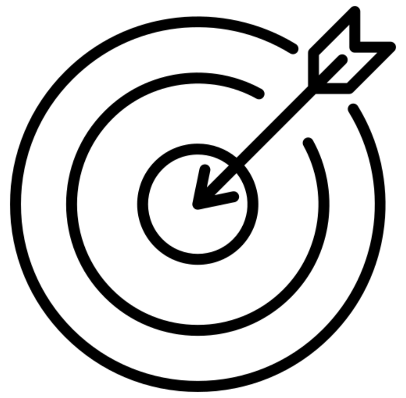
Monitoring ML experiments with Weights & Biases

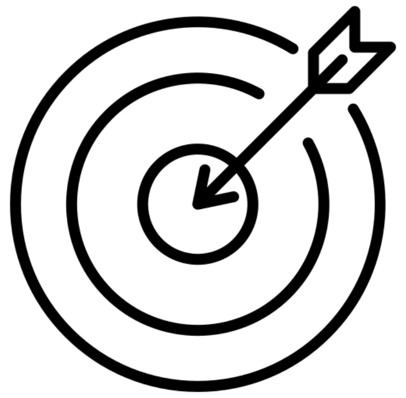
Tutorial 3

Sungho Bae

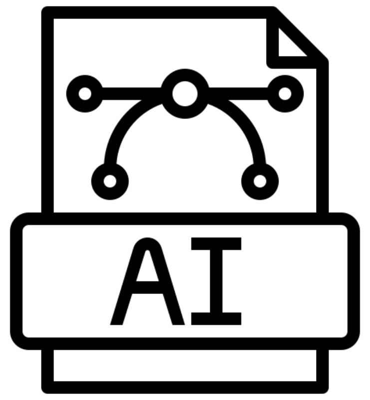


Goal

Intro: Motivation



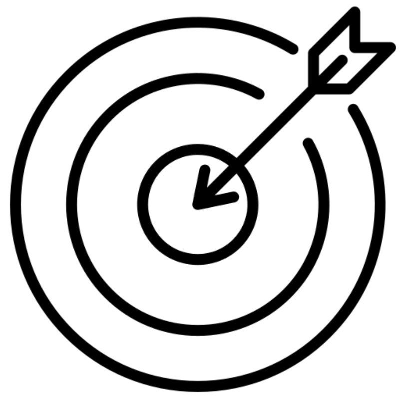
Goal



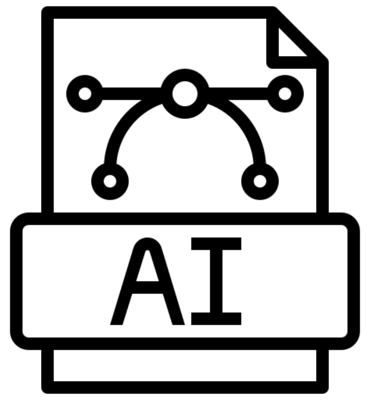
Model

Intro: Motivation

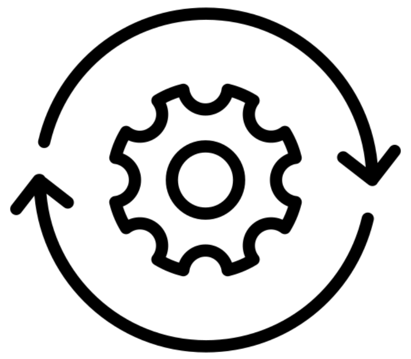
Q. Have you ever done **train/valid metric monitoring** in Jupyter Notebook or colab as follows?



Goal



Model

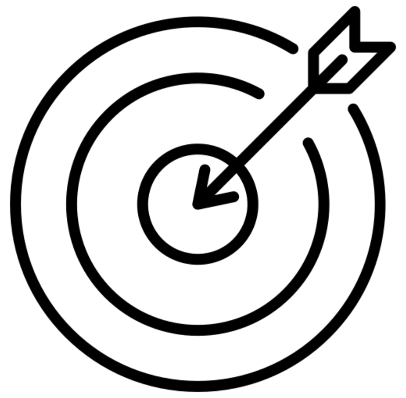


Experiments

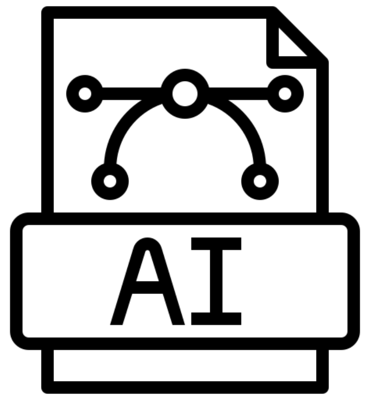
a lots of,,,,,,

```
Train on 2395 samples, validate on 295 samples
Epoch 1/500
2395/2395 [==============================] - 0s 132us/step - loss: 0.0559 - val_loss: 0.0151
Epoch 2/500
2395/2395 [==============================] - 0s 67us/step - loss: 0.0101 - val_loss: 0.0095
Epoch 3/500
2395/2395 [==============================] - 0s 66us/step - loss: 0.0071 - val_loss: 0.0067
Epoch 4/500
2395/2395 [==============================] - 0s 68us/step - loss: 0.0060 - val_loss: 0.0064
Epoch 5/500
2395/2395 [==============================] - 0s 64us/step - loss: 0.0057 - val_loss: 0.0063
Epoch 6/500
2395/2395 [==============================] - 0s 68us/step - loss: 0.0056 - val_loss: 0.0063
```

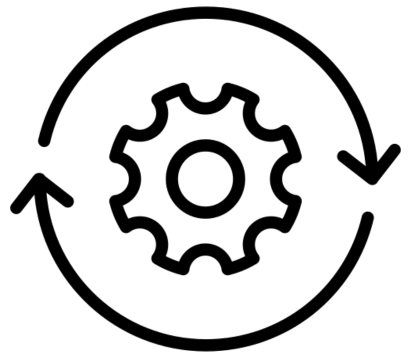
Q. Have you ever struggled to **compare and navigate the results** and experiences for each experiment?



