

Containerization. Pipeline API and Airflow setup

GitHub repository: <https://github.com/xphoenixua/brain-invaders-mlops>

Quick recap on ML pipeline

This machine learning pipeline is designed to classify P300 event-related potentials from electroencephalography (EEG) signals. The system processes raw EEG data to extract features, trains a classification model, and provides an API for inference.

Raw EEG data, specifically from the visual P300 Brain-Computer Interface (BCI) paradigm, is initially provided in individual subject-specific Parquet files in the project root directory on local machine. This raw data contains 16 EEG channels and 1 stimulus channel, sampled at 512 Hz.

The pipeline processes this raw data through a series of steps to prepare it for machine learning. Preprocessing involves filtering the EEG signals, segmenting the continuous data into short epochs time-locked to visual stimuli, performing baseline correction, and applying basic artifact rejection. Features are then extracted from these cleaned epochs by downsampling the data to 100 Hz and concatenating the time-series values across channels into a single feature vector. This processed data, along with corresponding labels (Target or Non-Target), is stored in an object storage emulating S3 API, MinIO, ready for model training or inference.

The core machine learning model used for classification is Linear Discriminant Analysis (LDA), which takes the extracted feature vectors as input to predict the presence or absence of a P300 response.

Introduction to service-based pipeline structure

The pipeline is composed of several microservices, each executing a specific function in the data and machine learning workflow.

MinIO acts as the central object storage, hosting `p300-landing-zone` for newly arrived raw data, `p300-raw-data` for ingested raw data ready for processing, `p300-processed-features` for prepared features and labels, and `p300-models` for versioned model artifacts.

The `minio-init` service is a one-off job that establishes this initial bucket structure and populates the `p300-landing-zone` with new raw subject data.

An Airflow DAG orchestrates the core data processing: it first simulates the arrival of new raw data into the `p300-landing-zone`, then ingests this data by copying it to `p300-raw-data` and removing it from the landing zone. The DAG then identifies subjects in `p300-raw-data` not yet processed and triggers the `preprocessing-job` for them. The `preprocessing-job`

processes raw data from `p300-raw-data` and stores the resulting features in `p300-processed-features`.










The `training-job` reads from `p300-processed-features` to train a model, saves the versioned model and its details to `p300-models`, and updates the active model pointer there. The `p300-serving` service continuously hosts the active model from `p300-models` via an API for real-time inference requests.

Finally, the `app-simulator-job` retrieves processed data, sends inference requests to `p300-serving`, and collects detailed performance metrics, which are then logged to the external Weights & Biases service for tracking and visualization.

Initialization

`docker-compose.yaml`

```
user@MacBook-Pro braininvaders_mlops % docker compose up --build -d
```




<input type="checkbox"/>	Name	Container ID	Image	Port(s)	CPU (%)	Last started
<input type="checkbox"/>	 <code>braininvaders_mlops</code>	-	-	-	2.8%	60 seconds ago
<input type="checkbox"/>	 <code>minio-server</code>	50ba40c07ccc	minio/minio:latest	9000:9000  Show all ports (2)	2.54%	1 minute ago
<input type="checkbox"/>	 <code>minio-init</code>	d1ea52b5728e	braininvaders_mlops-mini		0%	1 minute ago
<input type="checkbox"/>	 <code>model-serving-service</code>	b9495c355f18	braininvaders_mlops-moc	8000:8000 	0.26%	1 minute ago
<input type="checkbox"/>	 <code>preprocessing-job-1</code>	d9d4fb25ca25	braininvaders_mlops-prep		0%	1 minute ago
<input type="checkbox"/>	 <code>training-job-1</code>	7cb88b06f917	braininvaders_mlops-train		0%	1 minute ago
<input type="checkbox"/>	 <code>app-simulator-job-1</code>	41b298745b84	braininvaders_mlops-app		0%	1 minute ago

```
user@MacBook-Pro braininvaders_mlops % docker compose ps
```

