

# Model Monitoring

*MLflow, Tensorboard, Weights & Biases*

Credit to TA.Cheetah & TA.Phu

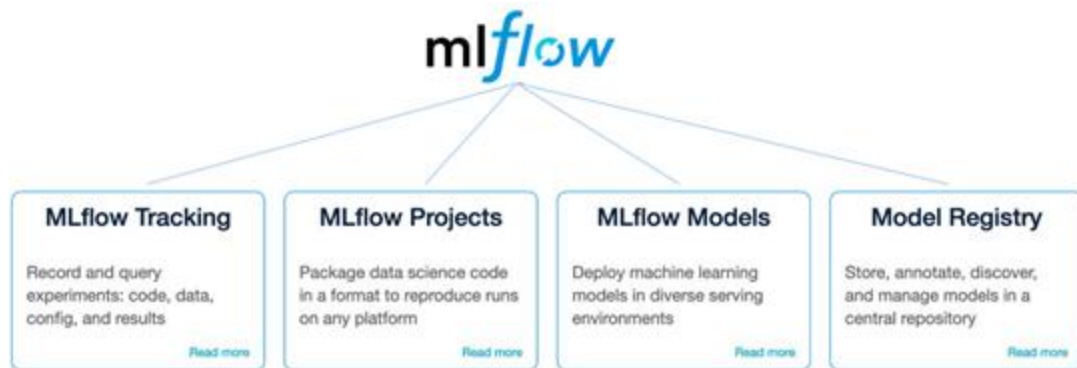
# Outline

- Mlflow
- Tensorboard
- Weights & Biases

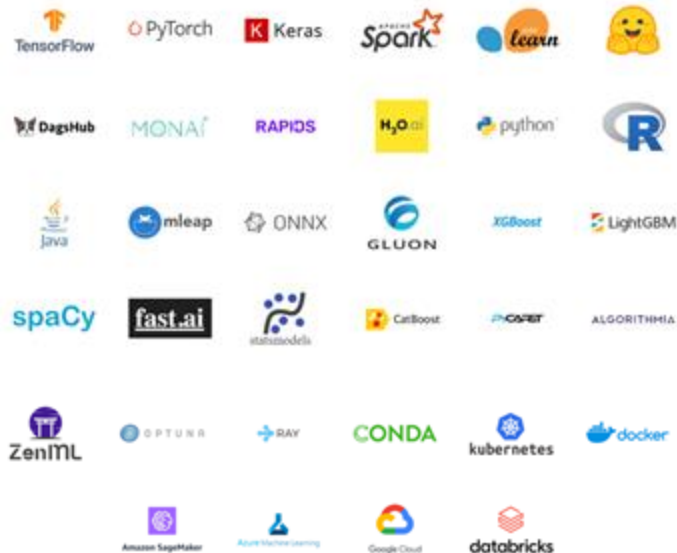
# MLflow

# What is MLflow?

- MLflow makes it simple to construct end-to-end Machine Learning pipelines in production, and this article will teach you all you need to know about the platform. This implies that at the conclusion of this tutorial, you'll be able to utilize MLflow for Machine Learning pipelines from model experimentation through model deployment.



Integrations with:



# How to run MLflow

- Install mlflow

```
!pip install mlflow --quiet
```

Python

- Import libraries

```
# Importing all Libraries
import mlflow
import mlflow.sklearn

import numpy as np
import pandas as pd
from sklearn.tree import DecisionTreeClassifier
from sklearn.datasets import load_breast_cancer
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score, classification_report
```

Python

- Load dataset and define evaluation metrics

```
# Load and split dataset
X, Y = load_breast_cancer(return_X_y=True)
X_train, X_test, y_train, y_test = train_test_split(X, Y, test_size=0.2, random_state=0)
print("Training Data Shape: ", X_train.shape, y_train.shape)
print("Testing Data Shape: ", X_test.shape, y_test.shape)

def eval_metrics(actual, pred):
    accuracy = accuracy_score(actual, pred)
    return accuracy
```

