logging artifacts with Aim:

```
from aim import Run

run = Run()
```

- Use S3 as artifacts storage:

```
run.set_artifacts_uri('s3://aim/artifacts/')
```

- Use file-system as artifacts storage

```
run.set_artifacts_uri('file:///home/user/aim/artifacts/')
```

Artifacts: Storing objects

- Aim will create directory with the name of `run.hash` and store all artifacts there.
- Note that setting artifacts storage URI is required only once per run.

```
# Log run configuration files as artifacts
run.log_artifact('config.yaml', name='run-config')
```

Adapters: Integrating Aim into an existing project

- 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
```

Adapters: Integrating Aim into an existing project

- 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')
```

Adapters: Integrating Aim into an existing project

```
# 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]
)
```

Adapters: Integrating Aim into an existing project

- Integration with **Pytorch Lightning**:
- We only require 2 steps to simply and easily inject Aim into pytorch lightining:

```
from aim.pytorch_lightning import AimLogger
```

Adapters: Integrating Aim into an existing project

- 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

```
aim_logger = AimLogger(
    experiment='aim_on_pt_lightning',
    train_metric_prefix='train_',
    val_metric_prefix='val_',
)
```

Adapters: Integrating Aim into an existing project

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

```
trainer = Trainer(gpus=1, progress_bar_refresh_rate=20, max_epochs=5, logger=aim_logger)
```

Migrate: Importing runs from other trackers into Aim

- Show **TensorBoard** logs in Aim:
- To convert **TensorBoard** logs, aim convert command must be run on your log directory.

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

Migrate: Importing runs from other trackers into Aim

- Show **Weights and Biases** logs in Aim:
- To be able to explore Weights & Biases runs with Aim, please run the WandB to Aim converter. All the metrics, tags, config, experiment description/notes and some artifacts will be converted and stored in .aim repo.
- Execute aim convert wandb command to start converting Weights & Biases experiments:

```
aim init
```

```
aim convert wandb --entity 'my_team' --project 'test_project'
```

Migrate: Importing runs from other trackers into Aim

- The convertor will iterate over all the experiments and create a distinct Aim run for each experiment. If you want to process only a single experiment, please specify the experiment id or name when running the command:

```
aim convert wandb --entity my_team --project test_project --run-id '<HASH_OF_RUN>'
```

UI

- **Runs Management**: Run explorer, bookmarks and tags
- **Explorers**: Metrics, Parameters, and Text explorers