NAME	IMAGE	COMMAND	SERVICE	CREATED
STATUS	PORTS			
minio-server	minio/minio:latest	"/usr/bin/docker-ent..."	minio-server	About a minute ago
Up About a minute (healthy)	0.0.0.0:9000-9001->9000-9001/tcp			
model-serving-service	braininvaders_mlops-model-serving-service	"uvicorn main:app --..."	model-serving-service	About a minute ago
Up About a minute	0.0.0.0:8000->8000/tcp			

`minio-init` service set up initial buckets for further processing and was stopped after that. Meanwhile, `minio-server` hosts the actual MinIO storage instance throughout the whole app lifecycle. We can see the initial storage structure by accessing MinIO web UI:



Filter Buckets				
Name	Objects	Size	Access	
 p300-landing-zone	0	0.0 B	R/W	
 p300-models	0	0.0 B	R/W	
 p300-processed-features	0	0.0 B	R/W	
 p300-raw-data	0	0.0 B	R/W	

I initialized `p300-models` bucket to make this pipeline more demonstrative. Since I use Weights&Biases for logging post-training and post-inference metrics, it would be natural to also dump model artifacts there. However, when I started developing this whole system I hadn't yet come to model artifacts monitoring stage, so for then I configured this MinIO model logging version.

airflow-compose.yaml

```
user@MacBook-Pro braininvaders_mlops % docker compose -f airflow-compose.yaml  
up --build -d
```

```
[+] Running 9/9  
✓ Network airflow_default Created 0.0s  
✓ Volume "braininvaders_mlops_postgres-db-volume" Created 0.0s  
✓ Container braininvaders_mlops-redis-1 Healthy 6.7s  
✓ Container braininvaders_mlops-postgres-1 Healthy 6.7s  
✓ Container braininvaders_mlops-airflow-init-1 Exited 14.5s  
✓ Container braininvaders_mlops-airflow-worker-1 Started 14.5s  
✓ Container braininvaders_mlops-airflow-triggerer-1 Started 14.5s  
✓ Container braininvaders_mlops-airflow-scheduler-1 Started 14.5s  
✓ Container braininvaders_mlops-airflow-webserver-1 Started 14.6s
```

Creating Airflow-related instances required a separate compose file to build and run them, since I downloaded a predefined `docker-compose.yaml` file from Airflow official website with

```
_curl -Lf0 'https://airflow.apache.org/docs/apache-airflow/2.10.5/docker-  
compose.yaml_'
```

I could merge my custom Docker compose for all the services shown earlier in this report, but it would be messy so I decided to keep them separate. Also, I tried Airflow 3.0.1 but encountered an immense number of issues throughout setting it up, even though I followed exactly the same configuration steps as for Airflow 2.10.5 that I tried after that. I found Airflow 3.0.1 much easier to work with via its UI (much clearer what to press than 2.10.5 for a noob like me) but setting the DAG was a nightmare with all the strange bugs I had.

We can access the Airflow web UI to see that our DAG is set up correctly with all the tasks defined. I also added those dummy operator jobs like `start_pipeline_run` and `end_pipeline_run` to identify downstream and upstream clearer and serve as indicators that the DAG script is readable.

Sign In

Enter your login and password below:

Username:

airflow

Password:

.....

