Лабораторная работа №3-4, Часть 1: Знакомство с платформой Hugging Face Hub

Цель работы: Освоить базовые принципы работы с платформой Hugging Face Hub - центральным репозиторием моделей, датасетов и приложений машинного обучения. Получить практические навыки поиска, оценки и загрузки моделей и датасетов для задачи текстовой классификации.

Стек технологий:

- **Операционная система:** Ubuntu 24.04 LTS
- **Менеджер пакетов и окружений:** Conda (окружение `mlops-lab`)
- **Библиотеки:** `huggingface_hub`, `datasets`, `transformers`, `pandas`, `numpy`
- **Платформа:** Hugging Face Hub

Теоретическая часть (краткое содержание)

1. Введение в Hugging Face и экосистему Transformers Hugging Face — это компания и сообщество, создавшее самую популярную в мире open-source платформу для машинного обучения. Ключевые продукты:

- **Transformers:** Библиотека, предоставляющая тысячи предобученных моделей для NLP, компьютерного зрения, аудио и других задач.
- **Hugging Face Hub:** Централизованный репозиторий для обмена моделями, датасетами и демо-приложениями (Spaces).
- **Datasets:** Библиотека для простой загрузки и обработки датасетов.

Hugging Face Hub функционирует как "GitHub для ML", где исследователи и инженеры могут:

- **Обнаруживать** предобученные модели и датасеты
- **Совместно работать** над ML-проектами
- **Делиться** своими разработками с сообществом

2. Ключевые концепции платформы

- **Модели (Models):** Предобученные веса архитектур нейронных сетей (BERT, GPT, ResNet и др.) для различных задач.
- **Датасеты (Datasets):** Коллекции данных для обучения и оценки моделей. Могут быть официальными (от создателей) или community-driven.
- **Spaces:** Интерактивные веб-демонстрации моделей с графическим интерфейсом.
- **Tasks:** Стандартизированные типы ML-задач (текстовая классификация, суммирование, перевод и т.д.).

3. Задача текстовой классификации Текстовая классификация — одна из фундаментальных задач NLP, включающая:

- **Классификация тональности** (sentiment analysis)
- **Классификация тем** (topic classification)

- **Определение спама**
- **Категоризация текстов**

Задание на практическую реализацию

Этап 1: Установка необходимых библиотек

1. **Активация окружения и установка пакетов:**

```
conda activate mlops-lab
pip install huggingface_hub datasets transformers pandas numpy
```

Этап 2: Работа с Hugging Face Hub через веб-интерфейс

1. **Знакомство с интерфейсом:**
   - Откройте Hugging Face Hub в браузере.
   - Изучите главную страницу: разделы Models, Datasets, Spaces, Documentation.
2. **Поиск датасета для текстовой классификации:**
   - Перейдите в раздел **Datasets**.
   - В поиске введите "sentiment analysis" или "text classification".
   - Найдите популярные датасеты:
     - **IMDb** - классификация тональности отзывов на фильмы
     - **AG News** - классификация новостей по темам
     - **Emotion** - классификация эмоций в тексте
   - Выберите датасет **emotion** (отметьте количество примеров, лицензию, язык).
3. **Поиск модели для текстовой классификации:**
   - Перейдите в раздел **Models**.
   - В фильтрах выберите:
     - **Task:** Text Classification
     - **Library:** Transformers
     - **Dataset:** emotion (опционально)
   - Изучите доступные модели, обращая внимание на:
     - Количество загрузок
     - Размер модели
     - Язык
     - Метрики качества (если указаны)
   - Выберите модель **bert-base-uncased** или **distilbert-base-uncased** (более легкая версия).

Этап 3: Программная работа с Hugging Face Hub

1. **Создание Python-скрипта для исследования:**

```
touch hf_hub_exploration.py
```

2. **Написание кода для загрузки датасета:**

```
from datasets import load_dataset
from huggingface_hub import list_models, list_datasets
import pandas as pd

# Исследование доступных датасетов
print("Доступные датасеты для текстовой классификации:")
datasets = list_datasets(filter="task_categories:text-classification")
for dataset in datasets:
    print(f"- {dataset.id}")

# Загрузка датасета emotion
print("\nЗагрузка датасета emotion...")
dataset = load_dataset("emotion")

# Исследование структуры датасета
print(f"\nСтруктура датасета: {dataset}")
print(f"\nПримеры из train split:")
train_df = pd.DataFrame(dataset['train'][:5])
print(train_df)

# Анализ распределения классов
print("\nРаспределение классов в тренировочных данных:")
label_counts = pd.DataFrame(dataset['train']['label']).value_counts()
print(label_counts)
```