Goal

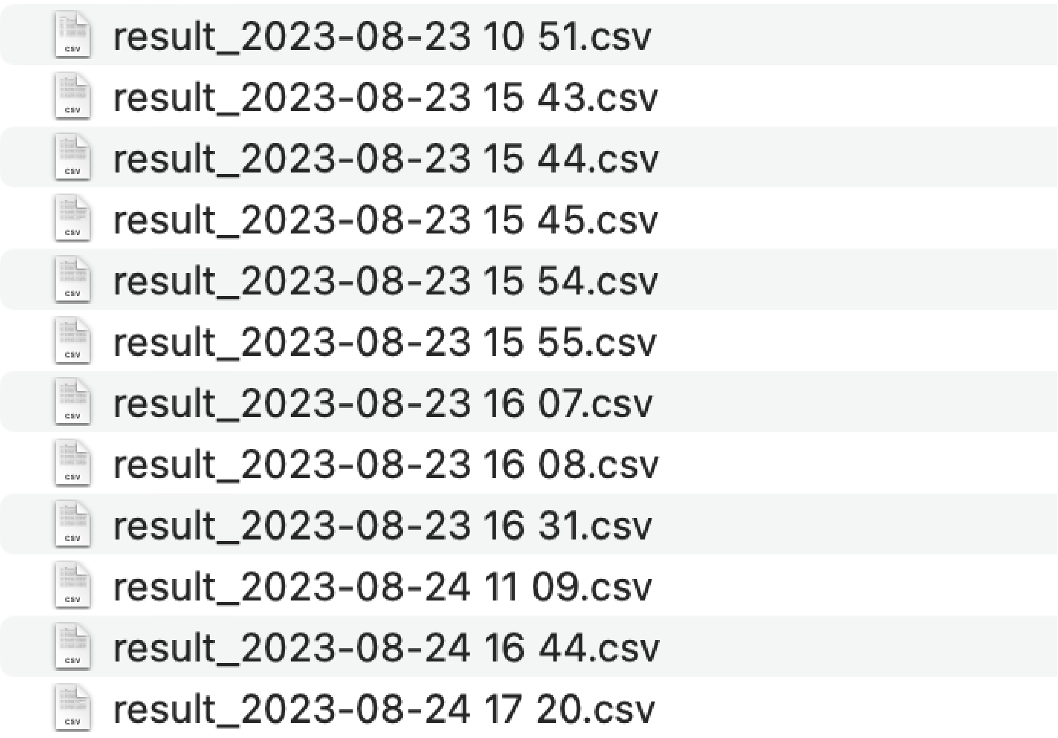


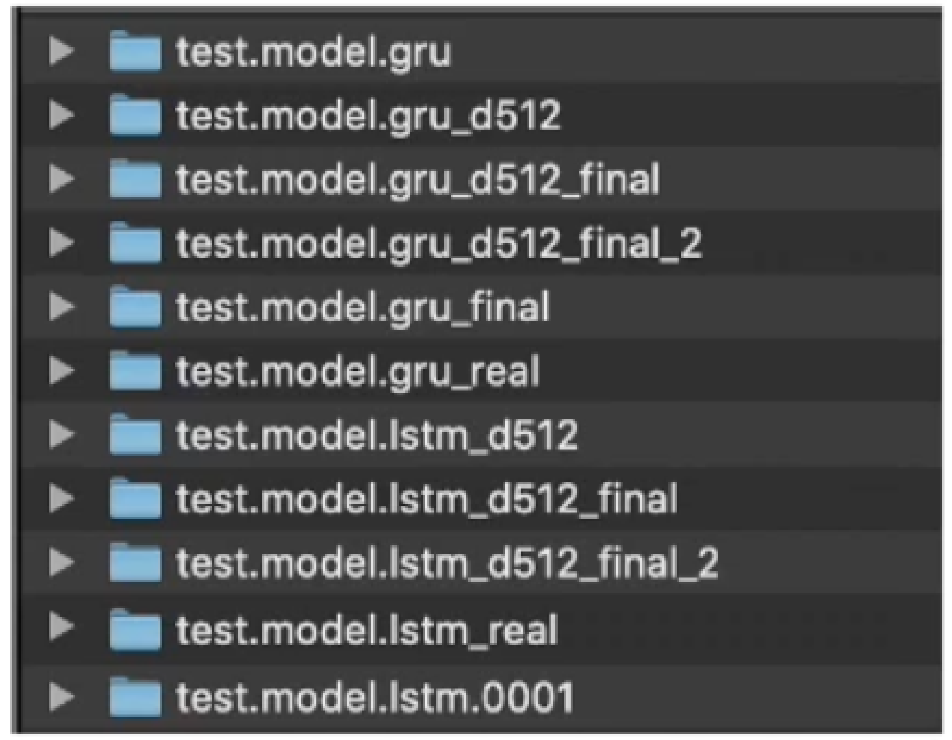
Model



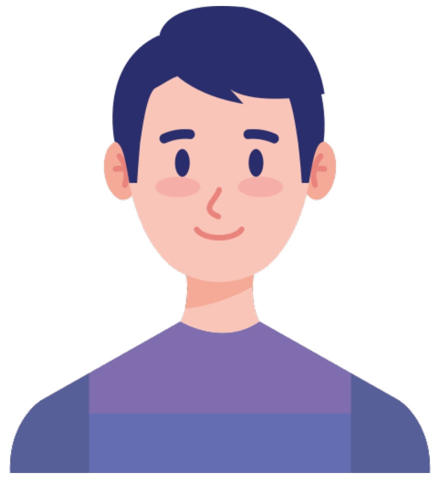
Experiments

a lots of,,,,,,





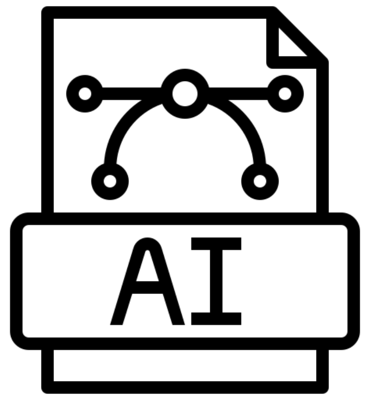
Intro: Motivation



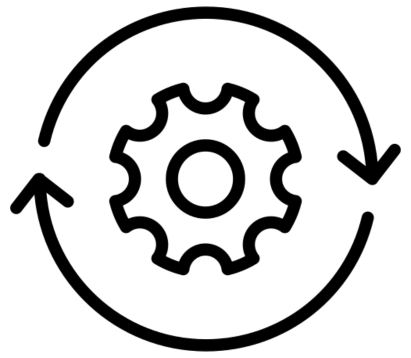
Me

Colleague

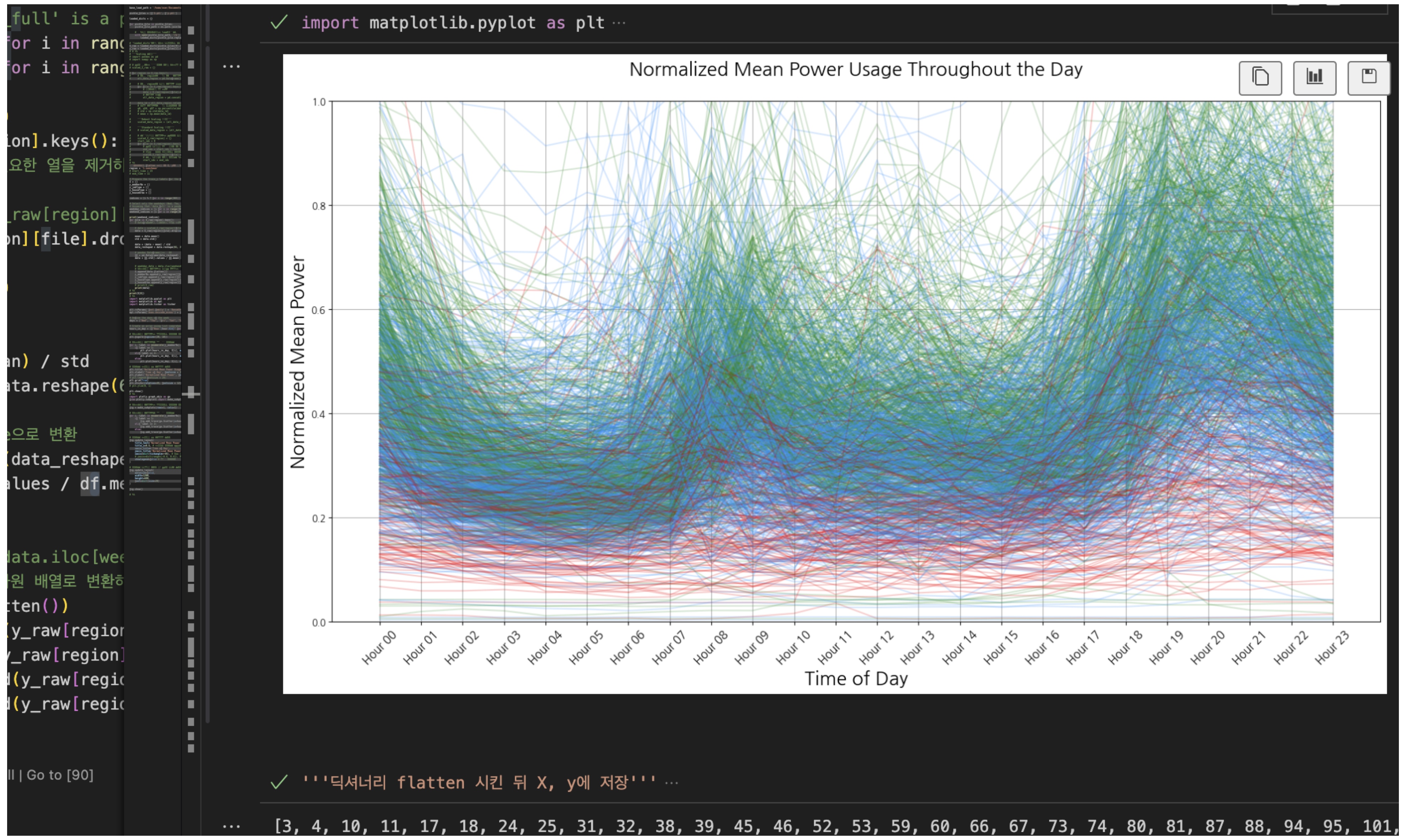
When **sharing or discussing an experiment** with a colleague,
have you ever **captured an image of the results** and sent it to them?



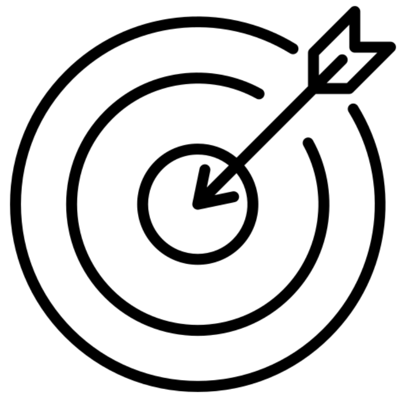
Model



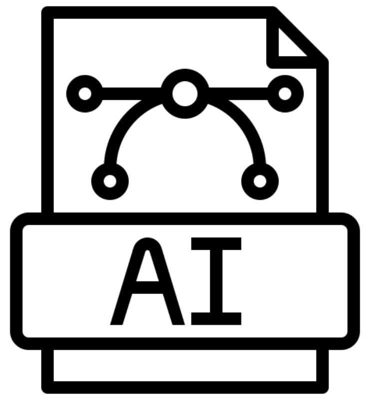
Experiments
a lots of,,,,,,



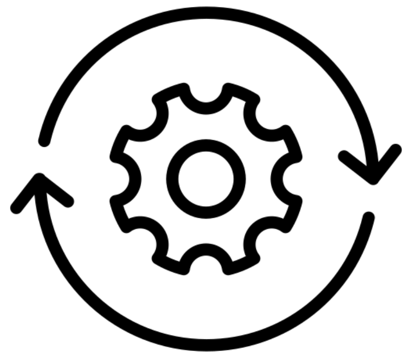
Intro: Motivation



Goal



Model

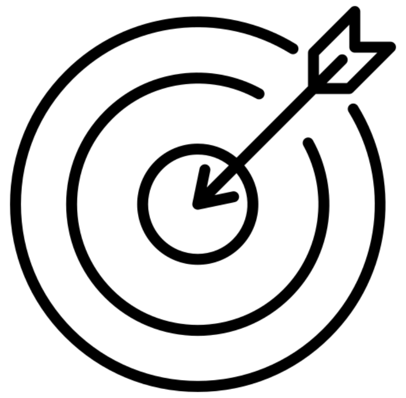


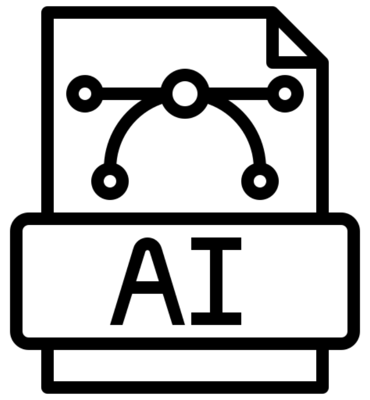
Experiments

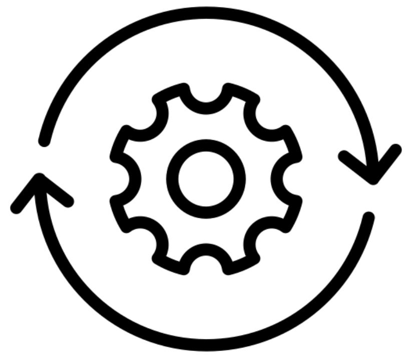
a lots of,,,,,,

Adjust **hyperparameters**, verify models and data, monitor Gradient flow and GPU systems, and more...

there's a lot to consider.







Goal

Model

Experiments

a lots of,,,,,,

Adjust **hyperparameters**, verify models and data, monitor Gradient flow and GPU systems, and more...

