Milestone 1

Tree

bash

Copy code

```
mlops-llmops-m1/
├─ README.md
├─ CONTRIBUTION.md
├─ LICENSE
├─ CODE_OF_CONDUCT.md
├─ Dockerfile
├─ docker-compose.yml              # (bonus)
├─ Makefile
├─ .pre-commit-config.yaml
├─ .gitignore
├─ .github/workflows/ci.yml
├─ pyproject.toml                  # black, ruff, build system
├─ requirements.txt
├─ src/
│  └─ app/
│     ├─ main.py                   # FastAPI app, /health, /predict
│     ├─ model.py                  # load model v1
│     ├─ instrumentation.py        # prometheus + tokens metric
│     └─ monitoring/
│        ├─ mlflow_utils.py
│        └─ evidently_report.py
├─ tests/
│  ├─ test_app.py
│  └─ golden_queries.json
├─ dvc.yaml                        # if using DVC
├─ data/                           # DVC-tracked or LFS
│  └─ raw/…
└─ models/
```

README

SmartTagger — classify short texts with a tiny LLM

> One-liner: A FastAPI service that tags short marketing texts using a distilled LLM.

Architecture

```mermaid

flowchart LR

  A[Data ingestion] --> B[Training pipeline]

  B --> C[Model registry (MLflow)]

  C --> D[Inference API (FastAPI)]

  D --> E[Monitoring: Prometheus → Grafana]

  D --> F[Evidently drift dashboard]
```

Make

`make dev` → create venv, install deps, pre-commit, run API + dashboards

`make test` → pytest with coverage

`make docker` → build image

`make run-docker` → run container on :8000

`make lint` → ruff & black --check

```
.PHONY: dev test lint docker run-docker audit

dev:
    $(PY) -m venv .venv && . .venv/bin/activate && pip install -U pip
    . .venv/bin/activate && pip install -r requirements.txt
    pre-commit install
    # run app + dashboards in parallel (basic):
    ( . .venv/bin/activate && uvicorn src.app.main:app --reload --port 8000 ) & \
    ( . .venv/bin/activate && python src/app/monitoring/evidently_report.py --serve ) & \
    wait

test:
    pytest -q --cov=src

lint:
    ruff check .
    black --check .

docker:
    docker build -t smarttagger:local .

run-docker:
    docker run --rm -p 8000:8000 smarttagger:local

audit:
    pip-audit --strict --requirement requirements.txt
```

README.md contd

D2 — CONTRIBUTION.md
Goal: show who did what, and the workflow norms.

Template:
```md
# CONTRIBUTION

## Team
| Name | ERP | Role | Areas |
|---|---|---|---|
| Aisha Khan | 23K-001 | Lead | API, CI |
| Bilal Raza | 23K-002 | ML | data prep, training, MLflow |
| Chen Wei | 23K-003 | Infra | Docker, Compose, Grafana |

## Member → Task mapping
- Aisha: FastAPI app, acceptance tests, GH Actions.
- Bilal: dataset curation, training script, MLflow model v1 registered.
- Chen: Dockerfile, non-root user, Prometheus+Grafana, Evidently server.

## Branch naming
- `feat/<scope>` new features (e.g., `feat/predict-endpoint`)
- `fix/<scope>` bug fixes
- `infra/<scope>` infra/tooling (compose, ci)
- `docs/<scope>` docs only
```

Dockerfile

```
# ---- builder ----
FROM python:3.11-slim AS builder
WORKDIR /w
RUN apt-get update && apt-get install -y --no-install-recommends build-essential && rm -rf /var/lib/apt/lists/*
COPY requirements.txt .
RUN pip wheel --no-cache-dir --no-deps -r requirements.txt -w /wheels

# ---- runtime ----
FROM python:3.11-slim
ENV PYTHONUNBUFFERED=1 \
    PIP_DISABLE_PIP_VERSION_CHECK=1
# security hardening
RUN adduser --disabled-password --gecos "" app && \
    apt-get update && apt-get install -y --no-install-recommends curl && \
    rm -rf /var/lib/apt/lists/*
WORKDIR /home/app
COPY --from=builder /w/wheels /wheels
RUN pip install --no-cache /wheels/*
COPY src  /src
```

