Ad-Transparency-Agent — Starter Repo

This starter repo is a complete, runnable blueprint for an AI agent that pulls ad creatives and stats from **Meta (Ads Library / Graph API)** and **Google Ads Transparency (BigQuery public dataset)**, normalizes and scores them, and exposes a small Streamlit dashboard for inspection.

> Files are presented below. Copy into a Git repo (or I can generate a downloadable archive if you want). Follow the README to run locally or in a container.

Repo structure

```
ad-transparency-agent/
├─ README.md
├─ requirements.txt
├─ .env.example
├─ Dockerfile
├─ docker-compose.yml
├─ src/
│  ├─ meta_fetcher.py
│  ├─ google_bigquery_fetcher.py
│  ├─ normalizer.py
│  ├─ scorer.py
│  ├─ db.py
│  ├─ storage.py
│  ├─ streamlit_app.py
│  ├─ utils.py
│  └─ config.py
├─ infra/
│  ├─ airflow_dag_example.py
│  └─ k8s_deployment.yaml
└─ sql/
   └─ top_google_creatives.sql
```

README.md

```
# Ad Transparency Agent — Starter Repo

## What this repo contains
A starter implementation that:
- Fetches ads from Meta Ads Library (Graph API)
- Queries Google Ads Transparency public dataset in BigQuery
- Normalizes creatives into a common schema
- Scores creatives with a simple weighted model
- Serves a Streamlit dashboard to inspect top candidates

## Quick start (local)
1. Copy `.env.example` to `.env` and fill values (META_GRAPH_TOKEN,
GOOGLE_APPLICATION_CREDENTIALS path, GCP_PROJECT).
2. Create a Python venv and install deps: `pip install -r requirements.txt`.
3. Run streamlit: `streamlit run src/streamlit_app.py`.

## Docker
Build: `docker build -t ad-agent .`
Run: `docker run --env-file .env -p 8501:8501 ad-agent`

## Credentials
- Meta: Graph API token with Ads Archive access. Put in `.env` as
META_GRAPH_TOKEN.
- Google: Service account JSON and set `GOOGLE_APPLICATION_CREDENTIALS` env var
or place in `.env`.

## Notes
- This repo uses the BigQuery public dataset for Google Ads Transparency —
querying costs may apply to your GCP project.
- Respect Meta & Google rate limits. Use backoff and caching.
```

.env.example

```
# Meta
META_GRAPH_TOKEN=your_meta_graph_token_here
META_API_VERSION=v17.0
# Google
GOOGLE_APPLICATION_CREDENTIALS=/path/to/gcp-service-account.json
GCP_PROJECT=your-gcp-project
# General
DEFAULT_COUNTRY=US
DB_URL=sqlite:///ad_agent.db
```

---

requirements.txt

```
requests
google-cloud-bigquery
pandas
sqlalchemy
streamlit
python-dotenv
tqdm
pillow
pytesseract
python-multipart
fastapi
uvicorn
werkzeug
beautifulsoup4
lxml
```

---

src/config.py

```python
from dotenv import load_dotenv
import os
load_dotenv()

META_GRAPH_TOKEN = os.getenv("META_GRAPH_TOKEN")
META_API_VERSION = os.getenv("META_API_VERSION", "v17.0")
GOOGLE_APPLICATION_CREDENTIALS = os.getenv("GOOGLE_APPLICATION_CREDENTIALS")
GCP_PROJECT = os.getenv("GCP_PROJECT")
DEFAULT_COUNTRY = os.getenv("DEFAULT_COUNTRY", "US")
DB_URL = os.getenv("DB_URL", "sqlite:///ad_agent.db")
```

---

src/db.py

```python
from sqlalchemy import create_engine, MetaData, Table, Column, Integer, String,
Text, Float
from sqlalchemy.orm import sessionmaker
from config import DB_URL

engine = create_engine(DB_URL, connect_args={}
                       if DB_URL.startswith("sqlite") else {})
```

```
Session = sessionmaker(bind=engine)
metadata = MetaData()

creatives = Table(
    'creatives', metadata,
    Column('id', Integer, primary_key=True, autoincrement=True),
    Column('creative_id', String, index=True),
    Column('platform', String),
    Column('advertiser_name', String),
    Column('creative_text', Text),
    Column('media_urls', Text),
    Column('first_seen', String),
    Column('last_seen', String),
    Column('times_shown_lower', Integer),
    Column('times_shown_upper', Integer),
    Column('score', Float),
    Column('raw_json', Text),
)


def init_db():
    metadata.create_all(engine)

if __name__ == '__main__':
    init_db()
```