# Model tracking

1. Start an experiment using **mlflow.start\_run()** which switches the context of your existing model code to enable mlflow tracking.
2. We log the run parameters with **mlflow.log\_param()**
3. We log the model metrics (mean accuracy on the training set in this case) with **mlflow.log\_metric()**.
4. After model training and evaluation, I have logged the model using **mlflow.sklearn.log\_model()**.
5. The model signature passed to the **mlflow.sklearn.log\_model()**. ensures the schema of the model's input and output is accurately documented.

```
def train_model(criterion, max_depth):  
  
    # Starting the Experiment  
    with mlflow.start_run():  
  
        # Model building  
        model = DecisionTreeClassifier(criterion=criterion, max_depth=max_depth, random_state=0)  
        model.fit(X_train, y_train) # Model Training  
        y_pred = model.predict(X_test) # Model Prediction on Testing data  
        (accuracy) = eval_metrics(y_test, y_pred)  
  
        print('Decision tree (criterion=%s, max_depth=%d):'%(criterion, max_depth))  
        print('Accuracy: {:.4f}'.format(accuracy))  
  
        # Logging Parameters  
        mlflow.log_param("criterion", criterion)  
        mlflow.log_param("max_depth", max_depth)  
  
        # Logging Metrics  
        mlflow.log_metric("accuracy", accuracy_score(y_test, y_pred))  
  
        # Model Logging  
        signature = infer_signature(X_train, model.predict(X_train))  
        mlflow.sklearn.log_model(  
            model,  
            "model",  
            signature=signature,  
        )  
  
    return model
```

# Train model and search best 5 runs

- Train 10 models with different hyperparameters

```
param_grid = {  
    'criterion': ['gini', 'entropy'],  
    'max_depth': [1, 2, 3, 4, 5],  
}  
  
all_combinations = itertools.product(  
    param_grid['criterion'],  
    param_grid['max_depth'],  
)  
  
for combination in all_combinations:  
    criterion, max_depth = combination  
    train_model(criterion, max_depth)
```

# Load best model (MLflow models)

artifact_uri	start_time	end_time	metrics.accuracy	params.max_depth	params.criterion	tags.mlflow.sour
/mlruns/1/5d63abf908a643ac93724bd076496e19/ar...	2023-08-23 01:47:10.328000+00:00	2023-08-23 01:47:10.584000+00:00	0.964912	3	gini	/usr/local/lib/python3 packages
/mlruns/1/d845b4259e634b5cb8485012cc6bc01b/ar...	2023-08-23 01:47:10.014000+00:00	2023-08-23 01:47:10.263000+00:00	0.964912	2	gini	/usr/local/lib/python3 packages
/mlruns/1/4746bcfe0a5b4936b3350514515e0536/ar...	2023-08-23 01:47:10.639000+00:00	2023-08-23 01:47:10.961000+00:00	0.956140	4	gini	/usr/local/lib/python3 packages
/mlruns/1/09ad160ad4734e68a6044e59228c805a/ar...	2023-08-23 01:47:11.931000+00:00	2023-08-23 01:47:12.131000+00:00	0.947368	3	entropy	/usr/local/lib/python3 packages
/mlruns/1/d16fe0518d2d4bff80f343f9a58be6b4/ar...	2023-08-23 01:47:11.200000+00:00	2023-08-23 01:47:11.406000+00:00	0.947368	5	gini	/usr/local/lib/python3 packages

```
# Load model as a PyFuncModel.
loaded_model = mlflow.pyfunc.load_model(model_uri=f"runs:/{run_id}/model") # run_id of best model

# Predict on a Pandas DataFrame.
predicted = loaded_model.predict(pd.DataFrame(X_test))
print(classification_report(y_test, predicted, target_names=['Non-DD', 'DD'], digits=4))
```

Python

	precision	recall	f1-score	support
Non-DD	0.9778	0.9362	0.9565	47
DD	0.9565	0.9851	0.9706	67
accuracy			0.9649	114
macro avg	0.9671	0.9606	0.9636	114
weighted avg	0.9653	0.9649	0.9648	114



# Model registry

- The MLflow Model Registry component is a centralized model store, set of APIs, and UI, to collaboratively manage the full lifecycle of an MLflow Model. It provides model lineage, model versioning, stage transitions (for example from staging to production), and annotations.

```
#Register best model
mlflow.register_model(model_uri=model_uri, name="breast_cancer")
```

Python

Successfully registered model 'breast\_cancer'.  
2023/08/23 14:12:33 INFO mlflow.tracking.\_model\_registry.client: Waiting up to 300 seconds for model version to finish creation. Model name: breast\_cancer, Created version '1' of model 'breast\_cancer'.