Compose

```
version: "3.9"
services:
  app:
    build: .
    image: ghcr.io/your/repo:${GIT_SHA:-dev}
    ports: ["8000:8000"]
    environment:
      - CANARY=${CANARY:-false}
      - PROMETHEUS_MULTIPROC_DIR=/tmp
    depends_on: [prometheus]

  prometheus:
    image: prom/prometheus
    ports: ["9090:9090"]
    volumes:
      - ./ops/prometheus.yml:/etc/prometheus/prometheus.yml:ro

  grafana:
    image: grafana/grafana
    ports: ["3000:3000"]
    depends_on: [prometheus]

  evidently:
    build: .
    command: ["python","src/app/monitoring/evidently_report.py","--serve","--port","7000"]
    ports: ["7000:7000"]
```

Actions

```
name: ci

on:
  push:
    branches: [main]
    tags: ['v*']
  pull_request:

permissions:
  contents: read
  packages: write

env:
  REGISTRY: ghcr.io
  IMAGE_NAME: ${{ github.repository }}

jobs:
  lint:
    runs-on: ubuntu-latest
    steps:
      - uses: actions/checkout@v4
      - uses: actions/setup-python@v5
        with: { python-version: '3.11' }
      - run: pip install ruff black
      - run: ruff check .
      - run: black --check
```

MLFlow

python

Copy code

```python
# src/app/monitoring/mlflow_utils.py
import mlflow
from mlflow.models.signature import infer_signature

def log_and_register(model, X_sample, y_sample, name="smarttagger"):
    mlflow.set_tracking_uri("http://localhost:5000")  # or file:./mlruns
    with mlflow.start_run(run_name="train-v1"):
        mlflow.log_params({"model":"logreg","version":"1"})
        signature = infer_signature(X_sample, model.predict(X_sample))
        mlflow.sklearn.log_model(model, "model", signature=signature, registered_model_na
```

Evidently

python

```python
# src/app/monitoring/evidently_report.py
import argparse
from evidently.report import Report
from evidently.metrics import DataDriftPreset
from http.server import SimpleHTTPRequestHandler
from socketserver import TCPServer
import pandas as pd, json, os

def build_report():
    ref = pd.read_json("data/ref.json")
    cur = pd.read_json("data/current.json")
    r = Report(metrics=[DataDriftPreset()])
    r.run(reference_data=ref, current_data=cur)
    os.makedirs("artifacts", exist_ok=True)
    r.save_html("artifacts/evidently.html")

def serve(port=7000):
    os.chdir("artifacts")
    TCPServer.allow_reuse_address = True
    with TCPServer(("", port), SimpleHTTPRequestHandler) as httpd:
        print(f"Serving Evidently at :{port}")
        httpd.serve_forever()

if __name__ == "__main__":
    parser = argparse.ArgumentParser()
    parser.add_argument("--serve", action="store_true")
    parser.add_argument("--port", type=int, default=7000)
    args = parser.parse_args()
    build_report()
    if args.serve:
        serve(args.port)
```