---

src/meta_fetcher.py

```
import requests
import time
import json
from config import META_GRAPH_TOKEN, META_API_VERSION, DEFAULT_COUNTRY

BASE = f"https://graph.facebook.com/{META_API_VERSION}/ads_archive"

def search_meta_ads(search_terms: str, country: str = DEFAULT_COUNTRY, limit:
int = 50):
    """Simple wrapper to query Ads Library (Ads Archive)."""
    params = {
        'access_token': META_GRAPH_TOKEN,
        'search_terms': search_terms,
        'ad_reached_countries': country,
        'ad_active_status': 'ALL',
        'limit': limit,
    }
    resp = requests.get(BASE, params=params)
    if resp.status_code != 200:
        raise Exception(f"Meta API error {resp.status_code}: {resp.text}")
    return resp.json()

if __name__ == '__main__':
    out = search_meta_ads('weight loss', limit=10)
    print(json.dumps(out, indent=2))
```

Note: Meta endpoints and param names change; read official docs if you get errors. Add retry/backoff.

---

sql/top_google_creatives.sql

```sql
SELECT
  creative_id,
  advertiser_id,
  ANY_VALUE(advertiser_name) as advertiser_name,
  MAX(times_shown_upper_bound) as max_times_shown_upper,
  MIN(first_seen) as first_seen,
  MAX(last_seen) as last_seen,
  ANY_VALUE(creative_format) as format
FROM `bigquery-public-data.google_ads_transparency_center.creative_stats`
WHERE DATE(first_seen) >= DATE_SUB(CURRENT_DATE(), INTERVAL 30 DAY)
GROUP BY creative_id, advertiser_id
ORDER BY max_times_shown_upper DESC
LIMIT 200;
```

---

src/google_bigquery_fetcher.py

```python
from google.cloud import bigquery
import pandas as pd
from config import GCP_PROJECT

client = bigquery.Client(project=GCP_PROJECT)

def query_top_google_creatives(sql: str):
    job = client.query(sql)
    return job.result().to_dataframe()

if __name__ == '__main__':
    sql = open('../sql/top_google_creatives.sql').read()
    df = query_top_google_creatives(sql)
    print(df.head())
```

---

src/normalizer.py

```
import json

# Convert platform-specific ad records into a canonical schema

def normalize_meta_ad(raw: dict):
    # raw is the JSON object from Meta
    return {
        'creative_id': raw.get('id') or raw.get('ad_snapshot_url'),
        'platform': 'meta',
        'advertiser_name': raw.get('page_name') or raw.get('sponsored_by'),
        'creative_text': raw.get('ad_creative_body') or raw.get('body') or '',
        'media_urls': json.dumps(raw.get('image_urls', []) or
raw.get('video_urls', [])),
        'first_seen': raw.get('date_start'),
        'last_seen': raw.get('date_stop'),
        'times_shown_lower': raw.get('impressions_lower_bound'),
        'times_shown_upper': raw.get('impressions_upper_bound'),
        'raw_json': json.dumps(raw),
    }


def normalize_google_row(row):
    return {
        'creative_id': str(row['creative_id']),
        'platform': 'google',
        'advertiser_name': row.get('advertiser_name'),
        'creative_text': row.get('creative_text', ''),
        'media_urls': row.get('media_urls', ''),
        'first_seen': str(row.get('first_seen')),
        'last_seen': str(row.get('last_seen')),
        'times_shown_lower': row.get('times_shown_lower'),
        'times_shown_upper': row.get('max_times_shown_upper'),
        'raw_json': str(row),
    }
```