- Load model from registered model

```
model_name = "breast_cancer"
model_version = 1
# Load model as a PyFuncModel.
loaded_model = mlflow.pyfunc.load_model(model_uri=f"models:/{model_name}/{model_version}")

# Predict on a Pandas DataFrame.
predicted = loaded_model.predict(pd.DataFrame(X_test))
```

Python

```
print(classification_report(y_test, predicted, target_names=['Non-DO', 'DO'], digits=4))
```

Python

	precision	recall	f1-score	support
Non-DO	0.9778	0.9362	0.9565	47
DO	0.9565	0.9851	0.9706	67
accuracy			0.9649	114
macro avg	0.9671	0.9606	0.9636	114
weighted avg	0.9653	0.9649	0.9648	114

# MLflow UI

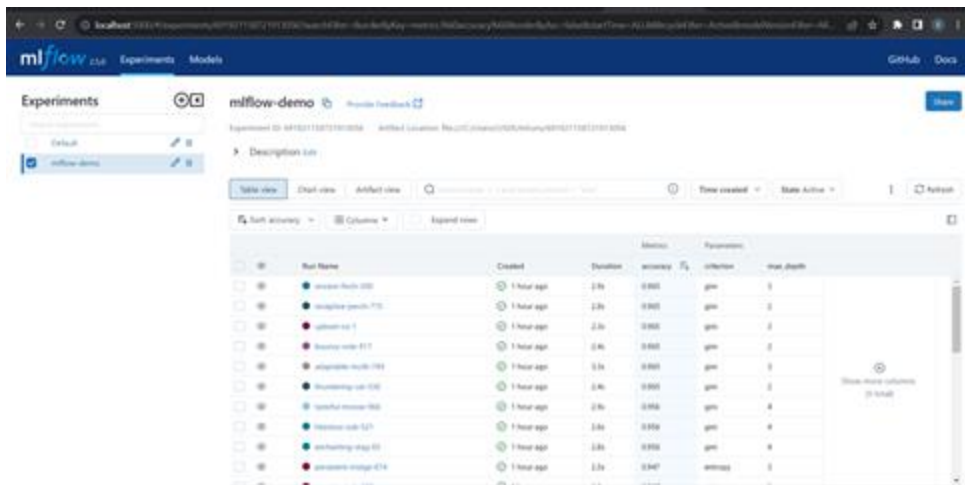
- View MLflow runs and experiments

```
!mlflow ui
# Access this link: http://localhost:5000/
```

Python

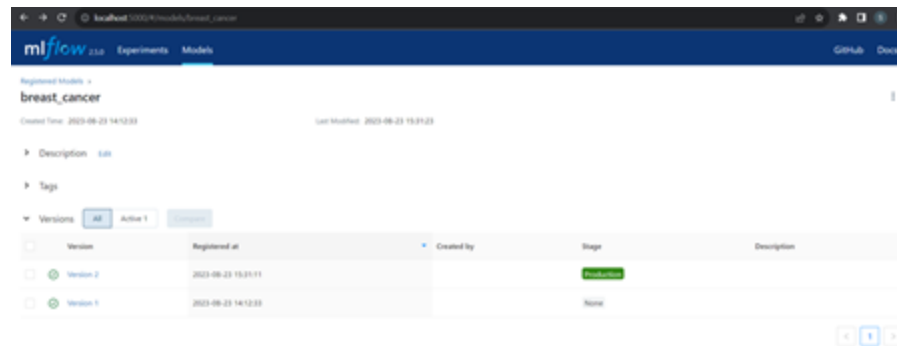
Compare performance

Registered model



The screenshot shows the MLflow UI 'Experiments' page. The 'mlflow-demo' experiment is selected. Below the experiment name, there are tabs for 'Table view', 'Chart view', and 'Artifact view'. The 'Table view' is active, displaying a table of runs. The table has columns for Run Name, Created, Duration, Accuracy, Criterion, and Max Depth. There are 10 runs listed, each with a status icon and a 'Run' button. A 'Show more columns' button is visible at the bottom right of the table.