EVIDENTLY AI 0.4.0

DOCS

Home / Demo Project

project id: 9293ead0-cb3b-4785-9a47-a1ebb9596675

DASHBOARD REPORTS TEST SUITES COMPARISONS

From 30/01/2023, 18:17 To 28/02/2023, 18:17

Bike Rental Demand Forecast

Model Calls

697

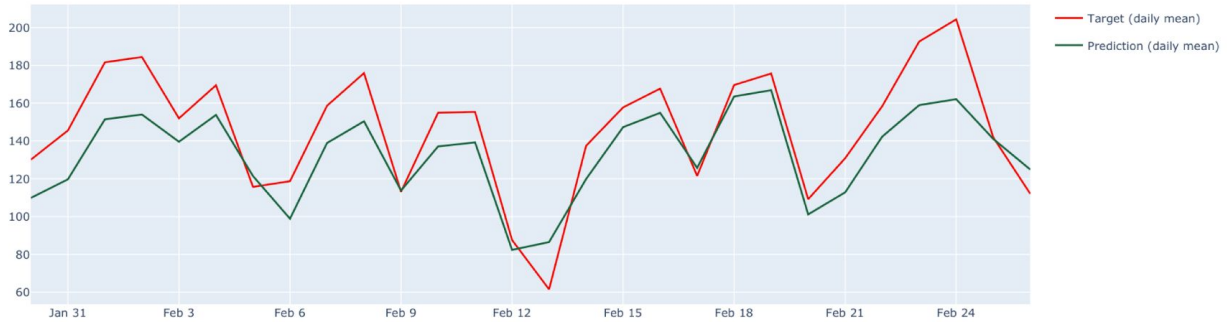
count

Share of Drifted Features

0.545

share

Target and Prediction



Prometheus

python

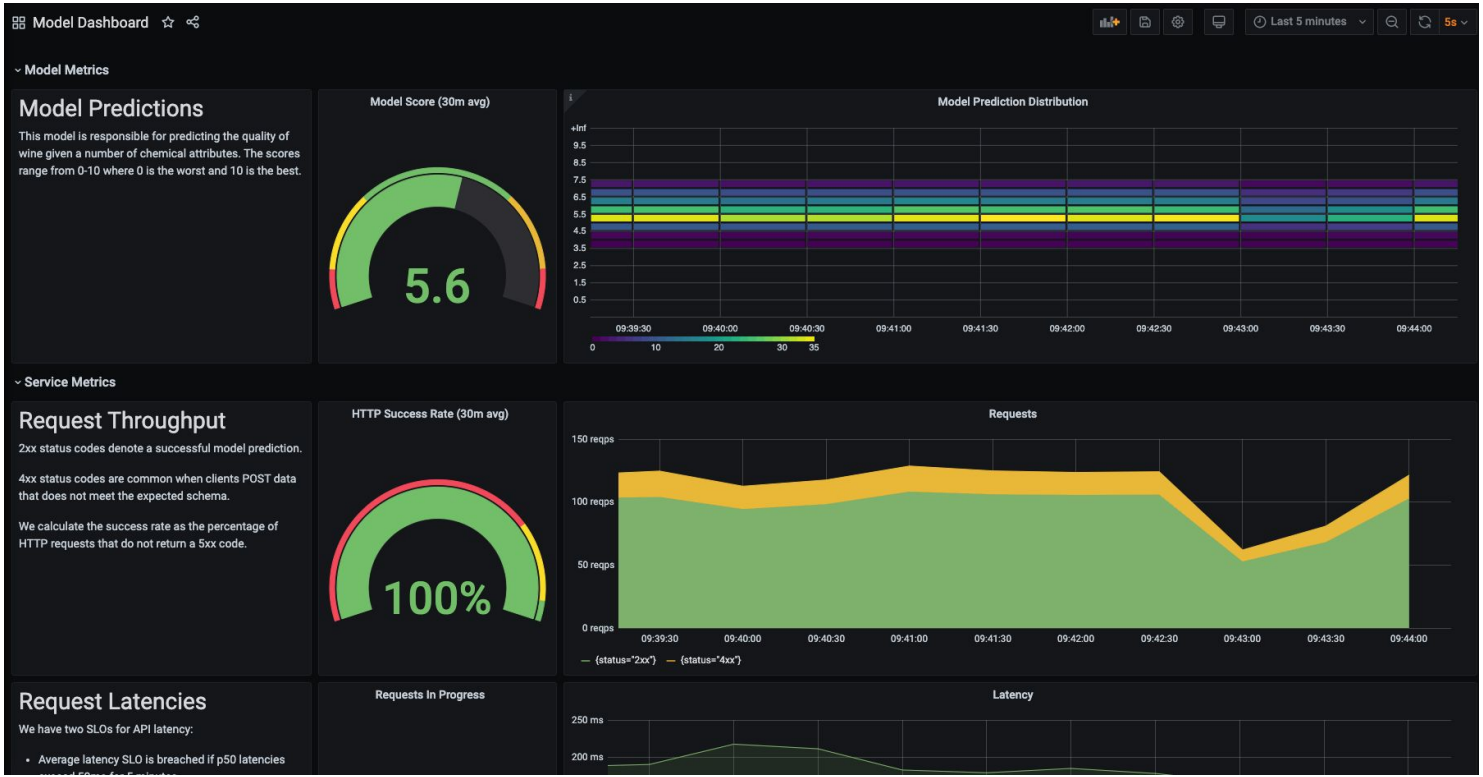
Copy code

```python
# src/app/instrumentation.py
import time
from prometheus_client import Counter, Histogram, Gauge
from starlette.middleware.base import BaseHTTPMiddleware

REQUEST_LATENCY = Histogram("request_latency_seconds", "Request latency", ["endpoint"])
TOKENS_PER_CALL = Histogram("tokens_per_call", "Tokens per call")
GPU_UTIL = Gauge("gpu_utilisation", "GPU Utilisation percent")

class MetricsMiddleware(BaseHTTPMiddleware):
    async def dispatch(self, request, call_next):
        start = time.time()
        resp = await call_next(request)
        dur = time.time() - start
        REQUEST_LATENCY.labels(request.url.path).observe(dur)
        return resp

def observe_tokens(n):
    TOKENS_PER_CALL.observe(n)
```