---

src/scorer.py

```python
# Simple weighted scoring engine

def score_creative(c):
    # c is normalized dict
    score = 0.0
    # impression signal
    upp = c.get('times_shown_upper') or 0
    low = c.get('times_shown_lower') or 0
    # normalize impression to log scale
    import math
    imp_signal = math.log(1 + upp)
    score += imp_signal * 30 / 10.0

    # recency bonus
    from datetime import datetime
    try:
        last_seen = datetime.fromisoformat(c.get('last_seen'))
        days_old = (datetime.utcnow() - last_seen).days
        if days_old <= 7:
            score += 10
        elif days_old <= 30:
            score += 5
    except Exception:
        pass

    # engagement proxy: presence of comments/likes in raw_json
    raw = c.get('raw_json', '')
    if 'comment' in raw or 'like' in raw:
        score += 10

    # creative richness: media urls
    if c.get('media_urls'):
        score += 5

    # clamp
    if score > 100:
        score = 100.0
    return score
```

src/streamlit_app.py

```python
import streamlit as st
import pandas as pd
from src.google_bigquery_fetcher import query_top_google_creatives
from src.normalizer import normalize_google_row
from src.scorer import score_creative
import json, os

st.set_page_config(page_title='Ad Transparency Explorer')
st.title('Ad Transparency Explorer — Starter')

st.markdown('Use the controls to pull recent top creatives from Google or
inspect saved Meta results.')

keyword = st.text_input('Keyword to search (Google BigQuery filter - simple):',
'weight loss')
if st.button('Run Google Query'):
    sql = open('sql/top_google_creatives.sql').read()
    # naive replacement - in prod use parameterized queries
    df = query_top_google_creatives(sql)
    st.write('Results from BigQuery (sample)')
    st.dataframe(df.head(20))

st.markdown('---')

st.markdown('Local DB: Top scored creatives')
# In a real implementation, read from DB. Here we show placeholder.
st.info('This demo shows how results would be displayed. Use the fetchers to
populate the DB.')
```

---

infra/airflow_dag_example.py

```python
# Example Airflow DAG pseudocode — save as an Airflow DAG and adapt to your env
from airflow import DAG
from airflow.operators.python_operator import PythonOperator
from datetime import datetime, timedelta

def fetch_google():
    from src.google_bigquery_fetcher import query_top_google_creatives
    sql = open('/opt/airflow/sql/top_google_creatives.sql').read()
    df = query_top_google_creatives(sql)
    df.to_csv('/opt/airflow/data/google_creatives.csv', index=False)
```

```
def fetch_meta():
    from src.meta_fetcher import search_meta_ads
    data = search_meta_ads('weight loss', country='US', limit=100)
    import json
    with open('/opt/airflow/data/meta_creatives.json', 'w') as f:
        json.dump(data, f)

with DAG(dag_id='ad_transparency_pipeline', start_date=datetime(2025,1,1),
schedule_interval='@daily') as dag:
    t1 = PythonOperator(task_id='fetch_google', python_callable=fetch_google)
    t2 = PythonOperator(task_id='fetch_meta', python_callable=fetch_meta)
    t1 >> t2
```

Dockerfile

```
FROM python:3.11-slim
WORKDIR /app
COPY requirements.txt ./
RUN pip install --no-cache-dir -r requirements.txt
COPY . /app
ENV PYTHONUNBUFFERED=1
CMD ["streamlit", "run", "src/streamlit_app.py", "--server.port=8501", "--
server.address=0.0.0.0"]
```

Notes & next steps

1. **Credentials**: Provide a Meta Graph API token and a GCP service account key (or set `GOOGLE_APPLICATION_CREDENTIALS`) to run the fetchers.
2. **Rate-limiting & retries**: Add robust exponential backoff and paging for the Meta fetcher. Consider caching recent responses.
3. **Enhancements**:
4. OCR (pytesseract) on images to extract on-image copy.
5. Video transcription (OpenAI Whisper or cloud speech-to-text).
6. More advanced scoring (machine learning model / heuristic improvements).
7. Add a FastAPI layer to expose search endpoints and integrate with the Streamlit UI.
8. **Legal & compliance**: Use official APIs and respect terms of service.

If you want, I can: - Export this as a zip file you can download and run. - Add CI (GitHub Actions) and a one-click deploy to GCP Cloud Run. - Expand the Streamlit UI to show images, media players, and filtering/save-to-DB actions.