JUST WITH

5 LINES?

Intro: Motivation

```
import wandb

# 1. Start a New run
wandb.init(project="gpt-3")

# 2. Save model inputs and hyperparameters
config = wandb.config
config.learning_rate = 0.01

# 3. Log gradients and model parameters
wandb.watch(model)
for batch_idx, (data, target) in enumerate(train_loader):
  ...
  if batch_idx % args.log_interval == 0:
  # 4. Log metrics to visualize performance
  wandb.log({"loss": loss})
```

Those who don’t track training are doomed to repeat it.

From. Weights & Biases.

Weights & Biases

What is **Wandb**?

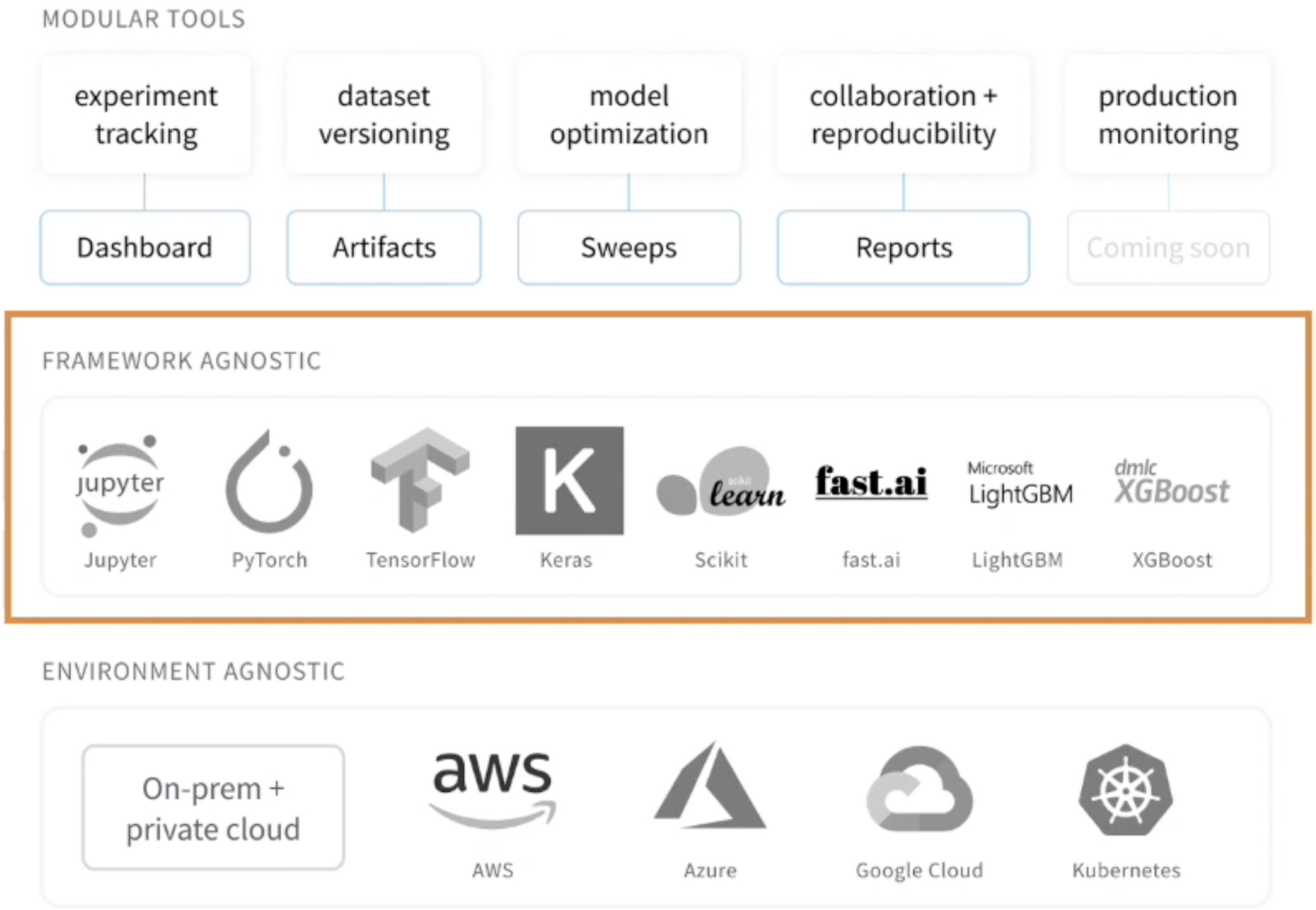
Weight & Bias?

> A tool to **easily track and visualise the progress**
>
> of your deep learning experiments

> **W (weights) & b (bias)**,
>
> commonly used in deep learning, called wandb for short

> Can be **integrated with various frameworks**
>
> such as pytorch, tensorflow, keras, huggingface, etc.

Ch1. What is Wandb?



What Wandb can do?

- Save the **hyper-parameters** used during training
- **Explore, compare, and visualise each experiment**
- Analyse systems in the learning environment
- **Collaborate** with others
- **Replicate** the results of past experiments
- Allows **hyper-parameter tuning**
- **Permanently store a record** of all your experiments

What Wandb can do?

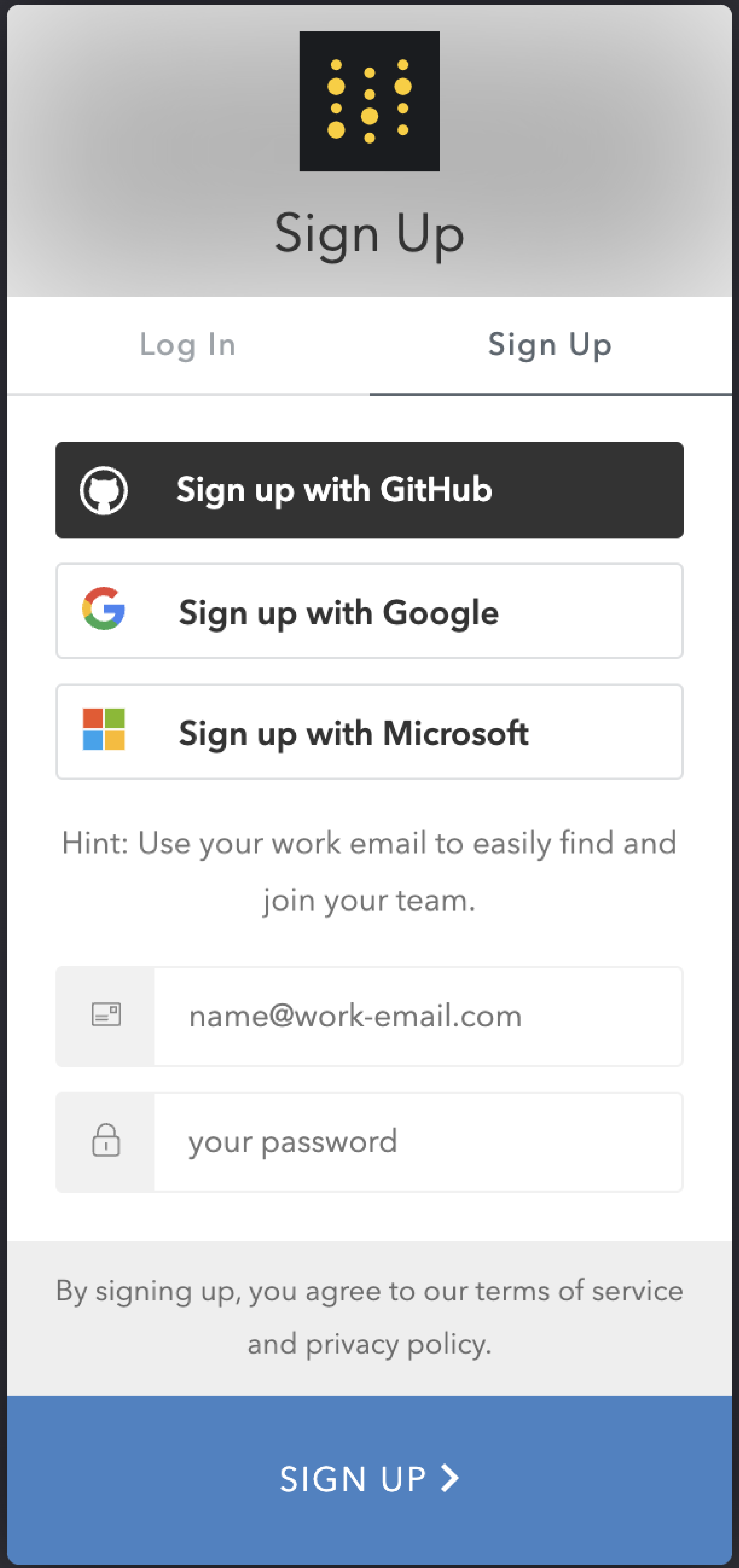
- Save the **hyper-parameters** used during training
- **Explore, compare, and visualise each experiment**
- Analyse systems in the learning environment
- **Collaborate** with others
- **Replicate** the results of past experiments
- Allows **hyper-parameter tuning**
- **Permanently store a record** of all your experiments

Weights & Biases

Wandb Tutorial

Ch2. tutorial for wandb

Sign Up & Setting



https://kr.wandb.ai/

Ch2. tutorial for wandb

Colab URL: https://colab.research.google.com/drive/1m52CXom83J6lXn1GqrDq6auFNx1ILBY-?usp=sharing

👆 Click!

Quickstart: Tracking your first run in Weights & Biases

Weights & Biases' tools make it easy for you to quickly track experiments, visualize results, spot regressions, and more. Simply put, Weights & Biases enables you to build better models faster and easily share findings with colleagues.

