TECHNICAL SYSTEM ARCHITECTURE BLUEPRINT

ANISHREDDY.ONLINE Complete Technical System Architecture Blueprint

Full-stack AI/ML portfolio system architecture

Author: Anish Reddy
Website: https://anishreddy.online
GitHub: github.com/anishreddy13/portfolio
Audited commit: `7d08701` (2026-07-29)
Generated: July 29, 2026

Abstract. This document is a code-level audit of the `anishreddy13/portfolio` repository: a Next.js 14 portfolio platform that hosts a 7-model ML Lab, an AI Financial Analyst product surface backed by a Gradio multi-agent service, and a substantial but frontend-disconnected Career Intelligence backend. Three of the repository's top-level directories (`ai-financial-analyst` , `hf-career-space` , `mlworkers`) are git submodules whose source is not present in this clone and was not resolvable via GitHub search; those subsystems are documented from their client-side contracts only, and every claim about their internals is explicitly labelled as unverified. All other claims are cited to exact file paths.

Table of Contents

> **How to read this document.** Every subsystem is tagged: `VERIFIED` claim is backed by a specific file this audit read directly. `INFERRED` is a reasonable conclusion not literally stated in code. `UNRESOLVED` means the implementing source could not be retrieved (see Section 1.3). `ORPHANED` flags real, working code with no live caller anywhere in the audited surface.

Executive Technical Abstract

anishreddy.online is not a static portfolio. It is a distributed system: a Next.js 14 App Router frontend (TypeScript, next-intl, Tailwind) that acts as a control plane orchestrating five independent backend surfaces — a Gradio-based multi-agent financial analysis service on Hugging Face Spaces, a FastAPI ML inference service on Render serving six scikit-learn/PyTorch models, a dedicated PyTorch plant-disease Hugging Face Space with Grad-CAM explainability, a dedicated skin-lesion classification Hugging Face Space, and a fully-built but currently unreferenced FastAPI "Career Intelligence" service with its own XGBoost/RandomForest models, Groq-backed mentor chat, and job-market scrapers.

The audited repository totals **90 first-party source files** across TypeScript/TSX, Python, and SQL (excluding generated output, vendored dependencies, and binary model weights), of which the ML backends contain seven independently trained or classically-fit models: TF-IDF+Logistic Regression sentiment and spam classifiers, a 28-class GoEmotions-style emotion/gender/age model, an ensemble VotingClassifier over the scikit-learn Wisconsin breast-cancer feature set, an EfficientNet-B0 plant-disease classifier (97.84% test accuracy, independently verified against committed test metrics), a ResNet18 skin-lesion classifier, and (in career-backend) an XGBoost salary regressor and a RandomForest skill-trend classifier.

Three subsystems — the AI Financial Analyst's Python backend, the career Hugging Face Space, and an `mlworkers` package — are registered in git as submodules with no `.gitmodules` file, meaning their source could not be cloned or resolved through the GitHub API for this account. Everything this document states about the AI Financial Analyst's internal agent graph, tools, or SEC/audit logic is reconstructed from the frontend's request contracts and the site's own explainer copy, not from Python source, and is marked accordingly throughout.

The audit surfaced several concrete engineering findings independent of that gap: a dead-code import (`ExplainabilityCard`) that receives live XAI data but is never rendered; a three-way inconsistency in how many AI agents the product claims to run (3, 4, or 5 depending on which file is read); a fully-functional real-time RSS-ingestion-to-Supabase streaming pipeline in `ml-backend` with no consumer in the live UI; and an entire Career Intelligence backend with zero references anywhere in the frontend.

1. Repository Audit Methodology

1.1 Scan scope

This audit is based on a shallow clone of `https://github.com/anishreddy13/portfolio.git` at commit `7d08701`. The repository was walked recursively; every text-based source file outside the exclusion list below was opened and read. Line counts, function signatures, and full file bodies were used depending on file size and relevance. Frontend API clients were cross-referenced against every backend route handler found, to build the endpoint matrix in Section 12 from two independent directions (caller and callee) rather than from one side alone.