Break

Experiment Tracking Workshop

UrarTU

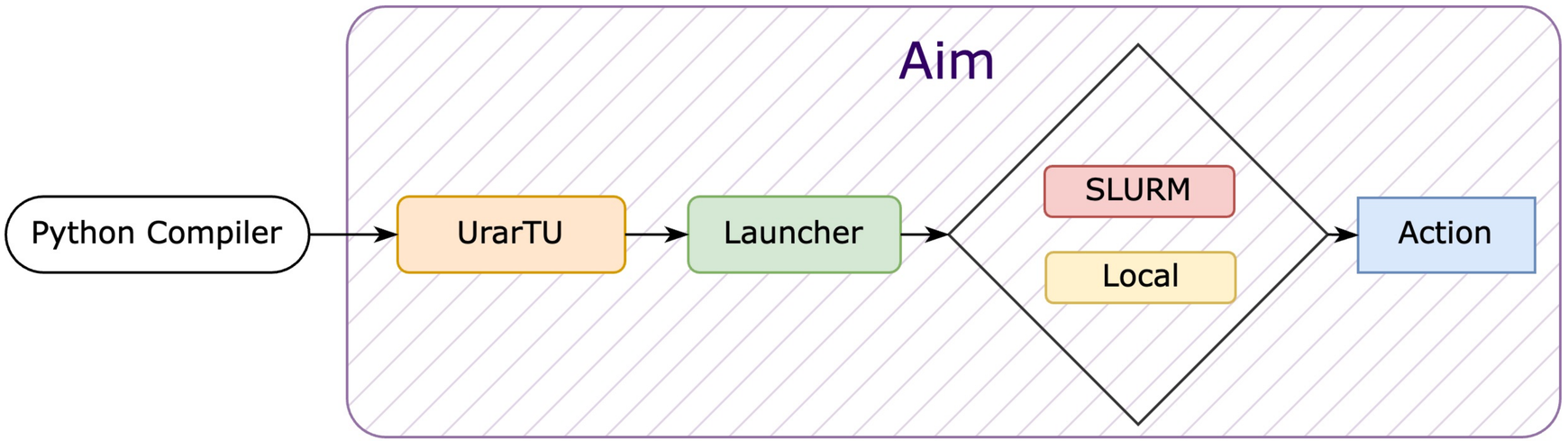
Experiment Tracking Workshop

Session 2

UrarTU

github.com/tamohannes/urartu

An open-source NLP framework that offers high-level wrappers designed for effortless launch, enhanced reproducibility, superior control, and unmatched flexibility for your experiments.



UrarTU

The structure of UrarTU with an “example” project

- **Module structure**: Project named “example” shall follow this structure

```
example
├── __init__.py
├── actions
│   ├── __init__.py
│   └── generate.py
├── configs
│   ├── __init__.py
│   └── action_config
│       └── generate.yaml
└── configs_tamoyan
    ├── __init__.py
    ├── aim.yaml
    └── slurm.yaml
```

UrarTU

The structure of UrarTU with an “example” project

- **generate.yaml** content

```yaml
# @package _global_
action_name: generate

aim:
  repo: ./

action_config:
  workdir: ./
  experiment_name: "Example - next token prediction"
  device: "cpu" # auto, cuda, cpu (default)

  task:
    model:
      type:
        _target_: urartu.models.causal_lm_model.CausalLMModel
      name: gpt2
      dtype: torch.float32
      cache_dir: ""

      generate:
        max_length: 100
        num_beams: 5
        no_repeat_ngram_size: 2

    dataset:
      type:
        _target_: urartu.datasets.hf_datasets.HFDatasets
      name: truthfulqa/truthful_qa
      subset: generation
      split: validation
      input_key: "question"
```

UrarTU

The structure of UrarTU with an “example” project

- **generate.yaml** is a general configuration file for the next token prediction project.
- If multiple team members are working on the same project and have their specific configurations, follow these steps:
  - Create a directory at the same level as the configs directory and name it `configs_{username}`, where `{username}` is your OS username.
  - Copy the content of the general configuration file and paste it into the `configs_{username}` directory.
  - Customize the specific settings as needed. Suppose I prefer my Aim repository to be a remote URL rather than a local path.

UrarTU

The structure of UrarTU with an “example” project

- **aim.yaml**

```
repo: aim://0.0.0.0:43800
log_system_params: true
```

UrarTU

The structure of UrarTU with an “example” project

- **slurm.yaml**

```yaml
use_slurm: true
name: "example"
comment: "example job"
partition: ""
account: ""
log_folder: "./slurm_logs"
time_hours: 1
time_minutes: 0
constraint: ""
mem_gb: 100
port_id: 40050
num_cpu_per_proc: 4
num_gpu_per_node: 4
num_nodes: 1
num_proc_per_node: 1
additional_parameters: {}
```

UrarTU

The structure of UrarTU with an “example” project

- **generate.py**

```
from omegaconf import DictConfig
from aim import Run, Text

from tqdm import tqdm

from urartu.common.action import Action
from urartu.common.dataset import Dataset
from urartu.common.model import Model


class Generate(Action):
    def __init__(self, cfg: DictConfig, aim_run: Run) -> None:
        super().__init__(cfg, aim_run)

    def main(self):
        model = Model.get_model(self.task_cfg.model)
        dataset = Dataset.get_dataset(self.task_cfg.dataset)

        for idx, sample in tqdm(enumerate(dataset.dataset)):
            prompt = sample[self.task_cfg.dataset.get("input_key")]
            self.aim_run.track(Text(prompt), name="input")

            output = model.generate(prompt)
            self.aim_run.track(Text(output), name=“output")
...
```

UrarTU

The structure of UrarTU with an “example” project

- **generate.py** cont.