3. **Написание кода для исследования моделей:**

```
# Исследование доступных моделей
print("\n\nДоступные модели для текстовой классификации:")
models = list_models(
    filter="task:text-classification",
    sort="downloads",
    direction=-1,
    limit=5
)

for model in models:
    print(f"\nМодель: {model.id}")
    print(f"Загрузок: {model.downloads}")
    print(f"Тэги: {model.tags}")
    if model.pipeline_tag:
        print(f"Тип задачи: {model.pipeline_tag}")
```

4. **Загрузка выбранной модели:**

```
from transformers import AutoTokenizer, AutoModelForSequenceClassification

# Загрузка токенизатора и модели
model_name = "distilbert-base-uncased"
print(f"\nЗагрузка модели {model_name}...")

tokenizer = AutoTokenizer.from_pretrained(model_name)
model = AutoModelForSequenceClassification.from_pretrained(
    model_name,
    num_labels=6  # Количество классов в датасете emotion
)

print("Модель и токенизатор успешно загружены!")
print(f"Размер словаря: {tokenizer.vocab_size}")
print(f"Архитектура модели: {model.__class__.__name__}")
```

5. **Тестирование работы токенизатора:**

```
# Тестирование токенизатора
test_text = "I am feeling very happy today!"
print(f"\nТекст для теста: {test_text}")

tokens = tokenizer(test_text, return_tensors="pt")
print(f"Токены: {tokens}")
print(f"Декодированные токены: {tokenizer.decode(tokens['input_ids'][0])}")
```

6. **Запуск скрипта:**

```
python hf_hub_exploration.py
```

Этап 4: Сохранение локальных копий

1. **Создание директории для проекта:**

```
mkdir text-classification-project
cd text-classification-project
```

2. **Сохранение информации о выбранных ресурсах:**

```
echo "Датасет: emotion" > resources.txt
echo "Модель: distilbert-base-uncased" >> resources.txt
```

```
echo "Количество классов: 6" >> resources.txt
```

Требования к оформлению и отчету

Критерии оценки для Части 1:

- **Удовлетворительно:** Успешно выполнены Этапы 1-2 (установка библиотек, исследование Hub через веб-интерфейс, выбор датасета и модели). Создан файл `resources.txt` с информацией о выбранных ресурсах.
- **Хорошо:** Дополнительно успешно выполнен Этап 3 (написан и запущен скрипт `hf_hub_exploration.py`, который загружает датасет и информацию о моделях). Продемонстрировано понимание структуры датасета.
- **Отлично:** Все задания выполнены в полном объеме. Скрипт дополнен функционалом анализа датасета (статистика по длине текстов, визуализация распределения классов) и тестирования работы модели на примерах из датасета.

Рекомендуемая литература

1. **Официальная документация Hugging Face:** https://huggingface.co/docs
2. **Hugging Face Transformers Documentation:** https://huggingface.co/docs/transformers
3. **Hugging Face Datasets Documentation:** https://huggingface.co/docs/datasets
4. **Статья "Getting Started with Hugging Face":** https://towardsdatascience.com/getting-started-with-hugging-face-transformers-
5. **Книга "Natural Language Processing with Transformers":** Lewis et al. (Главы 1-2)

Лабораторная работа №3-4, Часть 2: Тонкая настройка модели для текстовой классификации

Цель работы: Освоить практические навыки тонкой настройки (fine-tuning) предобученных моделей для задачи текстовой классификации с использованием библиотеки Transformers. Получить опыт подготовки данных, настройки обучения и оценки качества модели.