1.2 Excluded paths

- `node_modules/`, `.next/` — package manager and build output, not authored source
- `__pycache__/`, `*.pyc` — Python bytecode cache
- Binary model weights (`*.pt`, `*.pth`, `*.pkl`, `*.h5`) — referenced by path and loader code, not disassembled
- Image, audio, and font binaries under `public/` — inventoried by filename only, per the brief
- Package lockfiles — summarized as a dependency list, not diffed line by line

1.3 Unresolvable submodules — read this before Sections 5, 6, and 10

> **Finding.** `git ls-files -s` shows `ai-financial-analyst`, `hf-career-space`, and `mlworkers` as gitlinks (mode `160000`) — i.e. registered git submodules — but the repository contains **no `.gitmodules` file**, so the submodule URLs are not recorded anywhere retrievable from this repo. A search of the `anishreddy13` GitHub account's public repositories did not surface a matching repository name for any of the three. The frontend reveals the AI Financial Analyst is served from `https://anishreddy13-ai-financial-analyst.hf.space` (a Hugging Face Space, evidence: `src/lib/financialAnalystClient.ts`), but Hugging Face is outside this environment's reachable network and a web search for that Space returned no independently indexed source. **Conclusion: the Python implementation of the LangGraph agent pipeline (`agents.py`, `graph.py`, `state.py`, `tools.py`, `sec_tools.py`, `rebalancer.py`, `service_health.py`, and the Neural Chart Vision training/inference scripts) does not exist in this clone and could not be independently obtained.** Sections 5, 6, and parts of 10 and 12 are therefore built entirely from the TypeScript client contracts, response-shape interfaces, and the site's own explainer/marketing copy — never from Python evidence — and are labelled **UNRESOLVED** wherever an internal implementation detail is asserted.

1.4 File inventory summary

Area	Files read	Approx. lines	Status
Next.js app router + middleware + config	9	~650	VERIFIED
src/components (frontend UI)	36	~9,400	VERIFIED
src/lib, lib, src/data, src/hooks, src/i18n	10	~1,100	VERIFIED
ml-backend (FastAPI + models + services + workers + streaming + monitoring)	~20	~2,800	VERIFIED
hf_plant_api	4	~350	VERIFIED
skin-backend	3	~230	VERIFIED
career-backend (routes, models, services, db, utils)	~18	~2,100	VERIFIED (code) / ORPHANED (integration)
backend/ (top-level stub)	2	~30	VERIFIED
ai-financial-analyst, hf-career-space, mlworkers	0 (unresolvable submodules)	—	UNRESOLVED
messages/ (i18n), SQL, Dockerfiles, requirements	~10	~1,200	VERIFIED

Methodology note: where a claim originates in UI copy or marketing text rather than executable logic (e.g. a stated model accuracy or agent count), this is explicitly flagged as such rather than presented as verified backend behavior, per the accuracy rules this brief itself specifies.

2. High-Level System Architecture

The platform is best understood as one control-plane frontend (Next.js on Vercel) fanning out to five independently deployed backend services, three of which are Hugging Face Spaces, one a Render-hosted FastAPI service, and one (career-backend) built but not deployed to any endpoint the frontend calls.