Run Name	Created	Duration	Accuracy	Criterion	Max Depth
mlflow-demo-000	1 hour ago	2.5s	0.800	gini	5
mlflow-demo-001	1 hour ago	2.5s	0.800	gini	5
mlflow-demo-002	1 hour ago	2.5s	0.800	gini	5
mlflow-demo-003	1 hour ago	2.5s	0.800	gini	5
mlflow-demo-004	1 hour ago	2.5s	0.800	gini	5
mlflow-demo-005	1 hour ago	2.5s	0.800	gini	5
mlflow-demo-006	1 hour ago	2.5s	0.800	gini	5
mlflow-demo-007	1 hour ago	2.5s	0.800	gini	5
mlflow-demo-008	1 hour ago	2.5s	0.800	gini	5
mlflow-demo-009	1 hour ago	2.5s	0.800	gini	5



The screenshot shows the MLflow UI 'Registered Models' page. The 'breast\_cancer' model is selected. Below the model name, there are tabs for 'Description', 'Tags', and 'Versions'. The 'Versions' tab is active, displaying a table of model versions. The table has columns for Version, Registered at, Created by, Stage, and Description. There are 3 versions listed, each with a status icon and a 'Compare' button.

Version	Registered at	Created by	Stage	Description
Version 1	2023-08-23 14:12:33		Production	
Version 2	2023-08-23 14:12:33		None	
Version 3	2023-08-23 14:12:33		None	

# For run mlflow ui on google colab

```
# Load and split dataset
X, Y = load_breast_cancer(return_X_y=True)
X_train, X_test, y_train, y_test = train_test_split(X, Y, test_size=0.2, random_state=0)
print("Training Data Shape: ", X_train.shape, y_train.shape)
print("Testing Data Shape: ", X_test.shape, y_test.shape)
```

```
local_registry = "sqlite:///mlruns.db"
mlflow.set_tracking_uri(local_registry)
experiment_id = mlflow.set_experiment('test_experiment')
```

add 3 lines before with `mlflow.start_run():`

```
def eval_metrics(actual, pred):
    accuracy = accuracy_score(actual, pred)
    return accuracy
```

Get authtoken from <https://dashboard.ngrok.com/get-started/your-authtoken>

```
!pip install pyngrok --quiet
```

Python

```
from pyngrok import ngrok
ngrok.kill()

#Setting the authtoken (optional)
#Get your authtoken from https://dashboard.ngrok.com/auth
NGROK_AUTH_TOKEN = '' # Your authtoken
ngrok.set_auth_token(NGROK_AUTH_TOKEN)

# Open an HTTPS tunnel on port 5000 for http://localhost:5000
ngrok_tunnel = ngrok.connect(addr='5000', proto='http', bind_tls=True)
print("MLflow Tracking UI: ", ngrok_tunnel.public_url)
```

Python

```
WARNING [pyngrok.process.pyngrok] 2023-08-23T01:47:21.000000Z: "ngrok config file found at legacy location, move to XDG location" xdg_path=/root/.con
MLflow Tracking UI: https://79c5-34-136-157-242.ngrok-free.app
```

access from this link

```
!mlflow ui --backend-store-uri sqlite:///mlruns.db
```

Python

# NGROK Authtoken

**Getting Started**

- Setup & Installation
- Your Authtoken**

**Cloud Edge**



- Endpoints
- Edges
- Domains
- TCP Addresses
- App Users

**Tunnels**

- Agents
- Authtokens
- SSH Public Keys
- Ingress


## Your Authtoken

This is your personal Authtoken. Use this to authenticate the ngrok agent that you downloaded.


.....   Copy

### Command Line

Authenticate your ngrok agent. You only have to do this once. The Authtoken is saved in the default configuration file.