Visualize your model training with

Python / Pytorch or Open in Colab

1. Set up the wandb library

Install the CLI and Python library for interacting with the Weights and Biases API.

```
pip install wandb
```

Next, log in and paste your API key when prompted.

```
wandb login
```

Your API key for logging in to the wandb library: 1931cdf62bf4d4f79cd0433…

Ch2. tutorial for wandb

👟 Run an experiment

1️⃣. **Start a new run** and pass in hyperparameters to track

2️⃣. **Log metrics** from training or evaluation

3️⃣. **Visualize results** in the dashboard

```
import random

# Launch 5 simulated experiments
total_runs = 5
for run in range(total_runs):
  # 🐝 1️⃣ Start a new run to track this script
  wandb.init(
      # Set the project where this run will be logged
      project="basic-intro",
      # We pass a run name (otherwise it'll be randomly assigned, like sunshine-lollypop-10)
      name=f"experiment_{run}",
      # Track hyperparameters and run metadata
      config={
      "learning_rate": 0.02,
      "architecture": "CNN",
      "dataset": "CIFAR-100",
      "epochs": 10,
      })

  # This simple block simulates a training loop logging metrics
  epochs = 10
  offset = random.random() / 5
  for epoch in range(2, epochs):
      acc = 1 - 2 ** -epoch - random.random() / epoch - offset
      loss = 2 ** -epoch + random.random() / epoch + offset

      # 🐝 2️⃣ Log metrics from your script to W&B
      wandb.log({"acc": acc, "loss": loss})

  # Mark the run as finished
  wandb.finish()
```

doesn't actually train the model.

Just a **sample code** that simply shows

what **ACC and LOSS graph** look like at Wandb Page

```
    # 🐝 2️⃣ Log metrics from your script to W&B
    wandb.log({"acc": acc, "loss": loss})

# Mark the run as finished
wandb.finish()
```

wandb: Currently logged in as: oy6uns (oy6uns_team). Use `wandb login --relogin` to force relogin
Tracking run with wandb version 0.16.5
Run data is saved locally in `/content/wandb/run-20240401_080837-av2dpar6`
Syncing run **experiment_0** to Weights & Biases (docs)
View project at https://wandb.ai/oy6uns_team/basic-intro
View run at https://wandb.ai/oy6uns_team/basic-intro/runs/av2dpar6/workspace

Run history:

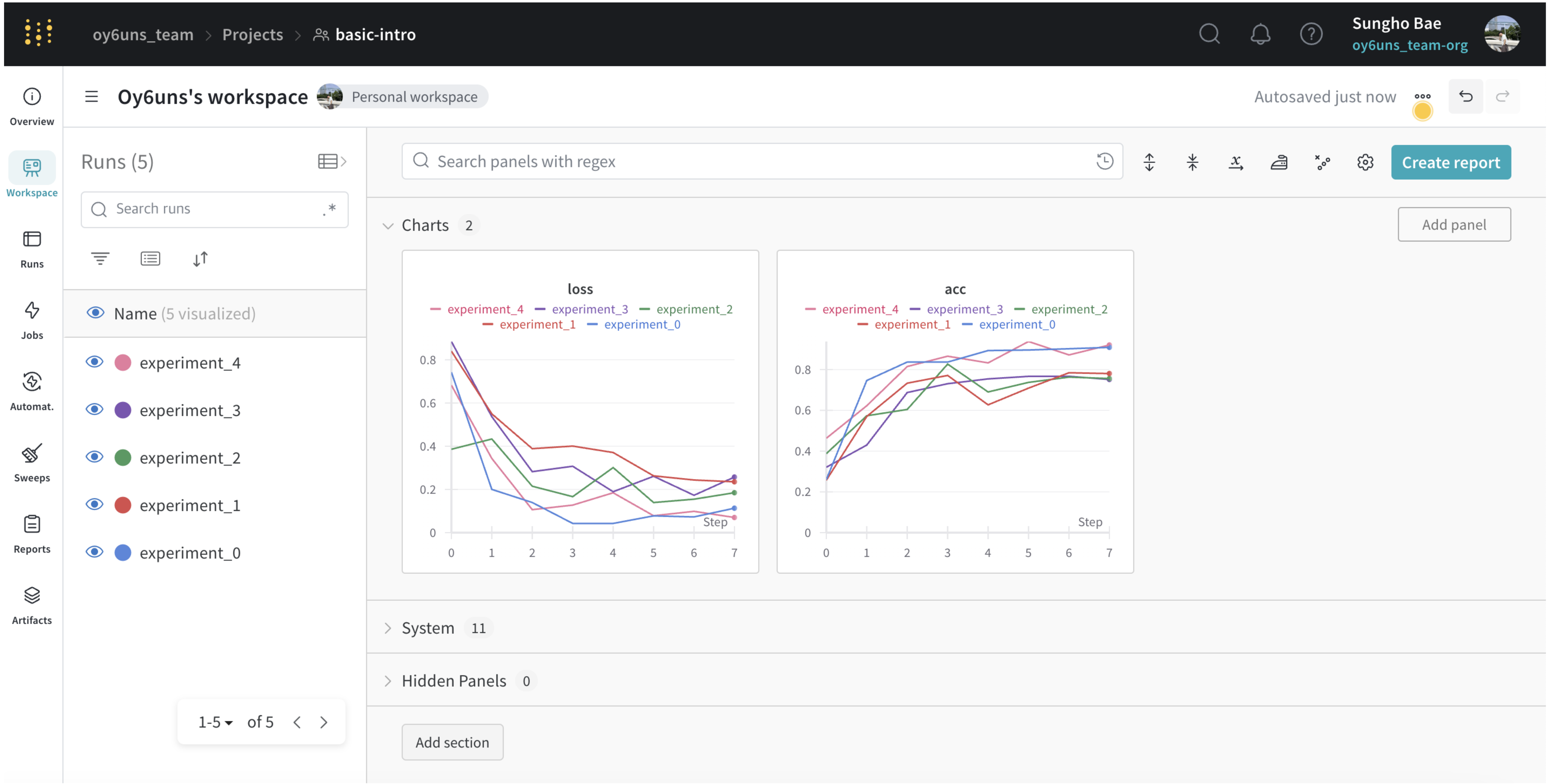
acc	
loss	

Run summary:

acc	0.90902
loss	0.11327

View run experiment_0 at: https://wandb.ai/oy6uns_team/basic-intro/runs/av2dpar6/workspace
Synced 4 W&B file(s), 0 media file(s), 0 artifact file(s) and 0 other file(s)
Find logs at: `./wandb/run-20240401_080837-av2dpar6/logs`
Tracking run with wandb version 0.16.5
Run data is saved locally in `/content/wandb/run-20240401_080842-8lyllfgy`
Syncing run **experiment_1** to Weights & Biases (docs)
View project at https://wandb.ai/oy6uns_team/basic-intro
View run at https://wandb.ai/oy6uns_team/basic-intro/runs/8lyllfgy/workspace

Ch2. tutorial for wandb



Weights & Biases

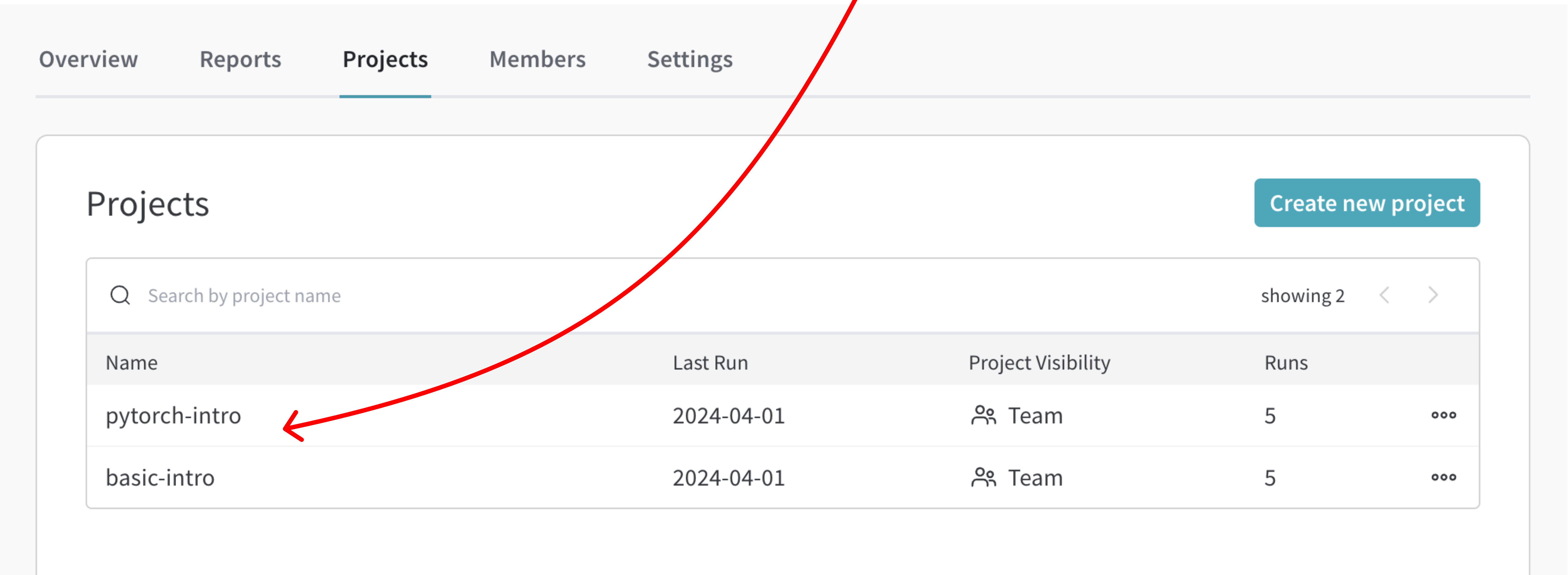
Wandb + PyTorch Tutorial

Ch3. tutorial for wandb + pytorch

In pseudocode, what we'll do is:

```
# import the library
import wandb

# start a new experiment
wandb.init(project="new-sota-model")

# capture a dictionary of hyperparameters with config
wandb.config = {"learning_rate": 0.001, "epochs": 100, "batch_size": 128}

# set up model and data
model, dataloader = get_model(), get_data()

# optional: track gradients
wandb.watch(model)

for batch in dataloader:
  metrics = model.training_step()
  # log metrics inside your training loop to visualize model performance
  wandb.log(metrics)

# optional: save model at the end
model.to_onnx()
wandb.save("model.onnx")
```


```
# import the library
import wandb

# start a new experiment
wandb.init(project="new-sota-model")
```