Model Dashboard
Last 5 minutes
5s
Model Metrics
Model Predictions
This model is responsible for predicting the quality of wine given a number of chemical attributes. The scores range from 0-10 where 0 is the worst and 10 is the best.
Model Score (30m avg)
5.6
Model Prediction Distribution
+Inf
9.5
8.5
7.5
6.5
5.5
4.5
3.5
2.5
1.5
0.5
09:39:30
09:40:00
09:40:30
09:41:00
09:41:30
09:42:00
09:42:30
09:43:00
09:43:30
09:44:00
0
10
20
30
35
Service Metrics
Request Throughput
2xx status codes denote a successful model prediction.
4xx status codes are common when clients POST data that does not meet the expected schema.
We calculate the success rate as the percentage of HTTP requests that do not return a 5xx code.
HTTP Success Rate (30m avg)
100%
Requests
150 reqps
100 reqps
50 reqps
0 reqps
09:39:30
09:40:00
09:40:30
09:41:00
09:41:30
09:42:00
09:42:30
09:43:00
09:43:30
09:44:00
{status="2xx"}
{status="4xx"}
Request Latencies
We have two SLOs for API latency:
Average latency SLO is breached if p50 latencies
Requests In Progress
Latency
250 ms
200 ms

Commit hooks

```
# .pre-commit-config.yaml
repos:
  - repo: https://github.com/pre-commit/pre-commit-hooks
    rev: v4.6.0
    hooks:
      - id: trailing-whitespace
      - id: end-of-file-fixer
      - id: check-added-large-files
  - repo: https://github.com/Yosai-Labs/detect-secrets
    rev: v1.4.0
    hooks:
      - id: detect-secrets
  - repo: https://github.com/psf/black
    rev: 24.8.0
    hooks: [{ id: black }]
  - repo: https://github.com/astral-sh/ruff-pre-commit
    rev: v0.6.9
    hooks: [{ id: ruff }]
```

FastAPI

python

Copy code

```python
# src/app/main.py
from fastapi import FastAPI, Query
from pydantic import BaseModel
import os
from .instrumentation import MetricsMiddleware, observe_tokens

app = FastAPI(title="SmartTagger")
app.add_middleware(MetricsMiddleware)

class PredictIn(BaseModel):
    text: str

class PredictOut(BaseModel):
    label: str
    tokens_used: int

@app.get("/health")
def health():
    return {"ok": True, "canary": os.getenv("CANARY", "false")}

@app.post("/predict", response_model=PredictOut)
def predict(inp: PredictIn):
    tokens = len(inp.text.split())
    observe_tokens(tokens)
    # mock logic:
    label = "POSITIVE" if "good" in inp.text.lower() else "NEUTRAL"
    return {"label": label, "tokens_used": tokens}
```

cURL

```
curl -X POST http://localhost:8000/predict \
  -H "Content-Type: application/json" \
  -d '{"text":"This is a good launch"}'
```

`tests/golden_queries.json`

json

Copy code

```
[
  "This is a good feature",
  "neutral statement",
  "good vibes only",
  "launch went good",
  "nothing to see here"
]
```

`tests/test_app.py`

python

Copy code

```python
from fastapi.testclient import TestClient
from src.app.main import app

client = TestClient(app)

def test_health():
    r = client.get("/health")
    assert r.status_code == 200
    assert "ok" in r.json()

def test_predict():
    r = client.post("/predict", json={"text": "This is good"})
    assert r.status_code == 200
    body = r.json()
    assert set(body.keys()) == {"label","tokens_used"}
```