Love it — you’ve got a legit prototype. Here’s a crisp “product-style” feature list first, then a dead‑simple, free deployment guide for Streamlit. After that, I’ll note how we’d port this to a React/Next.js front end.

**Features (organized like a product page)**

**1) Forecast Panel**

* **Next‑day close prediction** (LSTM on last 60 trading days).
* **Direction call** (UP/DOWN) with delta vs. last close.
* **Confidence band** (data‑driven: rolling RMSE).
* **Trade / No‑Trade signal** (skips trades when the move is within expected error).

**2) Tracking & Reconciliation**

* **Predict** button logs a forecast to session history.
* **Reconcile Actuals** fills in real closes later and computes:
  + Absolute / % error
  + Directional hit/miss
* **CSV export** of the prediction history.

**3) Backtesting (Recent Window)**

* Rolling, leakage‑free **7–60 trading day** backtest (configurable).
* **KPIs:** MAE, RMSE, MAPE, Directional Accuracy.
* **Benchmark:** naïve “yesterday = tomorrow” MAE/RMSE for context.
* **Charts:** Predicted vs Actual, **rolling Directional Accuracy** & **rolling RMSE**.

**4) P/L Simulation (Trader‑friendly)**

* **Long on UP / Short on DOWN** P/L in points and %.
* **Costs-aware** net P/L (per‑side cost input, round‑trip applied).
* **Trade filter** (No‑Trade during low‑edge conditions).
* **Cumulative P/L** and **Cumulative Return** charts.
* CSV export of backtest with P/L columns.

**Zero-cost deployment (Streamlit Community Cloud)**

**A) Make sure repo has these files (top level)**

* app.py (your Streamlit app)
* best\_lstm\_model.h5, scaler.save (model + scaler)
* requirements.txt (see below)
* runtime.txt (Python version)
* .streamlit/config.toml (theme/layout, optional but nice)
* .gitignore (you already added one)

**requirements.txt (lean + compatible)**

streamlit>=1.36

yfinance>=0.2.40

pandas>=2.0

numpy>=1.24

scikit-learn>=1.4,<1.6

tensorflow-cpu==2.15.\* # CPU build is smaller/lighter for free tier

altair>=5.3

joblib>=1.3

If TensorFlow build times out, switch to:

* tensorflow==2.15.\* (slower install, bigger image)  
  or try **Hugging Face Spaces** (CPU Image) as a fallback host.

**runtime.txt**

python-3.10

(3.10 is usually the safest with TF‑2.15 on free hosts.)

**.streamlit/config.toml (optional polish)**

[theme]

base="dark"

primaryColor="#4ade80"

**B) Push to GitHub**

Commit & push all the above to your GitHub repo (model files included).

**C) Deploy on Streamlit Community Cloud**

1. Go to [https://share.streamlit.io](https://share.streamlit.io/)
2. **New app** → select your repo, branch, and app.py.
3. No secrets needed (we use Yahoo Finance).
4. Click **Deploy**.
5. First build can take a while; after that it should be quick.

**Common gotchas**

* **Large wheels / build timeouts:** TF is heavy. Use tensorflow-cpu==2.15.\*.
* **Python mismatch:** If build fails, ensure runtime.txt is exactly python-3.10.
* **Model paths:** We load from relative paths; keep .h5 & scaler.save in the repo root.

**Alternatives if Streamlit Cloud struggles**

* **Hugging Face Spaces (Gradio/Streamlit template):**
  + Repo with app.py, requirements.txt.
  + Create Space → choose **Streamlit** → upload repo → it builds automatically.
* **Render Free Web Service:**
  + New **Web Service** → pick repo → Start command: streamlit run app.py --server.port $PORT --server.address 0.0.0.0
  + Use free instance (spins down on idle).
* **Railway**:
  + New Project → GitHub repo → set start command same as above.

**Next step: React/Next.js version (high-level plan)**

We’ll reuse your model service and wrap it in a modern UI:

**Option A (fastest):** Keep Streamlit as the **model API**

* Add one endpoint (e.g., /predict) using Streamlit’s st.experimental\_connection or switch to a tiny **FastAPI** microservice that loads the model once and exposes POST /predict and POST /backtest.
* Next.js (minimal Tailwind + shadcn/ui) fetches JSON from that endpoint.
* Pros: super quick to stand up, same Python code, modern UI.

**Option B (clean arch):** Split completely

* **Service:** Python **FastAPI** (model, scaler, yfinance), deploy to Render/Railway.
* **Frontend:** Next.js (App Router, Tailwind, shadcn/ui), deploy to Vercel (free).
* Endpoints:
  + POST /predict → returns { predicted\_close, band, direction, signal }
  + POST /reconcile → returns updated rows
  + POST /backtest?horizon=7 → returns KPIs + rows
* Pros: production‑ready separation of concerns, easy to scale later.

If you want, I can scaffold the **FastAPI service** (predict + backtest endpoints) and a **Next.js page** with a clean, minimal layout so you can iterate fast.