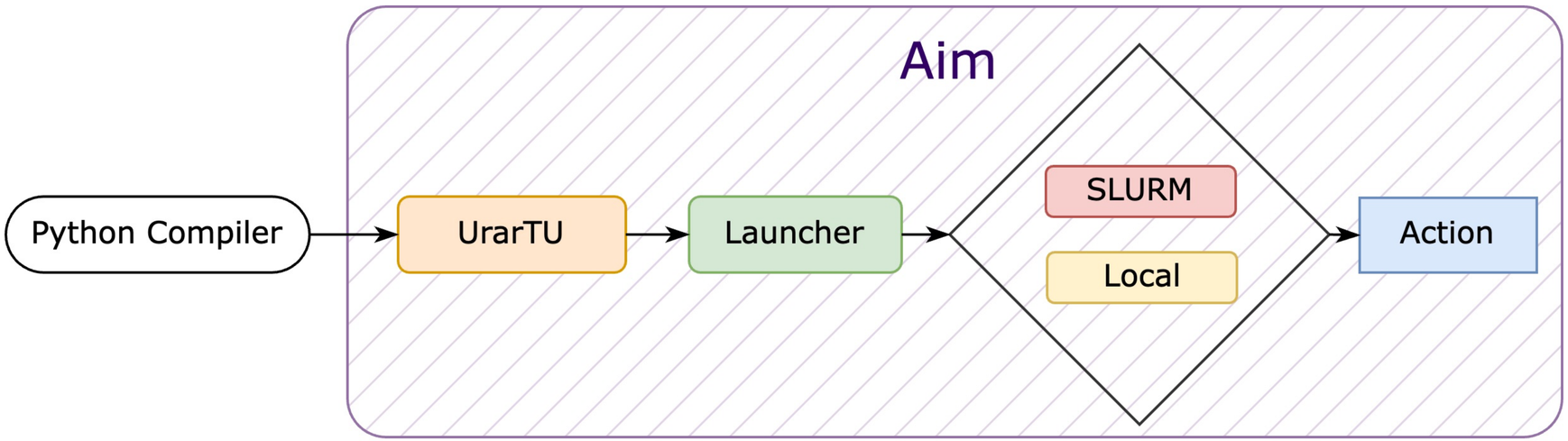
```
...

def main(cfg: DictConfig, aim_run: Run):
    example = Generate(cfg, aim_run)
    example.main()
```

UrarTU

The structure of UrarTU with an “example” project

- **Registrar**: Registers a given project name under a specified path

```
urartu register --name=example –path=PATH_TO_EXAMPLE_MODULE
```