The project name you set will be **displayed on the wandb homepage**

Ch3. tutorial for wandb + pytorch

In pseudocode, what we'll do is:

```
# import the library
import wandb

# start a new experiment
wandb.init(project="new-sota-model")

# capture a dictionary of hyperparameters with config
wandb.config = {"learning_rate": 0.001, "epochs": 100, "batch_size": 128}

# set up model and data
model, dataloader = get_model(), get_data()

# optional: track gradients
wandb.watch(model)

for batch in dataloader:
  metrics = model.training_step()
  # log metrics inside your training loop to visualize model performance
  wandb.log(metrics)

# optional: save model at the end
model.to_onnx()
wandb.save("model.onnx")
```

```
# capture a dictionary of hyperparameters with config
wandb.config = {"learning_rate": 0.001, "epochs": 100, "batch_size": 128}
```

Declare the **model's hyperparameters** in the python **dictionary type**

Ch3. tutorial for wandb + pytorch

In pseudocode, what we'll do is:

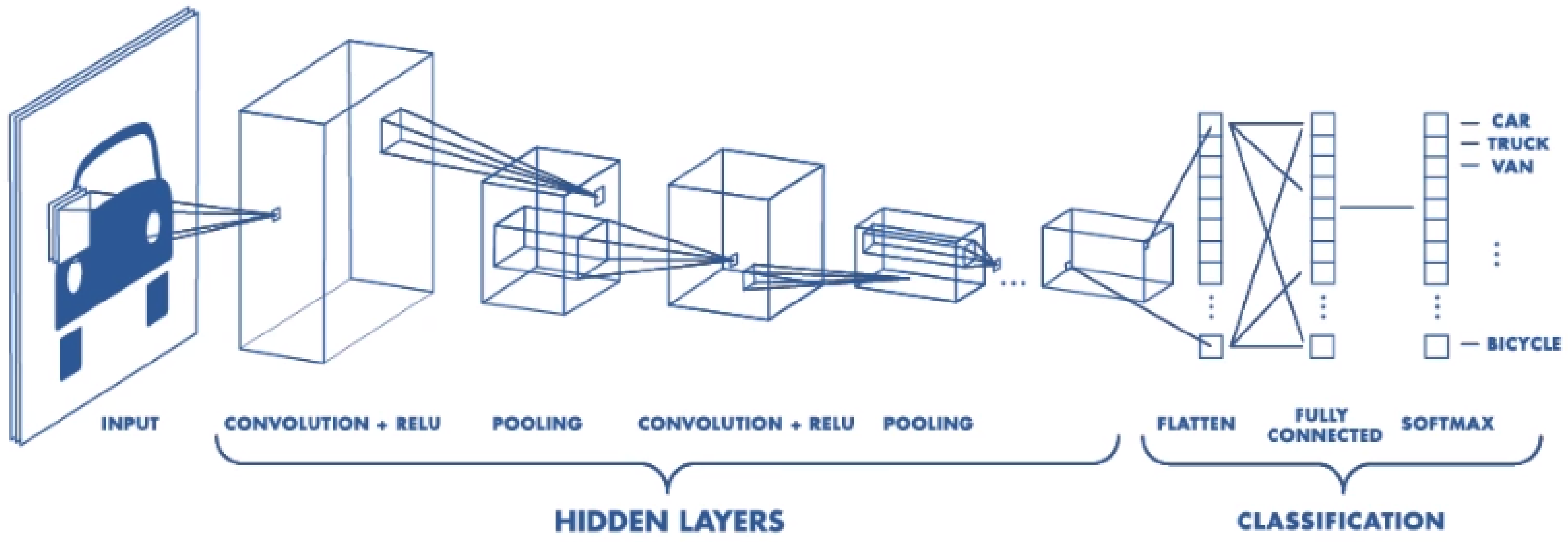
```
# import the library
import wandb

# start a new experiment
wandb.init(project="new-sota-model")

# capture a dictionary of hyperparameters with config
wandb.config = {"learning_rate": 0.001, "epochs": 100, "batch_size": 128}

# set up model and data
model, dataloader = get_model(), get_data()

# optional: track gradients
wandb.watch(model)

for batch in dataloader:
  metrics = model.training_step()
  # log metrics inside your training loop to visualize model performance
  wandb.log(metrics)

# optional: save model at the end
model.to_onnx()
wandb.save("model.onnx")
```

```
# set up model and data
model, dataloader = get_model(), get_data()
```

in this tutorial, we will use model: **CNN**

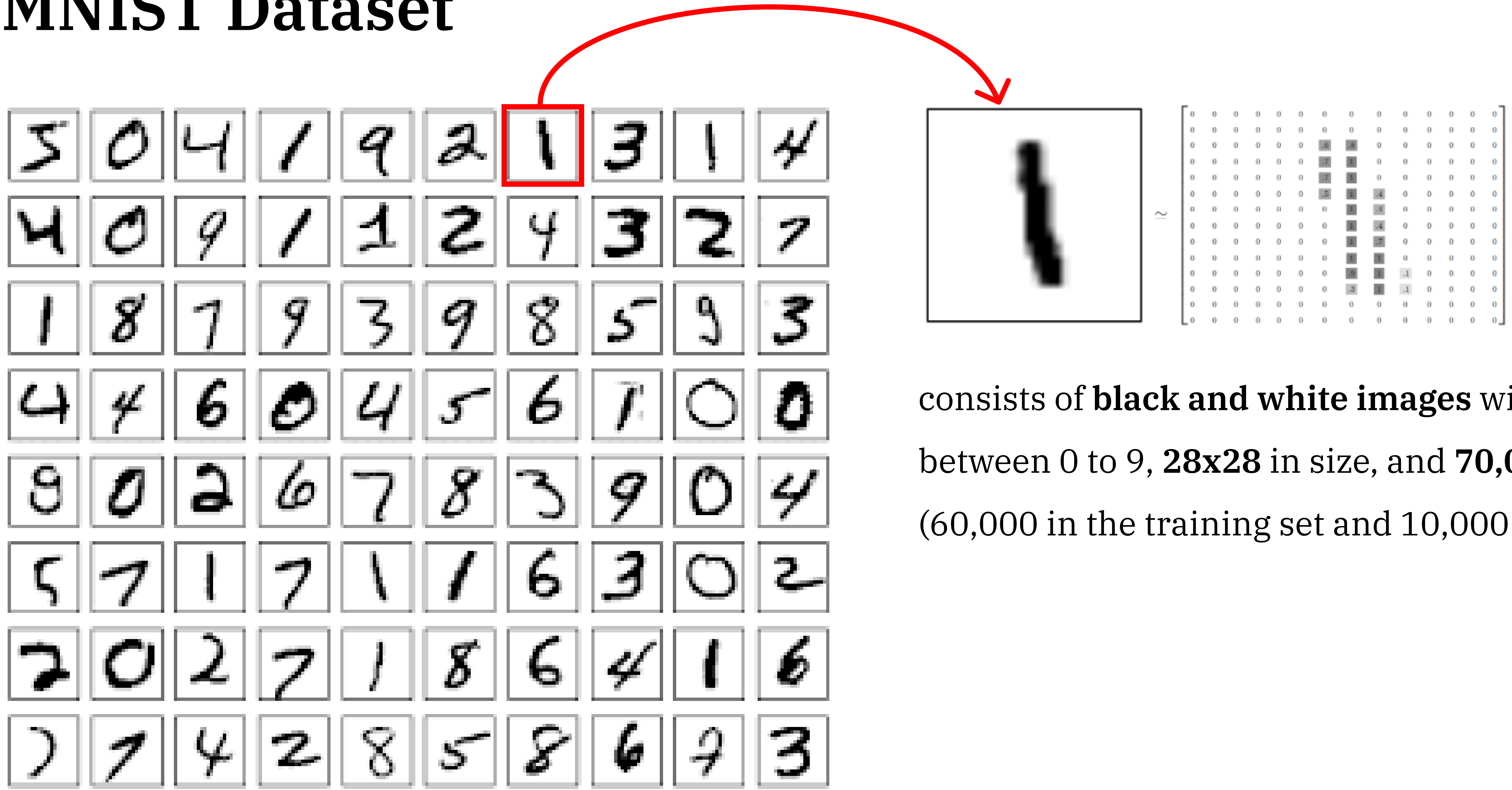
data: **MNIST**

CNN(Convolutional Neural Network)



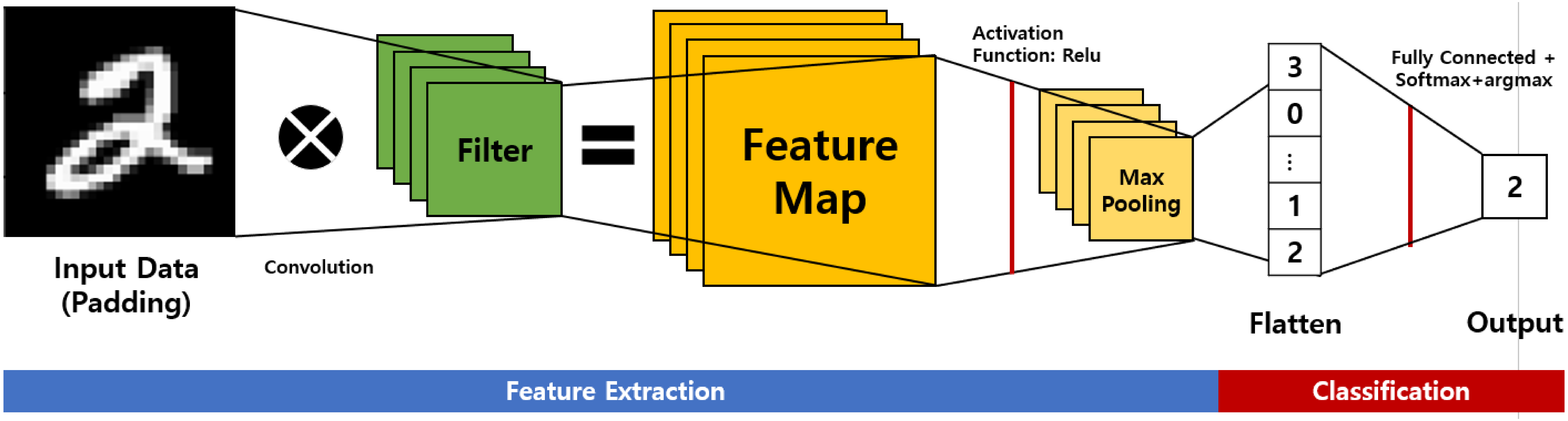
- Models that mimic human visual structure
- Useful for finding patterns to recognize images

MNIST Dataset



consists of **black and white images** with values between 0 to 9, **28x28** in size, and **70,000 images** (60,000 in the training set and 10,000 in the test set)

CNN with MNIST



Ch3. tutorial for wandb + pytorch

In pseudocode, what we'll do is:

```
# import the library
import wandb

# start a new experiment
wandb.init(project="new-sota-model")

# capture a dictionary of hyperparameters with config
wandb.config = {"learning_rate": 0.001, "epochs": 100, "batch_size"

# set up model and data
model, dataloader = get_model(), get_data()

# optional: track gradients
wandb.watch(model)

for batch in dataloader:
  metrics = model.training_step()
  # log metrics inside your training loop to visualize model performance
  wandb.log(metrics)

# optional: save model at the end
model.to_onnx()
wandb.save("model.onnx")
```

```
# optional: track gradients
wandb.watch(model)
```

Allows wandb to **track the process** of training the model.