Sign In

Airflow

DAGs

Cluster Activity

Datasets

Security

Browse

Admin

Docs

19:16 (+03:00)

AA

DAG: p300_full_ingest_and_process_pipeline

Schedule: None

Next Run ID: None

24.05.2025 19:15:51

All Run Types

All Run States

Clear Filters

Auto-refresh

25

Press **shift** + **/** for Shortcuts

deferred

failed

queued

removed

restarting

running

scheduled

shutdown

skipped

success

up_for_reschedule

up_for_retry

upstream_failed

no_status

DAG

p300_full_ingest_and_process_pipeline

Details

Graph

Gantt

Code

Event Log

Run Duration

Task Duration

Calendar

DAG Summary

Total Tasks

6

EmptyOperators

2

@tasks

3

DockerOperator

1

DAG Details

Dag display name

p300_full_ingest_and_process_pipeline

Dag id

p300_full_ingest_and_process_pipeline

Description

null

Fileloc

/opt/airflow/dags/process_raw_subjects.py

Has import errors

false

Has task concurrency limits

false

start_pipeline_run

simulate_raw_data_arrival

ingest_subjects_from_landing_zone

identify_subjects_for_preprocessing

preprocess_subject_batch []

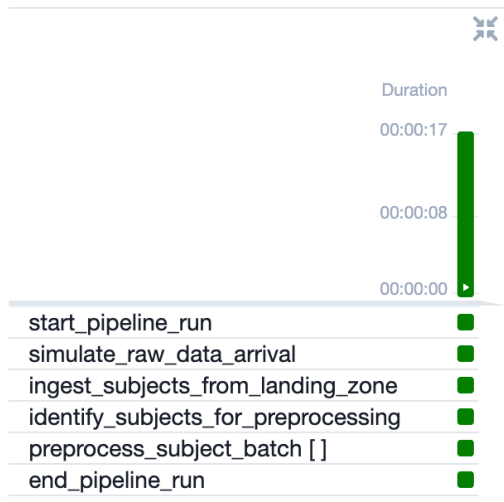
end_pipeline_run

Prepare data for training

To trigger the DAG for it to make data for training available, I press the corresponding button in the UI.



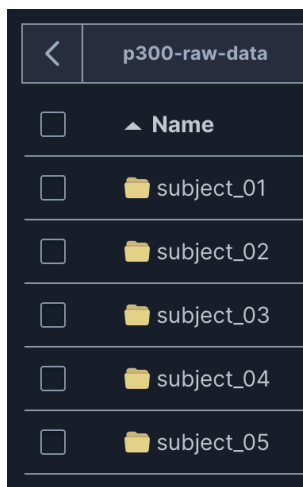
After waiting for some time, we see that our jobs executed successfully.



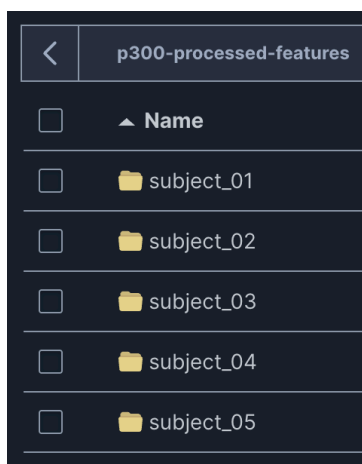
We can see that both buckets (however, only `p300-processed-features` is relevant during the training stage) were populated with data. `p300-landing-zone` served as an intermediate bucket for storing incoming data from a local directory

`./initial_data_for_minio/all_raw_subjects/` . It is not necessary but I just wanted to simulate that the data that is coming in batches is temporarily transferred to some landing bucket. After successfully copying the data from there to a "more persistent" storage – `p300-raw-data` bucket, it gets deleted from `p300-landing-zone` , again, to simulate on-line data transfer. I am not really sure if it is how it usually happens in the production but I tried to be creative with making my pipeline as realistic as possible so I decided to stick with this idea.

p300-raw-data bucket



p300-processed-features bucket



Training model

I ran my DAG once again to have 10 subjects in my preprocessed bucket.

Then I run this command:

```
user@MacBook-Pro braininvaders_mlops % docker compose run --rm --build  
training-job python train_model.py --training_subjects_percentage 1.0
```

Here we can specify how much of the whole preprocessed bucket storage's subjects will be used for training. I set `1.0` so 100% of them in `p300-processed-features` bucket will be used for training.