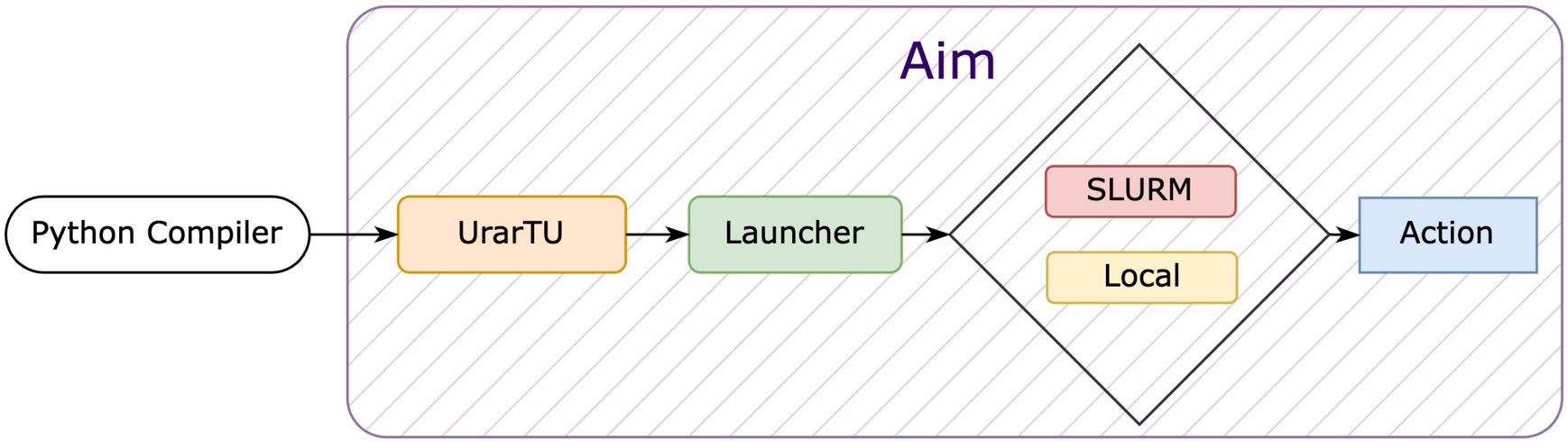


UrarTU

The structure of UrarTU with an “example” project

- **Launcher**: Submitit-based launcher launching jobs on local or slurm machine

```
urartu launch --name=example action_config=generate
```



Submit it!

github.com/facebookincubator/submitit

Submitit is a lightweight tool for submitting Python functions for computation within a Slurm cluster. It basically wraps submission and provide access to results, logs and more. Slurm is an open source, fault-tolerant, and highly scalable cluster management and job scheduling system for large and small Linux clusters. Submitit allows to switch seamlessly between executing on Slurm or locally.

- **Flexible Job Submission**: Submit any Python function, including
- **Error Handling**: Provides stack traces for failed jobs.
- **Preemption Handling**: Automatically requeues preempted
- **Checkpointing**: Supports stateful callable checkpointing when preempted or timed out.
- **Local and Slurm Execution**: Easily toggle between local execution and Slurm submission.
- **Task Management**: Offers easy access to task-specific details like local/global rank.

UrarTU

The structure of UrarTU with an “example” project

- **Configs**: Hyra-based hirarcical configurations
- Two ways to override configs: **Custom Config Files** and **CLI Approach**

```
urartu launch --name=example action_config=generate --slurm=slurm --aim=aim
```

```
urartu launch --name=example action_config=generate action_config.workdir=PATH_TO_WORKDIR
```

Hydra

github.com/facebookresearch/hydra

A framework for elegantly configuring complex applications

- **Dynamic Configuration**: Allows the creation and composition of hierarchical configurations dynamically.
- **Command-line Integration**: Seamlessly integrates with command-line interfaces to override configuration options.
- **Plug-in System**: Supports custom plug-ins to extend functionality, including integrations with popular ML libraries.

UrarTU

The structure of UrarTU with an “example” project

- **Multirun**: Submites multiple jobs with all the combinations using the given set of configs.
- Two ways to run multiple jobs: **Custom Config Files** and **CLI Approach**

```
urartu launch –name=example --multirun action_config=generate --slurm=slurm –aim=aim
```

```
hydra:
  sweeper:
    params:
      ++action_config.task.model.dtype: torch.float32,torch.bfloat16
```

```
urartu launch –name=example --multirun action_config=generate --slurm=slurm –aim=aim
            action_config.task.model.dtype= torch.float32,torch.bfloat16
```

Hands-On Practice Session

Experiment Tracking Workshop

Session 3

Machine Translation

Build an English to German (en-de) NMT system

github.com/tamohannes/experiment_tracking_workshop

- ⚠️ **DON’T** open the **main_w_aim.py** for now

- **main.py**: Simple NMT system fine-tuning script using T5 model
- **config.yaml**: Basic yaml config that is loaded using hydra

Machine Translation

Build an English to German (en-de) NMT system

github.com/tamohannes/experiment_tracking_workshop

- Integrate Aim into the existing source code.
- Initialize a new Run.
- Log the hyperparameters.
- Track useful metrics.
- Store the target, predicted, and source sequences.
- Fine-tune with different hyperparameters.
- *Optionaly* perform this using UrarTU

Post-Mortem

Experiment Tracking Workshop

Session 4

Analysis and Discussion

English to German (en-de) NMT system

- Get insights from the **Explorers**:
  - **Single Run**
  - **Runs**
  - **Metrics**
  - **Parameters**
  - **Text**

Aim Discord Channel

https://discord.com/invite/zXq2NfVdtF

Thank You

Questions?

Hovhannes Tamoyan

31 Oct. 2024