Ch3. tutorial for wandb + pytorch

In pseudocode, what we'll do is:

```
# import the library
import wandb

# start a new experiment
wandb.init(project="new-sota-model")

# capture a dictionary of hyperparameters with config
wandb.config = {"learning_rate": 0.001, "epochs": 100, "batch_size": 

# set up model and data
model, dataloader = get_model(), get_data()

# optional: track gradients
wandb.watch(model)

for batch in dataloader:
  metrics = model.training_step()
  # log metrics inside your training loop to visualize model performa
  wandb.log(metrics)

# optional: save model at the end
model.to_onnx()
wandb.save("model.onnx")
```

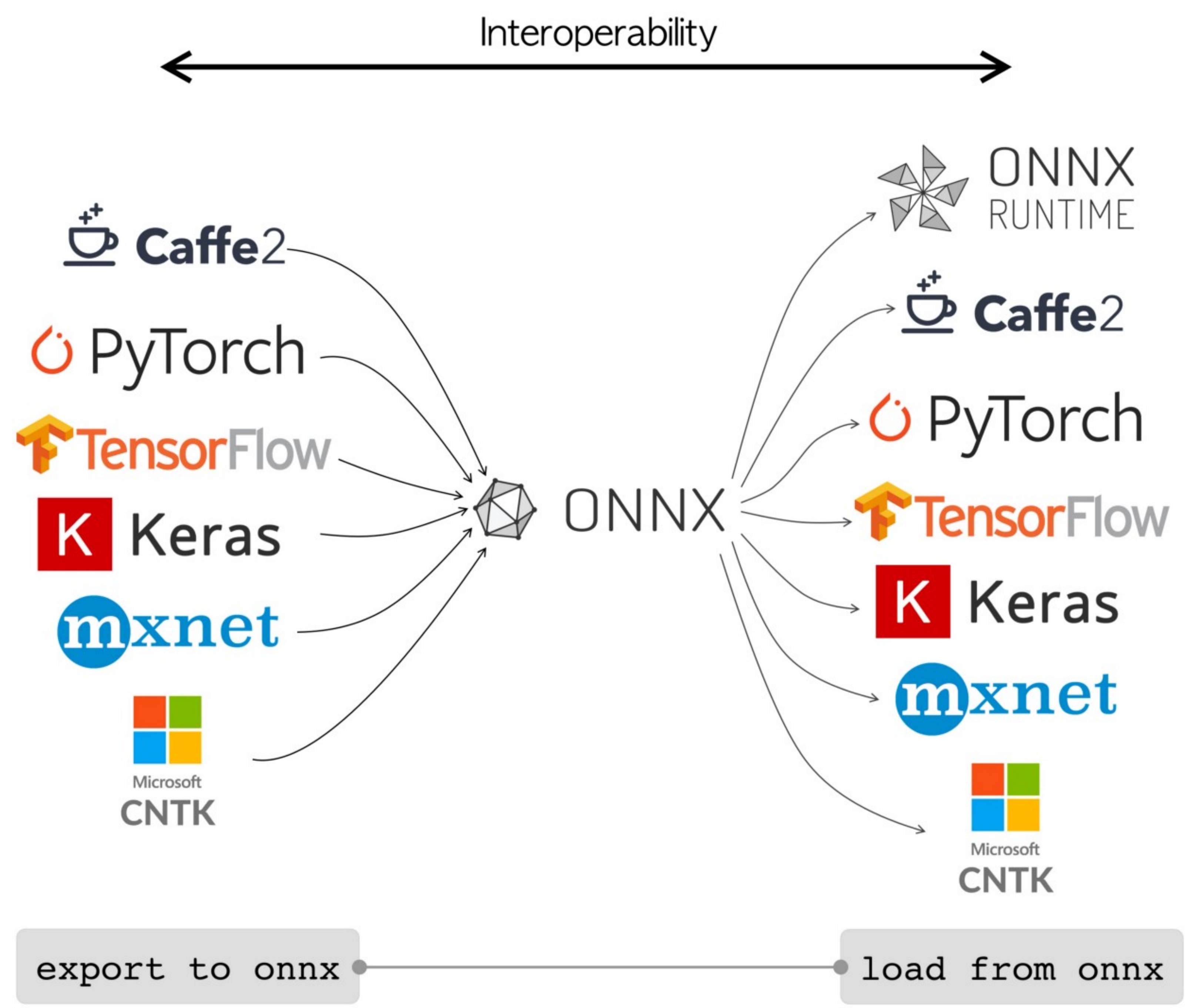
```
for batch in dataloader:
  metrics = model.training_step()
  # log metrics inside your training loop to visualize model performance
  wandb.log(metrics)
```

Train the model and **log** to wandb.

Ch3. tutorial for wandb + pytorch

In pseudocode, what we'll do is:

```
# import the library
import wandb

# start a new experiment
wandb.init(project="new-sota-model")

# capture a dictionary of hyperparameters with config
wandb.config = {"learning_rate": 0.001, "epochs": 100, "batch_size": 128}

# set up model and data
model, dataloader = get_model(), get_data()

# optional: track gradients
wandb.watch(model)

for batch in dataloader:
  metrics = model.training_step()
  # log metrics inside your training loop to visualize model performance
  wandb.log(metrics)

# optional: save model at the end
model.to_onnx()
wandb.save("model.onnx")
```



ONNX is short for **Open Neural Network Exchange**

- It is a **sharing platform** designed to make models created in **different DNN framework environments** (ex Tensorflow, PyTorch, etc.) compatible with each other.

```
# optional: save model at the end
model.to_onnx()
wandb.save("model.onnx")
```

Save the model

Ch3. tutorial for wandb + pytorch

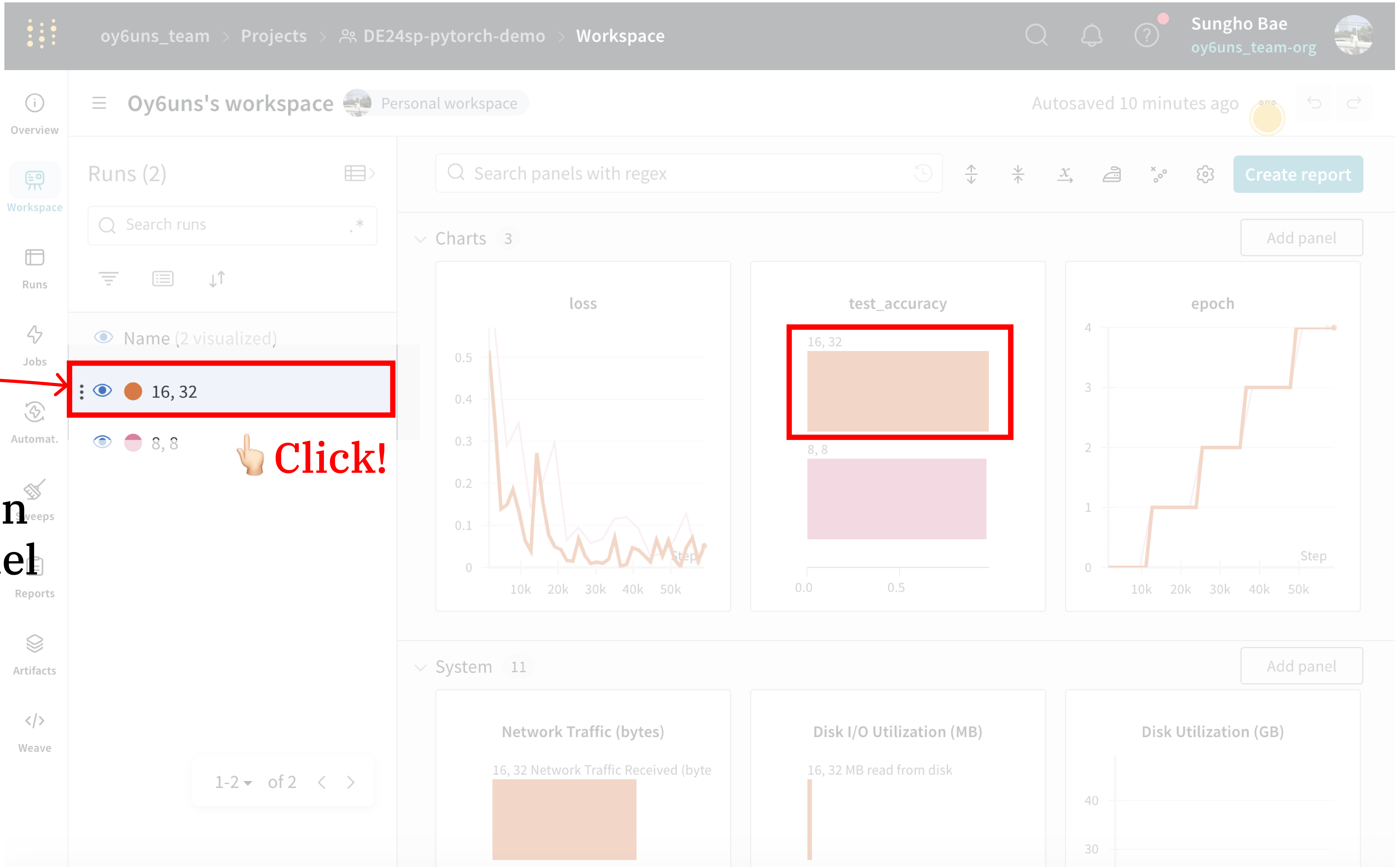
Colab URL: https://colab.research.google.com/drive/1pxMw2Z6cMNSgwCCaPrRTwleiE1l6zjt7?usp=sharing