 Command Line Show Authtoken

```
ngrok config add-authtoken $YOUR_AUTHTOKEN
```

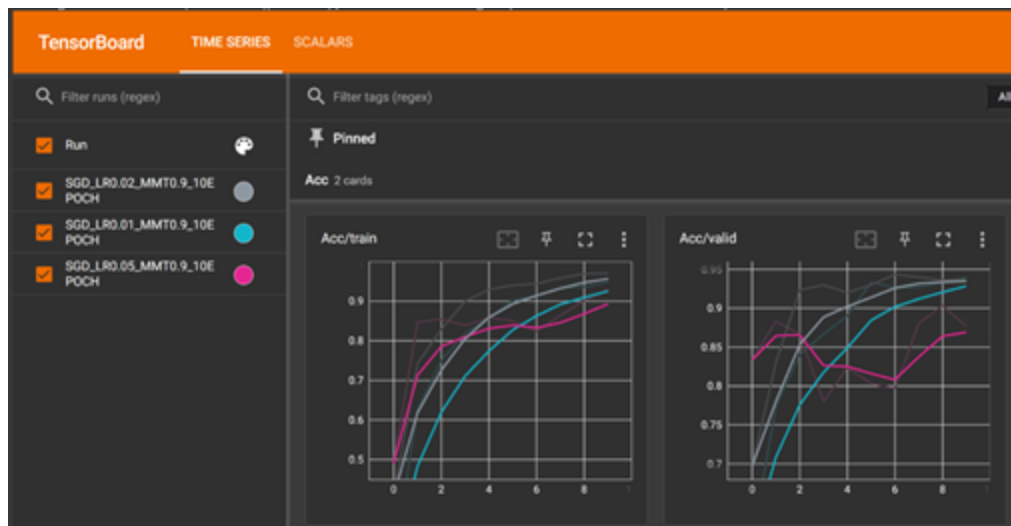


# Tensorboard

## Guide

# What is tensorboard

- 1) Visualization toolkit for machine learning training
- 2) Can visualize train/validate loss, accuracy etc.
- 3) Benefits in comparing between runs (adjust hyperparameters)



# Tensorboard steps

- 1) install `pip install -qq tensorboard`
- 2) import summarywriter  
`from torch.utils.tensorboard import SummaryWriter`
- 1) create directory to save log files e.g. `/content/runs/run1/`
- 2) instantiate writer `writer = SummaryWriter(log_dir="./runs/run1/")`
- 3) add scalar `writer.add_scalar("Name", value, round)`
- 4) write on disk `writer.flush()`
- 5) close `writer.close()`
- 6) launch tensorboard  
`%load_ext tensorboard`  
`%tensorboard --logdir runs`

# References

[1] <https://www.tensorflow.org/tensorboard>

[2] [https://pytorch.org/tutorials/recipes/recipes/tensorboard\\_with\\_pytorch.html](https://pytorch.org/tutorials/recipes/recipes/tensorboard_with_pytorch.html)



# WandB

## Guide

# What is WandB

- Special tools by Weights & Biases for
  - *experiments tracking*
  - results visualization
  - *hyperparameter adjustment (sweep)*
  - reproduce models
  - and more!
- Create account <https://wandb.ai/site>
- Get API key (Need when login) <https://wandb.ai/authorize>

# Steps: Dashboard

- 1) install `!pip install wandb`
- 2) import `import wandb`
- 3) login `wandb.login()` *# this one is for the imported wandb library*
- 4) initiate 

```
wandb.init(  
    project="Animal-EfficientNetB0",  
    config={"learning_rate": 0.02,  
           "architecture": "EfficientNetB0",  
           "dataset": "Animal2",  
           "epochs": 10}  
)
```
- 5) log `wandb.log({"acc": acc, "loss": loss})`
- 6) finish `wandb.finish()`

# Steps: Sweep

1) install

```
!pip install wandb
```

2) import

```
import wandb
```

3) login

```
wandb.login()
```

4) create config (dict)

5) write your own training function

6) write WandB training function on top

7) initiate sweep (via wandb agent)

8) get results at your account page

```
sweep_config = dict()
```

```
train()
```

```
trainer()
```

```
wandb.agent(sweep_id, train)
```

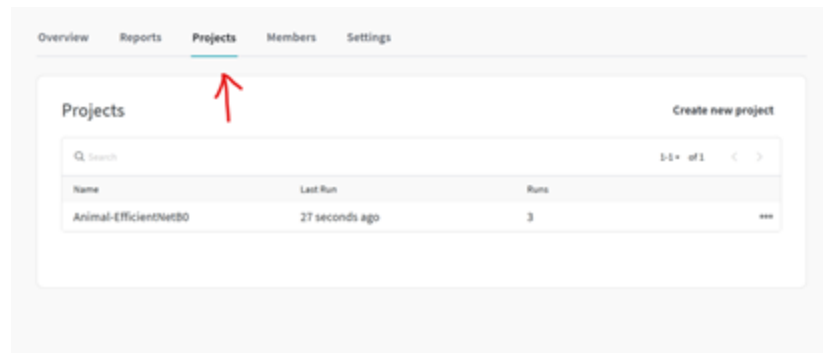
<https://wandb.ai/>

# Results

- Run history and run summary in your notebook



- Full dashboard in your wandb profile



Overview

Workspace

Runs

Jobs

Automat.