Стек технологий:

- **ОС:** Ubuntu 24.04 LTS
- **Окружение:** Conda (`mlops-lab`)
- **Библиотеки:** `transformers`, `datasets`, `torch`, `numpy`, `pandas`, `scikit-learn`
- **Фреймворк:** PyTorch
- **Модель:** DistilBERT (`distilbert-base-uncased`)
- **Датасет:** Emotion

Теоретическая часть

1. Тонкая настройка (Fine-tuning) Тонкая настройка — это процесс дополнительного обучения предобученной модели на специфичном для задачи наборе данных. В отличие от обучения с нуля, fine-tuning:

- Требует меньше данных
- Сходится быстрее
- Достигает лучшего качества на целевой задаче

2. Архитектура Transformer для классификации Модели на основе Transformer (BERT, DistilBERT) для классификации состоят из:

- **Энкодера:** Создает контекстуализированные эмбеддинги токенов
- **Пулинга:** Извлекает представление всего текста (обычно [CLS]-токен)
- **Классификационной головки:** Линейный слой для предсказания класса

3. Процесс обучения

- **Токенизация:** Преобразование текста в токены
- **Пакетная обработка:** Группировка примеров для эффективного обучения
- **Прямое распространение:** Получение предсказаний модели
- **Вычисление потерь:** Сравнение предсказаний с истинными метками
- **Обратное распространение:** Обновление весов модели

Задание на практическую реализацию

Этап 1: Подготовка среды и данных

1. **Активация окружения:**

```
conda activate mlops-lab
cd text-classification-project
```

2. **Создание скрипта для обучения:**

```
touch fine_tuning.py
```

3. **Инициализация и загрузка данных:**

```
from datasets import load_dataset
from transformers import (
    AutoTokenizer,
    AutoModelForSequenceClassification,
    TrainingArguments,
    Trainer,
    DataCollatorWithPadding
)
import numpy as np
from sklearn.metrics import accuracy_score, f1_score
import torch

# Загрузка датасета
dataset = load_dataset("emotion")

# Загрузка токенизатора
model_name = "distilbert-base-uncased"
tokenizer = AutoTokenizer.from_pretrained(model_name)
```

Этап 2: Предобработка данных

1. **Токенизация текста:**

```
def tokenize_function(examples):
    return tokenizer(
        examples["text"],
        truncation=True,
        padding=True,
        max_length=128
    )

# Применение токенизации ко всему датасету
tokenized_datasets = dataset.map(tokenize_function, batched=True)

# Форматирование данных для PyTorch
tokenized_datasets = tokenized_datasets.rename_column("label", "labels")
```

```
tokenized_datasets.set_format("torch", columns=["input_ids",
"attention_mask", "labels"])
```

2. **Создание DataCollator:**

```
data_collator = DataCollatorWithPadding(tokenizer=tokenizer)
```

Этап 3: Настройка модели и обучения

1. **Загрузка модели:**

```
# Определение количества классов
num_labels = len(set(dataset["train"]["label"]))

# Загрузка модели с правильным количеством классов
model = AutoModelForSequenceClassification.from_pretrained(
    model_name,
    num_labels=num_labels,
    id2label={0: 'sadness', 1: 'joy', 2: 'love', 3: 'anger', 4: 'fear', 5:
'surprise'},
    label2id={'sadness': 0, 'joy': 1, 'love': 2, 'anger': 3, 'fear': 4,
'surprise': 5}
)
```

2. **Определение метрик:**

```
def compute_metrics(eval_pred):
    predictions, labels = eval_pred
    predictions = np.argmax(predictions, axis=1)

    acc = accuracy_score(labels, predictions)
    f1 = f1_score(labels, predictions, average="weighted")

    return {"accuracy": acc, "f1_score": f1}
```

3. **Настройка гиперпараметров:**

```
training_args = TrainingArguments(
    output_dir="./results",
    learning_rate=2e-5,
    per_device_train_batch_size=16,
    per_device_eval_batch_size=16,
    num_train_epochs=3,
    weight_decay=0.01,
    evaluation_strategy="epoch",
```

```
    save_strategy="epoch",
    load_best_model_at_end=True,
    metric_for_best_model="f1_score",
    logging_dir="./logs",
    logging_steps=100,
    report_to="none"
)
```

Этап 4: Обучение модели

1. **Создание Trainer:**

```
trainer = Trainer(
    model=model,
    args=training_args,
    train_dataset=tokenized_datasets["train"],
    eval_dataset=tokenized_datasets["validation"],
    tokenizer=tokenizer,
    data_collator=data_collator,
    compute_metrics=compute_metrics,
)
```