Click!

model_pipeline(config) — **wandb.init(config)**

- **config = wandb.config**
- **make(config)** : define **model, train_loader, test_loader, criterion, optimizer**
  - **get_data()** : make **CustomDataset**
  - **make_loader()** : make **DataLoader**
  - **class ConvNet()** : define **CNN model** structure
- **train()** : **train** the model
  - **wandb.watch**
  - **train_batch()** : **calculate the loss** for each epoch
  - **train_log()** : **log to wandb** by using **wandb.log**
- **test()** : **test** model's final performance
  - **wandb.log**
  - **wandb.save**

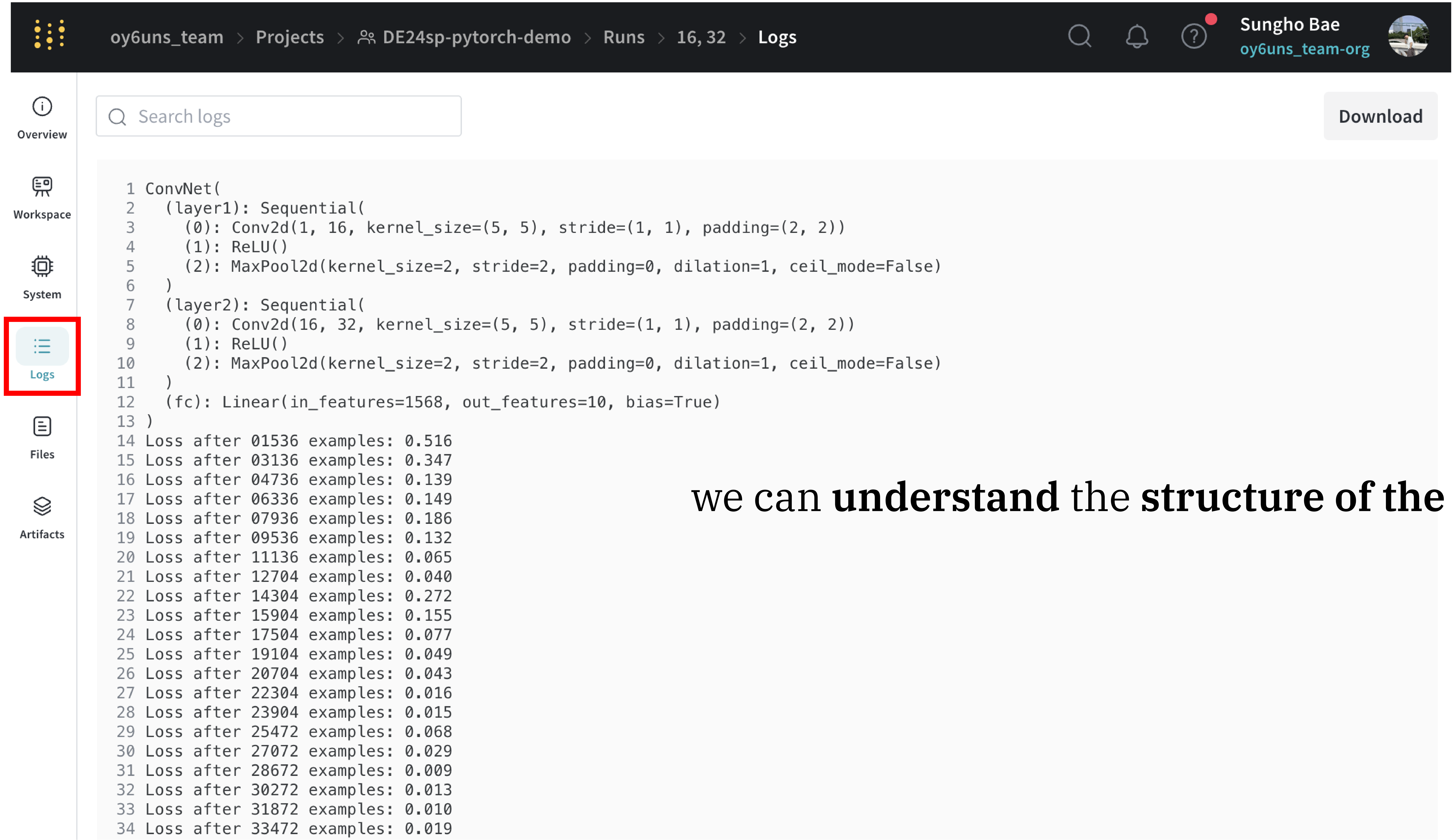
Ch3. tutorial for wandb + pytorch



Check out the information
of the best model

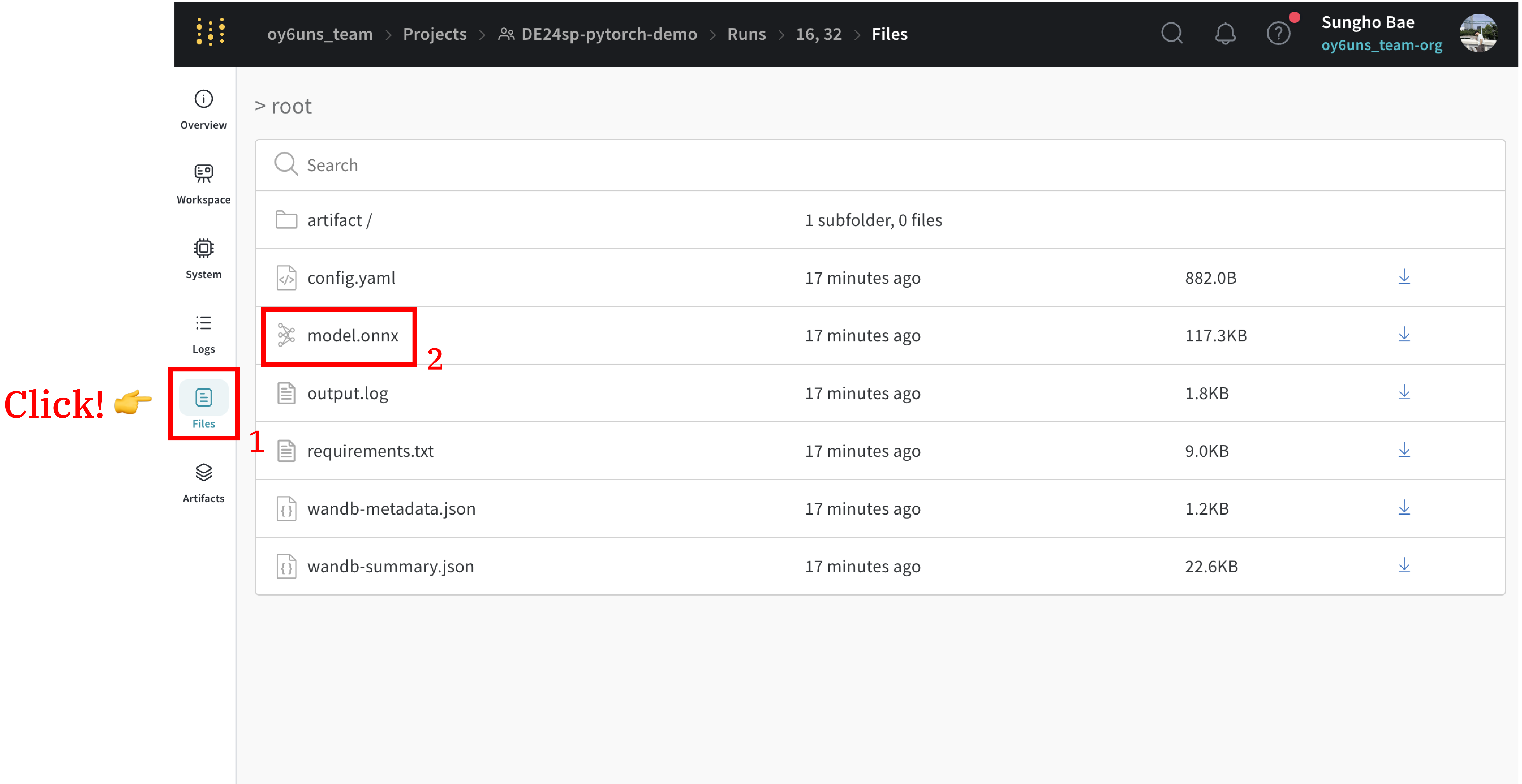
Ch3. tutorial for wandb + pytorch

Click! 👉

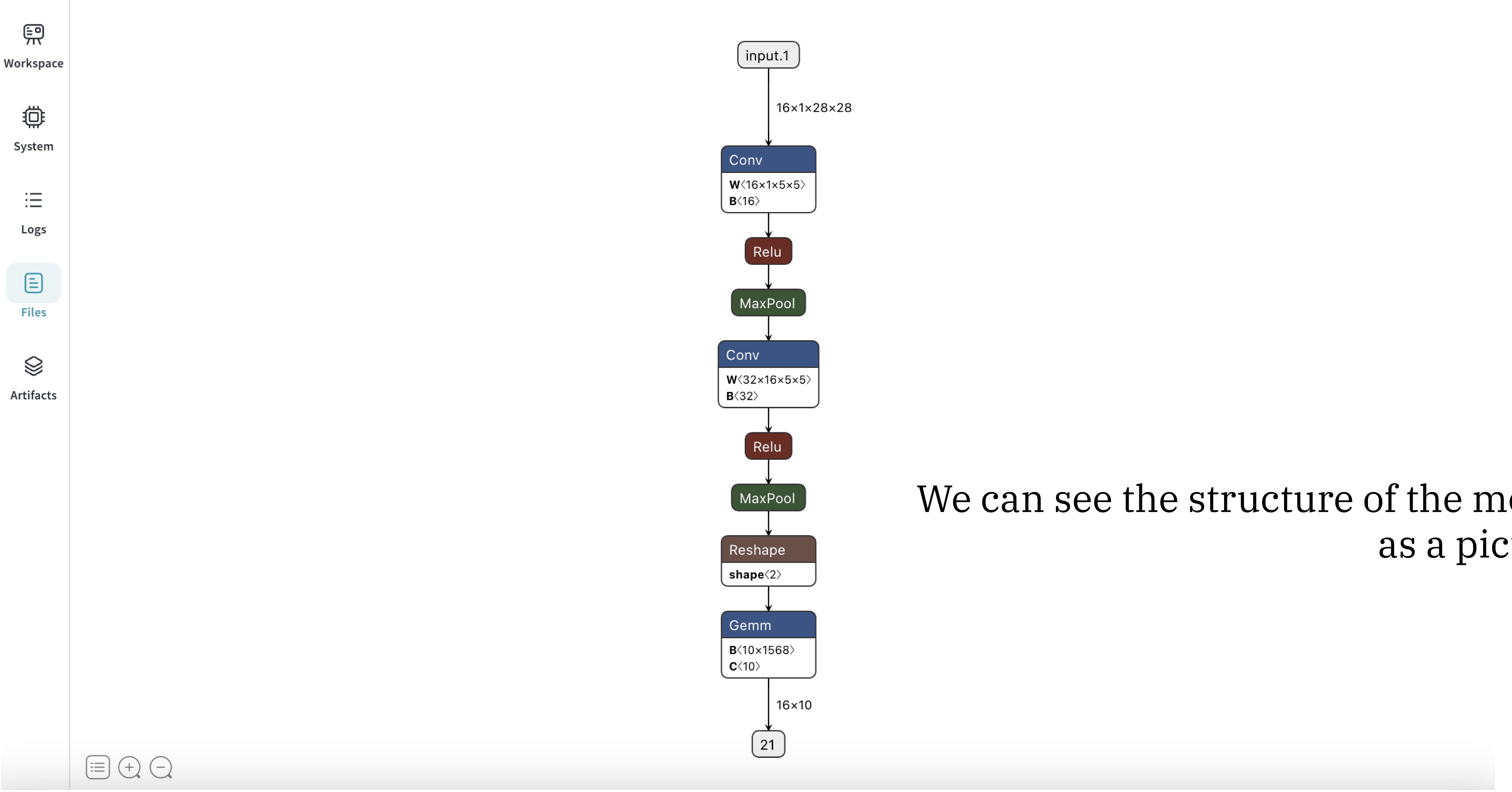


we can **understand** the **structure of the model.**

Ch3. tutorial for wandb + pytorch



Ch3. tutorial for wandb + pytorch



We can see the structure of the model as a picture.

Weights & Biases

Wandb Sweeps

way to find best parameters

Ch4. hyperparameters sweeps

💡simple task : minimize the **value 3x+y**

```python
# Import the W&B Python Library and log into W&B
import wandb

wandb.login()

# 1: Define objective/training function
def objective(config):
    score = config.x**3 + config.y
    return score

def main():
    wandb.init(project="my-first-sweep")
    score = objective(wandb.config)
    wandb.log({"score": score})

# 2: Define the search space
sweep_configuration = {
    "method": "random",
    "metric": {"goal": "minimize", "name": "score"},
    "parameters": {
        "x": {"max": 0.1, "min": 0.01},
        "y": {"values": [1, 3, 7]},
    },
}

# 3: Start the sweep
sweep_id = wandb.sweep(sweep=sweep_configuration, project="my-first-sweep")

wandb.agent(sweep_id, function=main, count=10)
```


```python
# 1: Define objective/training function
def objective(config):
    score = config.x**3 + config.y
    return score

def main():
    wandb.init(project="my-first-sweep")
    score = objective(wandb.config)
    wandb.log({"score": score})
```


https://docs.wandb.ai/guides/sweeps/walkthrough#2-sweep-config

Ch4. hyperparameters sweeps

```python
