Electronix AI - Assignment

Objective: Build and deliver an end-to-end microservice **and** a minimal web client for binary sentiment analysis:

1. **Python backend** – REST **or** GraphQL API that loads a Hugging Face Transformer for sentiment inference.
2. **React (or equivalent) frontend** – calls the backend and displays the prediction.
3. **Fine-tuning script** – a stand-alone CLI that accepts a small labelled dataset and updates model weights.
   Note: All components must be runnable locally and containerised with Docker Compose.

Core Features

1. Model loading & inference (backend)

- Pull any **English** text-classification model from the Transformers Hub.
- Expose `POST /predict` (or GraphQL mutation/query) that returns

```
{ "label": "positive" | "negative", "score": float }
```

- Re-load the latest fine-tuned weights on service start-up (if present in `./model`).

2. Fine-tuning (stand-alone script)

- CLI:

```
python finetune.py --data data.jsonl --epochs 3 --lr 3e-5
```

- `data.jsonl` format – one entry per line, e.g. `{"text": "Great product!", "label": "positive"}`
    - Implement a training loop with:
        - Cross-entropy (or BCE) loss
        - Gradient clipping
        - LR scheduler
    - Save updated weights to `./model/` (picked up automatically by the API on next restart).
- Pin random seeds (Python, NumPy, framework) for deterministic CPU runs.

---

3. Frontend (React / Vue / Svelte, etc.)

- One page with:
    - Textarea + *Predict* button
    - Display of returned label & score

- Optional: TypeScript implementation, GraphQL client, nice styling.

---

4. Containerisation & reproducibility

- `Dockerfile` for the backend + `docker-compose.yml` with:
  - **app** service on port 8000
  - **frontend** service on port 3000 (or your choice)
  - Optional: a **gpu** profile/service using a CUDA base image
- Project must start with:

```
docker-compose up --build
```

and run on **CPU-only** machines.

- Provide `requirements.txt` or `pyproject.toml`

---

5. Documentation & deliverables

- `README.md` (≈ 1 page) covering:
  - Setup & run instructions
  - Design decisions
  - Approx. CPU vs. GPU fine-tune times (if GPU tested)
- **API docs** (OpenAPI/GraphQL SDL or a short section in the README).
- A screen recording of your project (voiceover optional), explaining working, tech-stack, build process (under 3

minutes). You can upload it to youtube, unlist and share the link of the same.
- Deployed version of the same on platforms similar to vercel - if feasible.
- Docker file.
  IMPORTANT: Failing to complete and submit all the aforementioned deliverables before deadline will result in disqualification.

Optional Enhancements

- Support **both** PyTorch *and* TensorFlow via config flag.
- Async batching of `/predict` requests for higher throughput.
- Model quantisation (bitsandbytes / ONNX / TensorRT) with before/after speed results.
- GitHub Actions workflow that builds the Docker image(s) and runs unit tests on every push.
- Frontend extras: dark-mode toggle, live typing inference, etc.

Bonus Points

- TypeScript + React + GraphQL.
- Automatic hot-reload of weights without restarting the API (watch `./model` directory).
- Optimised multi-stage Docker build reducing final image size.

Best of Luck!

Team Electronix AI