Overall, I use this command because `training-job` is a batch job, not a continuously running server. It needs to perform a task and then exit. So it might be a bit confusing with the fact that I already built this service with `docker-compose.yml` at the beginning but that was only the image for this container (which I here rebuilt once again, but that's only for reassurance because I changed the code there frequently when compose stack was already initialized). So I basically run a script inside this container's environment.

Here are the logs for this training run:

```
=> => unpacking to docker.io/library/braininvaders_mlops-training-job:latest 0.0s
=> [training-job] resolving provenance for metadata file 0.0s
wandb: Currently logged in as: xphoenixua to https://api.wandb.ai. Use `wandb login --relogin` to force relogin
wandb: Tracking run with wandb version 0.19.11
wandb: Run data is saved locally in /app/wandb/run-20250524_163940-ffr2utm0
wandb: Run `wandb offline` to turn off syncing.
wandb: Syncing run train_v20250524163939_on_10subjects
wandb: ★View project at https://wandb.ai/xphoenixua/p300-bci-mlops
wandb: 🚀View run at https://wandb.ai/xphoenixua/p300-bci-mlops/runs/ffr2utm0
--- Starting Training Pipeline for 10 Subjects: [1, 2, 3, 4, 5, 6, 7, 8, 9, 10] ---
  Downloading processed data for Subject 01 from p300-processed-features...
  Downloading processed data for Subject 02 from p300-processed-features...
  Downloading processed data for Subject 03 from p300-processed-features...
  Downloading processed data for Subject 04 from p300-processed-features...
  Downloading processed data for Subject 05 from p300-processed-features...
  Downloading processed data for Subject 06 from p300-processed-features...
  Downloading processed data for Subject 07 from p300-processed-features...
  Downloading processed data for Subject 08 from p300-processed-features...
  Downloading processed data for Subject 09 from p300-processed-features...
  Downloading processed data for Subject 10 from p300-processed-features...
Combined data: X=(5113, 1120), Labels: [1 2]
Internal Train set: (4090, 1120), Internal Validation set: (1023, 1120)
  Training LDA model...
Model classes after fitting: [1 2]
```

```

--- Internal Validation Set Evaluation ---
AUC: 0.8405266339570651
Accuracy: 0.8690
Kappa: 0.4360
Confusion Matrix (Target=1, NonTarget=0):
[[820  37]
 [ 97  69]]

Using model version: v20250524163939
Model artifacts uploaded and W&B artifact logged.

```

```

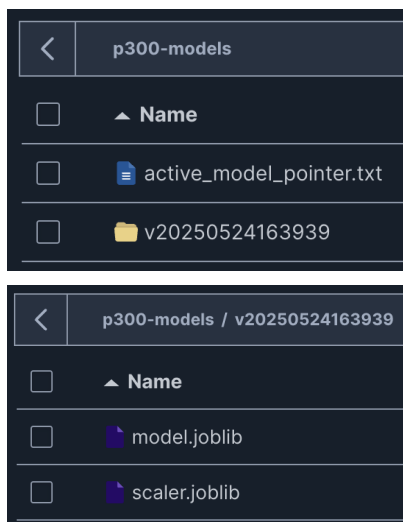
wandb:
wandb:
wandb: Run history:
wandb:     accuracy _
wandb:     auc _
wandb: conf_matrix_fn _
wandb: conf_matrix_fp _
wandb: conf_matrix_tn _
wandb: conf_matrix_tp _
wandb:     kappa _
wandb:
wandb: Run summary:
wandb:     accuracy 0.86901
wandb:     auc 0.84053
wandb: conf_matrix_fn 97
wandb: conf_matrix_fp 37
wandb: conf_matrix_tn 820
wandb: conf_matrix_tp 69
wandb:     kappa 0.43603
wandb:
wandb: 📄View run train_v20250524163939_on_10subjects at: https://wandb.ai/xphoenixua/p300-bci-mlops/runs/ffr2utm0
wandb: ★View project at: https://wandb.ai/xphoenixua/p300-bci-mlops
wandb: Synced 5 W&B file(s), 0 media file(s), 3 artifact file(s) and 0 other file(s)
wandb: Find logs at: ./wandb/run-20250524_163940-ffr2utm0/logs
--- Finished Training Pipeline. Active model version: v20250524163939 ---

```

We can see that my script automatically connected to my W&B account via an API key I set up in `.env` file in the root of the project directory, and after training it dumped all the metrics as a separate run in this newly created W&B project.