Sweeps

Reports

Artifacts

Runs (2)

Search

Filter

👁 Name (2 visualized)

👁 ● visionary-mountain-2

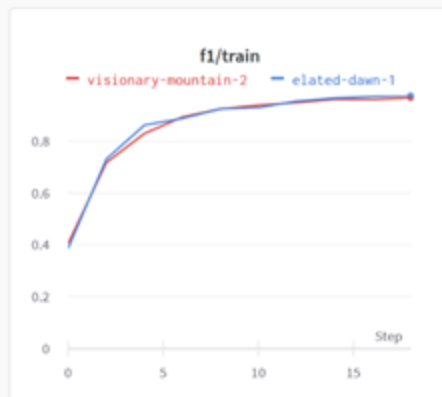
👁 ● elated-dawn-1

1-2 of 2 < >

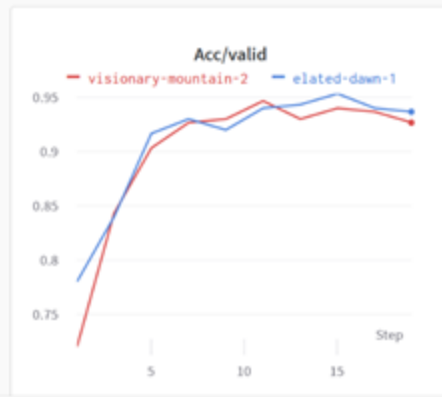
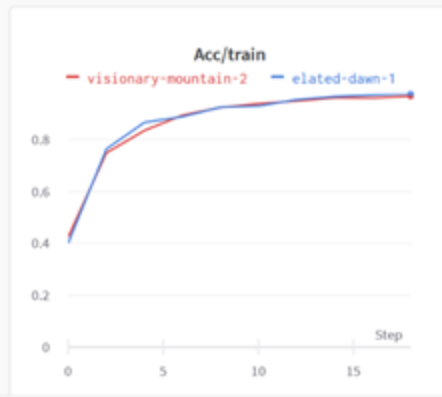
Search panels with regex



Create report



Acc 2



Add panel

# Check this out!

## Weave, a toolkit designed by Weights & Biases for tracking and evaluating LLM applications

The screenshot shows the Weave web interface. The top navigation bar includes 'Home', 'My projects', 'Applications', 'Profile', 'Teams', and 'Activity across your organization'. The main content area is titled 'Building LLM apps?' and features a 'Try it now' button. Below this, there's a section for 'Your recent projects' with three cards: 'uncategorized', 'RoBERTa Sentiment', and 'Animal-EfficientNet80'. Each card shows the last updated time and the number of runs. A 'New project' button is also present. The bottom section, 'You don't have any recent reports', includes a 'Create your first report for uncategorized' button and a link to 'Click to see how your charts from uncategorized look in a rep...'. The 'Activity across your organization' section shows a list of recent runs, including 'lemon-sweep-6' and 'skilled-sweep-5', both marked as 'Finished'.

The screenshot shows the Weave documentation page for the 'Introduction'. The left sidebar contains a navigation menu with sections like 'Getting Started', 'Introduction', 'Trace LLMs', 'Trace Applications', 'App versioning', 'Build an Evaluation', 'Evaluate a RAG App', 'Product Walkthrough', 'LLM Application Tracing', 'Calls', 'Ops', 'Objects', 'Models', 'Datasets', 'Evaluations', 'Feedback', 'Costs', 'Tools & Utilities', 'Integrations', and 'LLM Providers'. The main content area is titled 'Introduction' and describes Weave as a 'lightweight toolkit for tracking and evaluating LLM applications, built by Weights & Biases'. It states the goal is to bring rigor, best-practices, and composability to the inherently experimental process of developing AI applications. A 'Get started' section mentions decorating Python functions with `@weave.op()`. Below this, there's a 'Retriever.get\_relevant\_documents' function call with a 'Call' tab selected, showing the function's arguments and outputs. The 'Call' tab shows the function being called with a 'question' argument. The 'Summary' tab shows the function's output, which is a list of relevant documents. The 'Code' tab shows the function's implementation. The 'Outputs' tab shows the function's output, which is a list of relevant documents. The bottom of the page includes a 'Seriously, try the quickstart or Open in Colab' link.

# References

[1] <https://wandb.ai/home>