2. **Запуск обучения:**

```
print("Начало обучения...")
train_result = trainer.train()

# Сохранение модели
trainer.save_model("./emotion-classifier")
tokenizer.save_pretrained("./emotion-classifier")

print("Обучение завершено!")
print(f"Результаты обучения: {train_result.metrics}")
```

Этап 5: Оценка модели

1. **Оценка на тестовых данных:**

```
# Оценка на тестовом наборе
test_results = trainer.evaluate(tokenized_datasets["test"])
print(f"Результаты на тестовых данных: {test_results}")

# Сохранение результатов
with open("test_results.txt", "w") as f:
    f.write(f"Accuracy: {test_results['eval_accuracy']:.4f}\n")
    f.write(f"F1 Score: {test_results['eval_f1_score']:.4f}\n")
```

2. **Тестирование на примерах:**

```
# Функция для предсказания
def predict_emotion(text):
    inputs = tokenizer(text, return_tensors="pt", truncation=True,
padding=True)
    with torch.no_grad():
        outputs = model(**inputs)
    predictions = torch.nn.functional.softmax(outputs.logits, dim=-1)
    predicted_class = torch.argmax(predictions, dim=1).item()
    return model.config.id2label[predicted_class], predictions[0]
[predicted_class].item()

# Тестовые примеры
test_texts = [
    "I am feeling absolutely wonderful today!",
    "This is making me so angry and frustrated",
    "I'm scared about what might happen tomorrow"
]

print("\nТестирование модели:")
for text in test_texts:
    emotion, confidence = predict_emotion(text)
    print(f"Текст: '{text}'")
    print(f"Предсказание: {emotion} (уверенность: {confidence:.3f})")
    print()
```

3. **Запуск скрипта:**

```
python fine_tuning.py
```

Требования к оформлению и отчету

Критерии оценки для Части 2:

- **Удовлетворительно:** Успешно выполнены Этапы 1-3 (подготовка данных, настройка модели). Скрипт запускается без ошибок, начинается процесс обучения.
- **Хорошо:** Дополнительно успешно выполнен Этап 4 (модель прошла полное обучение, сохранена в директорию `emotion-classifier`). Получены метрики на валидационном наборе.
- **Отлично:** Все задания выполнены в полном объеме. Получены метрики на тестовом наборе (файл `test_results.txt`), реализована функция предсказания и протестирована на примерах. Проанализировано качество модели.

Рекомендуемая литература

1. **Hugging Face Transformers Documentation:** https://huggingface.co/docs/transformers/training
2. **Fine-tuning Tutorial:** https://huggingface.co/docs/transformers/training
3. **PyTorch Documentation:** https://pytorch.org/docs/stable/index.html
4. **Research Paper: DistilBERT:** https://arxiv.org/abs/1910.01108
5. **Practical NLP Book:** https://github.com/practical-nlp/practical-nlp

Лабораторная работа №3-4, Часть 3: Интеграция с MLflow для трекинга экспериментов

Цель работы: Освоить интеграцию процесса тонкой настройки моделей с платформой MLflow для комплексного трекинга экспериментов. Научиться автоматически логировать гиперпараметры, метрики, артефакты и модели в ходе обучения.

Стек технологий:

- **ОС:** Ubuntu 24.04 LTS
- **Окружение:** Conda (`mlops-lab`)
- **Библиотеки:** `mlflow`, `transformers`, `datasets`, `torch`
- **Платформа:** MLflow Tracking Server
- **Интеграция:** MLflow + Hugging Face Transformers

Теоретическая часть

1. Интеграция MLflow с Transformers MLflow предоставляет нативные интеграции с популярными ML-фреймворками, включая Hugging Face Transformers. Ключевые возможности:

- **Автоматическое логирование:** Автологирование параметров, метрик и артефактов
- **Модельный регистр:** Версионирование и управление моделями
- **Воспроизводимость:** Фиксация всех компонентов эксперимента

2. Компоненты трекинга для NLP

- **Параметры:** learning rate, batch size, архитектура модели
- **Метрики:** accuracy, F1-score, perplexity, loss
- **Артефакты:** модель, токенизатор, графики обучения
- **Тэги:** задача, датасет, версия модели

3. Стратегии логирования

- **Ручное логирование:** Полный контроль над процессом
- **Автологирование:** Автоматическая фиксация метрик
- **Колбэки:** Интеграция через системные хуки