Also, you may wonder how it calculated the metrics if all of the data in `p300-processed-features` bucket went for training – internally, I set a hard-coded train-validation ratio of 80%, so 20% of the epochs (rows in the dataset) is used for calculating metrics for current training run.

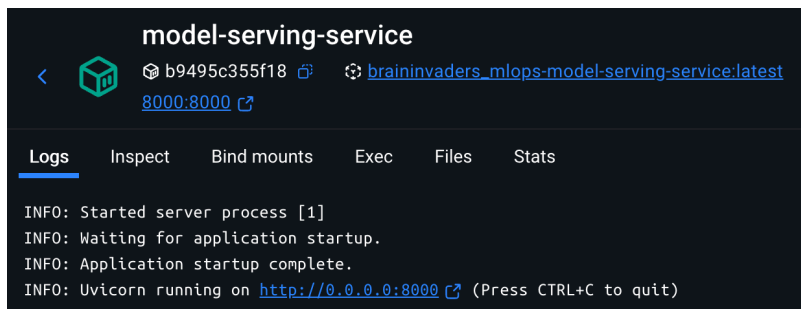
After training, we see that data transforming scikit-learn scaler object was serialized along with the LDA model. The `.txt`-file simply contains a string that points to active model identified under some name (a directory name).



We could see that `model-serving-service` runs continuously – I set it up like that, so it acts as a model API host to make inferences when called and invoked by another service – `application-simulator`.

To apply changes to this server (it started up with no model, because at the start we didn't have any model), I need to restart it so that it catches up to latest model.

Before:



After:

```
user@MacBook-Pro braininvaders_mlops % docker compose restart model-serving-service
[+] Restarting 1/1
✓ Container model-serving-service Started
```

model-serving-service

b9495c355f18 braininvaders_mlops-model-serving-service:latest 8000:8000

Logs Inspect Bind mounts Exec Files Stats

INFO: Uvicorn running on <http://0.0.0.0:8000> (Press CTRL+C to quit)
INFO: Shutting down
INFO: Waiting for application shutdown.
INFO: Application shutdown complete.
INFO: Finished server process [1]
FastAPI application startup...
Scikit-learn version in container: 1.6.1
Attempting to load active model from MinIO...
Found active model version pointer: 'v20250524163939'
Downloading model from s3://p300-models/v20250524163939/model.joblib
Model downloaded and loaded successfully.
Downloading scaler from s3://p300-models/v20250524163939/scaler.joblib
Scaler downloaded and loaded successfully.
Successfully loaded model version: v20250524163939
Loaded model classes_: [1 2]
INFO: Started server process [1]
INFO: Waiting for application startup.
INFO: Application startup complete.
INFO: Uvicorn running on <http://0.0.0.0:8000> (Press CTRL+C to quit)

Making inference

Before proceeding to this section, I ran the DAG once again to aggregate 5 additional subjects in `p300-processed-features` bucket for unbiased prediction.

To make an inference(s), I need to run this line from a code editor terminal (from local machine, not from Docker Desktop terminal!)

```
python application-simulator/application_simulator.py 11 12 13 14 15
```

Although I do have `app-simulator-job` defined in `docker-compose.yaml`, that's for when I would like to automate it as part of an Airflow DAG.

Its primary purpose in this demo is to send requests to the API, interact with exposed ports, and print results to my terminal. If I were to run it inside a Docker container from Docker Desktop, its output would go to the container's logs, making this demo less clear.