# Import the W&B Python Library and log into W&B
import wandb

wandb.login()

# 1: Define objective/training function
def objective(config):
    score = config.x**3 + config.y
    return score

def main():
    wandb.init(project="my-first-sweep")
    score = objective(wandb.config)
    wandb.log({"score": score})

# 2: Define the search space
sweep_configuration = {
    "method": "random",
    "metric": {"goal": "minimize", "name": "score"},
    "parameters": {
        "x": {"max": 0.1, "min": 0.01},
        "y": {"values": [1, 3, 7]},
    },
}

# 3: Start the sweep
sweep_id = wandb.sweep(sweep=sweep_configuration, project="my-first-sweep")

wandb.agent(sweep_id, function=main, count=10)
```

💡 simple task : minimize the **value 3x+y**


```python
# 2: Define the search space
sweep_configuration = {
    "method": "random",
    "metric": {"goal": "minimize", "name": "score"},
    "parameters": {
        "x": {"max": 0.1, "min": 0.01},
        "y": {"values": [1, 3, 7]},
    },
}
```


randomly search

x in 0.01 < x < 0.1

y in {1, 3, 7}

ex) x = 0.312, y = 3

Ch4. hyperparameters sweeps

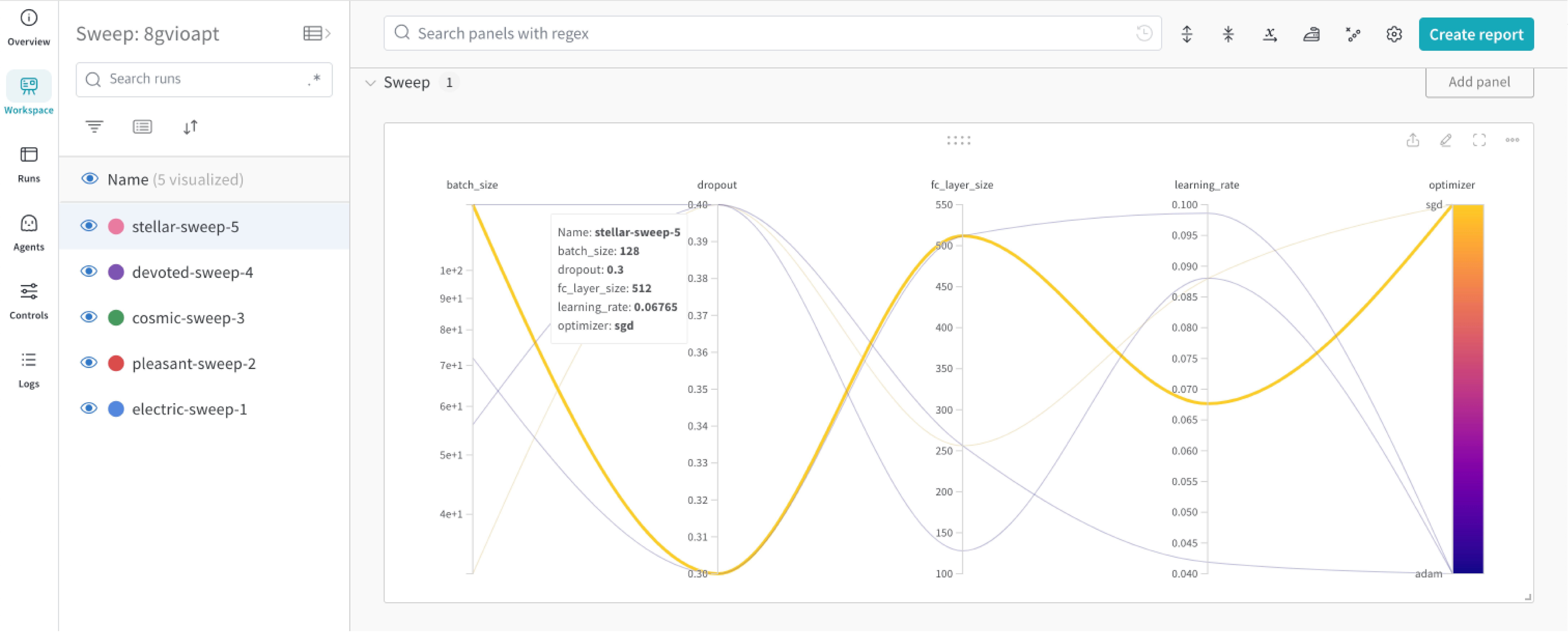
```python
# Import the W&B Python Library and log into W&B
import wandb

wandb.login()

# 1: Define objective/training function
def objective(config):
    score = config.x**3 + config.y
    return score

def main():
    wandb.init(project="my-first-sweep")
    score = objective(wandb.config)
    wandb.log({"score": score})

# 2: Define the search space
sweep_configuration = {
    "method": "random",
    "metric": {"goal": "minimize", "name": "score"},
    "parameters": {
        "x": {"max": 0.1, "min": 0.01},
        "y": {"values": [1, 3, 7]},
    },
}

# 3: Start the sweep
sweep_id = wandb.sweep(sweep=sweep_configuration, project="my-first-sweep")

wandb.agent(sweep_id, function=main, count=10)
```

```python
# 3: Start the sweep
sweep_id = wandb.sweep(sweep=sweep_configuration, project="my-first-sweep")

wandb.agent(sweep_id, function=main, count=10)
```

Randomize the parameters and search **10 times**

Ch4. hyperparameters sweeps

Colab URL: https://colab.research.google.com/drive/1BaG2rD_0tAWm7hSzKgqhfH3tz9BvIDKr?usp=sharing 👈 Click!



Weights & Biases

Thank you