Задание на практическую реализацию

Этап 1: Подготовка среды и конфигурация

1. **Активация окружения и проверка зависимостей:**

```
conda activate mlops-lab
cd text-classification-project
pip install mlflow
```

2. **Запуск MLflow Tracking Server:**

```
mlflow server --backend-store-uri sqlite:///mlflow.db --default-artifact-root ./mlruns --host 0.0.0.0 --port 5000
```

3. **Создание скрипта для интегрированного обучения:**

```
touch mlflow_integration.py
```

Этап 2: Модификация скрипта обучения с интеграцией MLflow

```
import mlflow
import mlflow.transformers
from datasets import load_dataset
from transformers import (
    AutoTokenizer,
    AutoModelForSequenceClassification,
    TrainingArguments,
    Trainer,
    DataCollatorWithPadding
)
import numpy as np
from sklearn.metrics import accuracy_score, f1_score
import torch
import os

# Настройка MLflow
mlflow.set_tracking_uri("http://localhost:5000")
mlflow.set_experiment("Emotion-Classification-FineTuning")

def compute_metrics(eval_pred):
    predictions, labels = eval_pred
    predictions = np.argmax(predictions, axis=1)

    acc = accuracy_score(labels, predictions)
    f1 = f1_score(labels, predictions, average="weighted")

    return {"accuracy": acc, "f1_score": f1}

def tokenize_function(examples):
    tokenizer = AutoTokenizer.from_pretrained("distilbert-base-uncased")
    return tokenizer(
        examples["text"],
        truncation=True,
        padding=True,
        max_length=128
```

```
    )

# Начало эксперимента MLflow
with mlflow.start_run():
    # Загрузка и подготовка данных
    dataset = load_dataset("emotion")
    tokenized_datasets = dataset.map(tokenize_function, batched=True)
    tokenized_datasets = tokenized_datasets.rename_column("label", "labels")
    tokenized_datasets.set_format("torch", columns=["input_ids", "attention_mask",
"labels"])

    # Параметры модели и обучения
    model_params = {
        "model_name": "distilbert-base-uncased",
        "num_labels": 6,
        "learning_rate": 2e-5,
        "batch_size": 16,
        "num_epochs": 3,
        "weight_decay": 0.01
    }

    # Логирование параметров
    mlflow.log_params(model_params)

    # Загрузка модели
    model = AutoModelForSequenceClassification.from_pretrained(
        model_params["model_name"],
        num_labels=model_params["num_labels"],
        id2label={0: 'sadness', 1: 'joy', 2: 'love', 3: 'anger', 4: 'fear', 5:
'surprise'},
        label2id={'sadness': 0, 'joy': 1, 'love': 2, 'anger': 3, 'fear': 4,
'surprise': 5}
    )

    # Настройка обучения
    training_args = TrainingArguments(
        output_dir="./results",
        learning_rate=model_params["learning_rate"],
        per_device_train_batch_size=model_params["batch_size"],
        per_device_eval_batch_size=model_params["batch_size"],
        num_train_epochs=model_params["num_epochs"],
        weight_decay=model_params["weight_decay"],
        evaluation_strategy="epoch",
        save_strategy="epoch",
        load_best_model_at_end=True,
        metric_for_best_model="f1_score",
        logging_dir="./logs",
        logging_steps=100,
        report_to="none"
    )

    data_collator =
DataCollatorWithPadding(tokenizer=AutoTokenizer.from_pretrained(model_params["mode
l_name"]))
```

```
    # Создание тренера
    trainer = Trainer(
        model=model,
        args=training_args,
        train_dataset=tokenized_datasets["train"],
        eval_dataset=tokenized_datasets["validation"],
        tokenizer=AutoTokenizer.from_pretrained(model_params["model_name"]),
        data_collator=data_collator,
        compute_metrics=compute_metrics,
    )

    # Обучение с логированием метрик
    print("Начало обучения с трекингом в MLflow...")
    train_result = trainer.train()

    # Логирование метрик обучения
    mlflow.log_metrics({
        "train_loss": train_result.metrics["train_loss"],
        "eval_loss": train_result.metrics["eval_loss"],
        "eval_accuracy": train_result.metrics["eval_accuracy"],
        "eval_f1_score": train_result.metrics["eval_f1_score"]
    })

    # Оценка на тестовых данных
    test_results = trainer.evaluate(tokenized_datasets["test"])
    mlflow.log_metrics({
        "test_accuracy": test_results["eval_accuracy"],
        "test_f1_score": test_results["eval_f1_score"]
    })

    # Сохранение и логирование модели
    model_path = "./emotion-classifier-mlflow"
    trainer.save_model(model_path)

    # Логирование модели в MLflow
    mlflow.transformers.log_model(
        transformers_model={
            "model": model,
            "tokenizer": AutoTokenizer.from_pretrained(model_params["model_name"])
        },
        artifact_path="emotion-classifier",
        registered_model_name="distilbert-emotion-classifier"
    )

    # Логирование дополнительных артефактов
    with open("training_summary.txt", "w") as f:
        f.write(f"Training completed successfully!\n")
        f.write(f"Final training loss:
{train_result.metrics['train_loss']:.4f}\n")
        f.write(f"Validation accuracy:
{train_result.metrics['eval_accuracy']:.4f}\n")
        f.write(f"Test accuracy: {test_results['eval_accuracy']:.4f}\n")
```

```
    mlflow.log_artifact("training_summary.txt")

    print("Эксперимент успешно завершен и записан в MLflow!")
```