Console output:

```
(base) user@MacBook-Pro braininvaders_mlops % python application-simulator/application_simulator.py 11 12 13 14 15
wandb: Currently logged in as: xphoenixua to https://api.wandb.ai. Use `wandb login --relogin` to force relogin
wandb: WARNING Using a boolean value for 'reinit' is deprecated. Use 'return_previous' or 'finish_previous' instead.
wandb: Tracking run with wandb version 0.19.11
wandb: Run data is saved locally in /Users/user/braininvaders_mlops/wandb/run-20250524_201552-tg0c6wfv
wandb: Run `wandb offline` to turn off syncing.
wandb: Syncing run inference_batch_20250524201551
wandb: ⭐ View project at https://wandb.ai/xphoenixua/p300-bci-mlops
wandb: 🔗 View run at https://wandb.ai/xphoenixua/p300-bci-mlops/runs/tg0c6wfv
--- Simulating predictions for Batch of Subjects: [11, 12, 13, 14, 15] ---

Processing Subject 11...
Downloading processed data for Subject 11 from s3://p300-processed-features...
Downloaded data shapes: Features=(178, 1120), Labels=(178,)
S11 Metrics: Acc=0.798, Kap=0.104, AUC=0.664

Processing Subject 12...
Downloading processed data for Subject 12 from s3://p300-processed-features...
Downloaded data shapes: Features=(490, 1120), Labels=(490,)
S12 Metrics: Acc=0.735, Kap=0.040, AUC=0.522

Processing Subject 13...
Downloading processed data for Subject 13 from s3://p300-processed-features...
Downloaded data shapes: Features=(75, 1120), Labels=(75,)
S13 Metrics: Acc=0.680, Kap=0.082, AUC=0.695

Processing Subject 14...
Downloading processed data for Subject 14 from s3://p300-processed-features...
Downloaded data shapes: Features=(140, 1120), Labels=(140,)
