MLflow

URL: https://main.taila597c2.ts.net:8443

Что это

MLflow — **трекер экспериментов и model registry** для flavor-ml. Каждый раз когда обучается Item2Vec (через Dagster-asset `ingredient_embedding_model`), training run логируется сюда автоматически: гиперпараметры, метрики оценки, обученный бинарь как артефакт. Со временем MLflow накапливает историю, которую можно искать, сравнивать и (со временем) промоутить выбранные runs в «production».

Если знаком Weights & Biases или Comet: MLflow занимает похожую нишу, но open-source и self-hosted. Три ключевые концепции:

- **Run** — одно выполнение тренировочного скрипта. Хранит params, metrics, artifacts. Иммутабельный.
- **Experiment** — именованный «ведёрко» runs (например `ingredient-embeddings`). Группировка связанных runs.
- **Registered Model** — именованный слот с версиями; каждая версия указывает на artifact какого-то run. Развязывает «вот это мы обучили» и «вот это мы отдаём в продакшен».

Что трекается в этом проекте

Подключено в коммите `8e60e93` («Log Dagster training runs into MLflow with eval probes»). При материализации `ingredient_embedding_model` MLflow записывает:

- **Params**: размер корпуса, размерность вектора, размер окна, число эпох (всё что trainer в `flavor_ml/training/train_embeddings.py` вызывает через `mlflow.log_param`)
- **Metrics**: eval probes — мелкие downstream-задачи, оценивающие качество эмбеддингов (например precision на отложенном наборе пар). См. `flavor_ml/eval/metrics.py`
- **Artifact**: обученный файл `ingredient-embeddings.bin`, хранится в MinIO под капотом (S3-совместимый bucket)
- **Ссылка на git commit**: SHA для воспроизводимости
- **Run name**: обычно Dagster run ID для traceability обратно

Типичные задачи

Посмотреть метрики последнего training run: 1. MLflow → **Experiments** → кликнуть эксперимент `ingredient-embeddings`. 2. Верхняя строка — самый свежий run. Кликнуть. 3. Tabs: Parameters / Metrics / Artifacts / Tags.

Сравнить два run'а бок о бок: 1. В списке runs отметить чекбоксы на двух. 2. **Compare** наверху таблицы. 3. Side-by-side diff params и metrics, плюс графики по каждой метрике.

Скачать обученную модель: 1. Run → tab **Artifacts** → клик по `ingredient-embeddings.bin`. 2. Правый клик → «Save link as» (или иконка download). 3. Файл приходит из MinIO через presigned URL.

Промоутить run в Staging: 1. На странице run кликнуть **Register Model**. 2. Зарегистрировать под существующим именем или создать новое (например `flavor-ml-embeddings`). 3. **Models** в верхнем nav → пик модель → версия → выставить **Stage** в `Staging`. 4. FastAPI inference читает `FLAVOR_MODEL_STAGE=Staging` из `.env.example` — промоушен модели = «деплой» новой версии в API без изменения кода.

Поиск / фильтр runs: - Searchbar понимает SQL-подобный синтаксис: `metrics.eval_precision > 0.7 and params.epochs = 10` - Полезно для запросов «best-so-far», когда runs много.

Что под капотом

- Контейнер: `flavor-ml-mlflow-1`
- Tracking server: MLflow 2.13.0, слушает на 5000 внутри контейнера
- Бэкенд (metadata): Postgres (контейнер `flavor-ml-postgres-1`)
- Бэкенд (artifacts): MinIO (контейнер `flavor-ml-minio-1`, S3-совместимый)
- Публичный доступ: Tailscale Funnel на `:8443` → `localhost:5000`

Подводные камни

- **Нет аутентификации.** Кто угодно с URL может смотреть runs, **удалять runs**, **удалять зарегистрированные модели**, скачивать артефакты. Это самое опасное из публично открытого — злонамеренный посетитель может стереть твою историю обучений. Терпимо пока URL остаётся приватным и проект экспериментальный; за этими границами — недопустимо.
- **Скачивание артефактов требует MinIO.** MLflow выдаёт presigned S3 URL на локальный MinIO. Если MinIO упал (`docker compose ps` показывает `flavor-ml-minio-1` не running), клик по артефакту вернёт S3 error.
- **«Production» — это label, а не среда хостинга.** Промоушен модели в MLflow не деплоит ничего сам — FastAPI inference читает stage лениво при первом запросе. Чтобы заставить перечитать — рестартануть контейнер `api`.

- **История runs живёт в Postgres.** Если сделать `docker compose down --volumes` — volume postgres стирается, история runs пропадает. Для штатных рестартов — `docker compose down` (без `--volumes`).
- **Сравнение runs тормозит** на больших объёмах (~сотни runs). Для экспериментального проекта пофиг.