Этап 3: Запуск и мониторинг эксперимента

1. **Запуск скрипта:**

```
python mlflow_integration.py
```

2. **Мониторинг в MLflow UI:**
   - Откройте `http://localhost:5000` в браузере
   - Найдите эксперимент "Emotion-Classification-FineTuning"
   - Изучите записанные параметры и метрики
   - Проверьте залогированную модель в разделе Artifacts

Этап 4: Дополнительные эксперименты

1. **Создание скрипта для сравнения гиперпараметров:**

```
touch hyperparameter_tuning.py
```

2. **Код для сравнения разных конфигураций:**

```
import mlflow
from mlflow_integration import train_model

# Эксперимент с разными learning rates
learning_rates = [1e-5, 2e-5, 5e-5]

for lr in learning_rates:
    with mlflow.start_run(nested=True):
        mlflow.log_param("learning_rate", lr)
        results = train_model(learning_rate=lr)
        mlflow.log_metrics(results)

print("Эксперимент по подбору learning rate завершен!")
```

Этап 5: Анализ результатов

1. **Создание скрипта для анализа:**

```
touch analyze_results.py
```

2. **Код для анализа экспериментов:**

```
import mlflow
from mlflow.tracking import MlflowClient

client = MlflowClient()

# Получение всех запусков эксперимента
experiment = client.get_experiment_by_name("Emotion-Classification-
FineTuning")
runs = client.search_runs(experiment.experiment_id)

print("Результаты экспериментов:")
for run in runs:
    print(f"Run ID: {run.info.run_id}")
    print(f"Parameters: {run.data.params}")
    print(f"Metrics: {run.data.metrics}")
    print("-" * 50)

# Нахождение лучшего запуска
best_run = min(runs, key=lambda x: x.data.metrics.get('eval_loss',
float('inf')))
print(f"Лучший запуск: {best_run.info.run_id}")
print(f"Лучшие метрики: {best_run.data.metrics}")
```

Требования к оформлению и отчету

Критерии оценки для Части 3:

- **Удовлетворительно:** Успешно выполнены Этапы 1-2 (настройка MLflow, модификация скрипта). Эксперимент запускается и базовые параметры записываются в MLflow.
- **Хорошо:** Дополнительно успешно выполнен Этап 3 (полное обучение с логированием всех метрик, модель зарегистрирована в MLflow). В UI отображаются все артефакты.
- **Отлично:** Все задания выполнены в полном объеме. Проведены дополнительные эксперименты (Этап 4) и выполнен анализ результатов (Этап 5). Сравнены разные конфигурации гиперпараметров.

Рекомендуемая литература

1. **MLflow Documentation:** https://mlflow.org/docs/latest/index.html
2. **MLflow Transformers Integration:** https://mlflow.org/docs/latest/python_api/mlflow.transformers.html
3. **Hugging Face Transformers Training:** https://huggingface.co/docs/transformers/training
4. **MLflow Tracking API:** https://mlflow.org/docs/latest/tracking.html
5. **Experiment Tracking Best Practices:** https://mlflow.org/docs/latest/tracking.html#organizing-runs-in-experiments