S14 Metrics: Acc=0.914, Kap=0.677, AUC=0.963

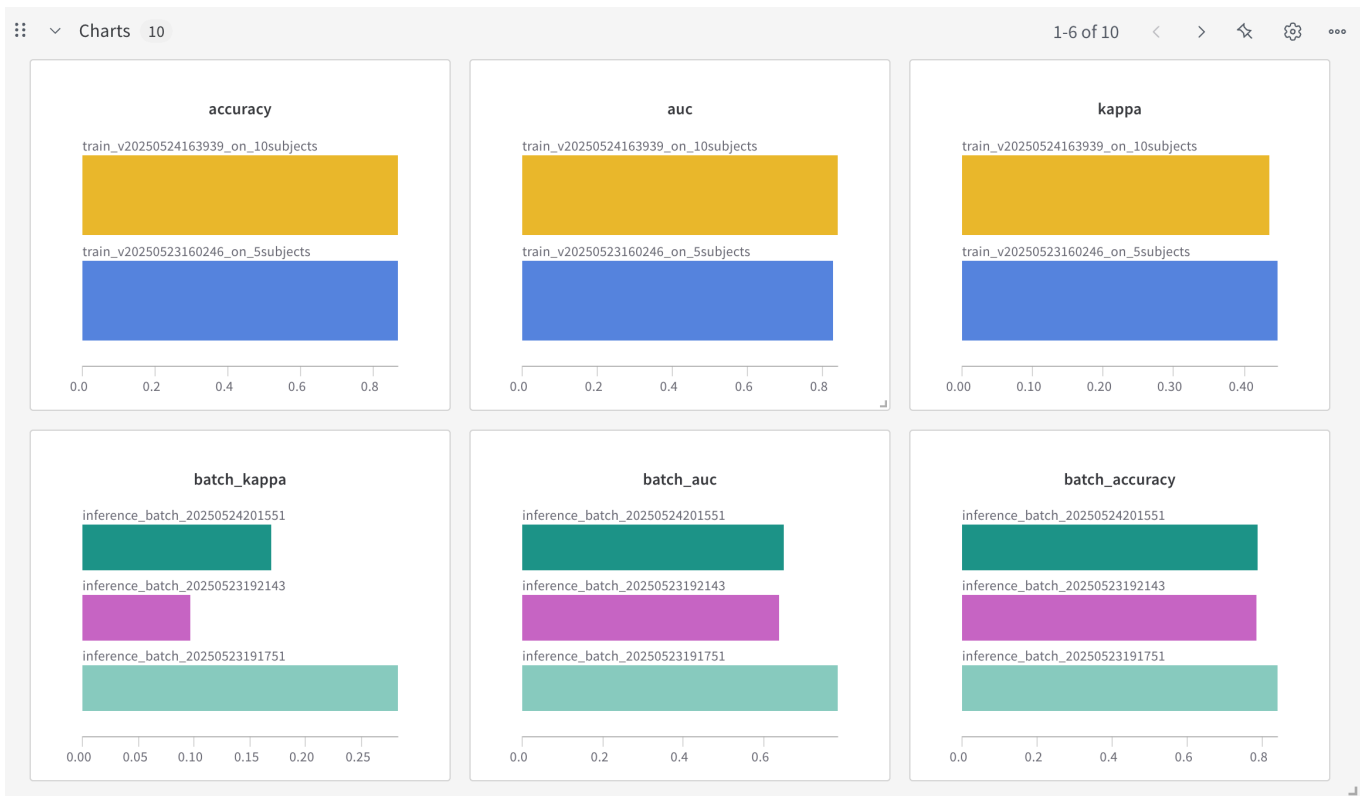
Processing Subject 15...
Downloading processed data for Subject 15 from s3://p300-processed-features...
Downloaded data shapes: Features=(210, 1120), Labels=(210,)
S15 Metrics: Acc=0.848, Kap=0.267, AUC=0.720

--- Aggregated Batch Metrics ---
Overall Accuracy: 0.7859
Overall Kappa: 0.1690
Overall AUC: 0.6479

wandb:
wandb:
wandb: Run history:
wandb: batch_accuracy _
wandb: batch_auc _
wandb: batch_kappa _
wandb:
wandb: Run summary:
wandb: batch_accuracy 0.78591
wandb: batch_auc 0.64792
wandb: batch_kappa 0.16899
wandb:
wandb: 🔗 View run inference_batch_20250524201551 at: https://wandb.ai/xphoenixua/p300-bci-mlops/runs/tg0c6wfv
wandb: ⭐ View project at: https://wandb.ai/xphoenixua/p300-bci-mlops
wandb: Synced 5 W&B file(s), 1 media file(s), 2 artifact file(s) and 0 other file(s)
wandb: Find logs at: ./wandb/run-20250524_201552-tg0c6wfv/logs
--- Finished batch simulation for Subjects: [11, 12, 13, 14, 15] ---
```

Monitoring metrics

When I go to my Weights&Biases project, I will see all the logged results from training and inference runs. For example, (some runs are not from this report's demonstration)



Tables 1

```
runs.summary["per_subject_inference_metrics_table"]
```

Filter

	SubjectID	Accuracy	Kappa	AUC	TN	FP	FN	TP
9	14	0.8643	0.3916	0.8581	113	3	16	8
10	15	0.8143	0.064	0.7221	168	7	32	3
11	6	0.862	0.4433	0.8718	325	18	39	31
12	7	0.8287	0.2082	0.7297	255	11	44	11
13	8	0.8348	0.3129	0.8025	259	25	30	19
14	9	0.8384	0.205	0.7576	725	23	122	27
	10	0.7794	0.3071	0.7594	47	8	7	6

9 - 15 of 15

Export as CSV Columns... Reset table