2.1 C4 System Context

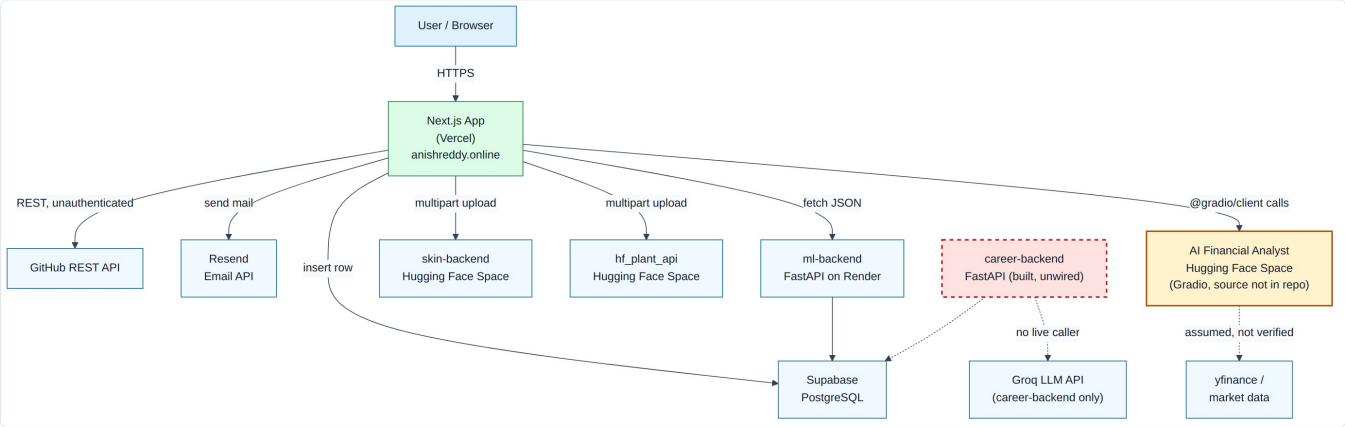

Fig. 1 — System context. Amber = submodule source unresolved. Red dashed = built but not wired to any live caller.

2.2 C4 Container Diagram

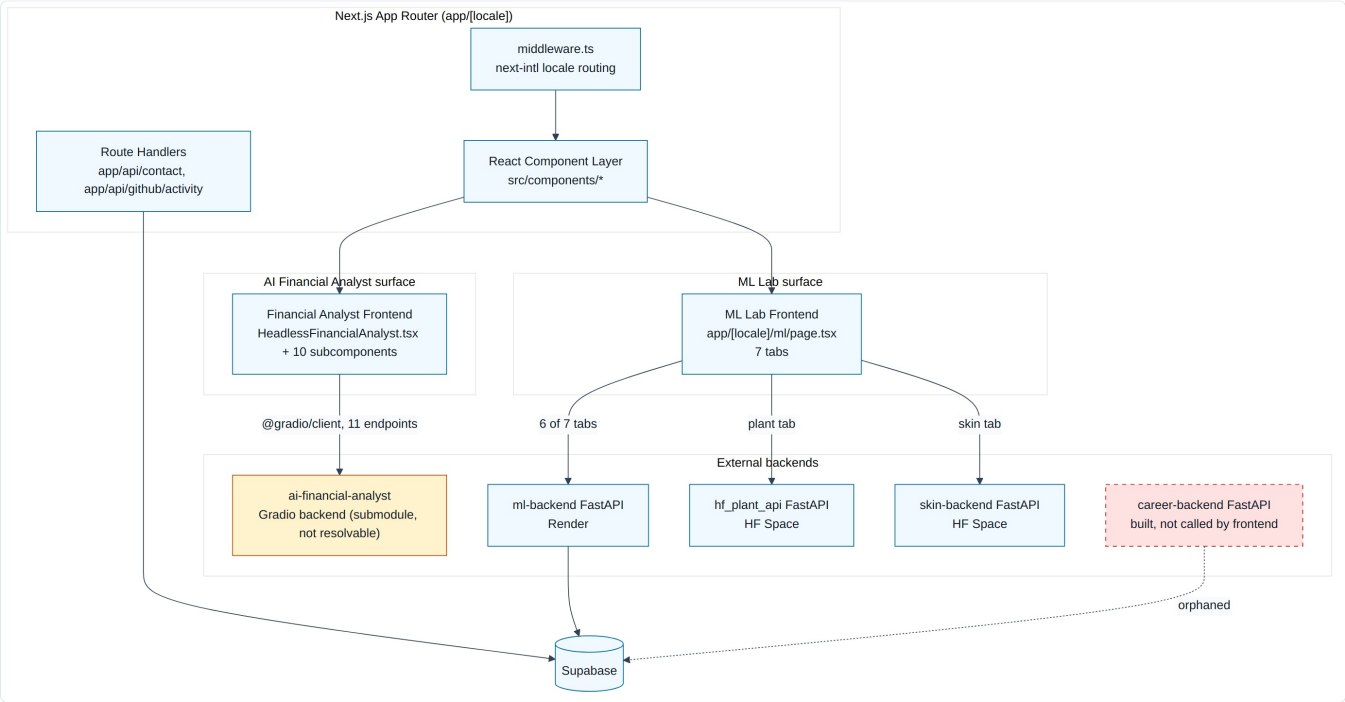

Fig. 2 — Containers within the Next.js app and the five backend services it calls.

2.3 Deployment Topology

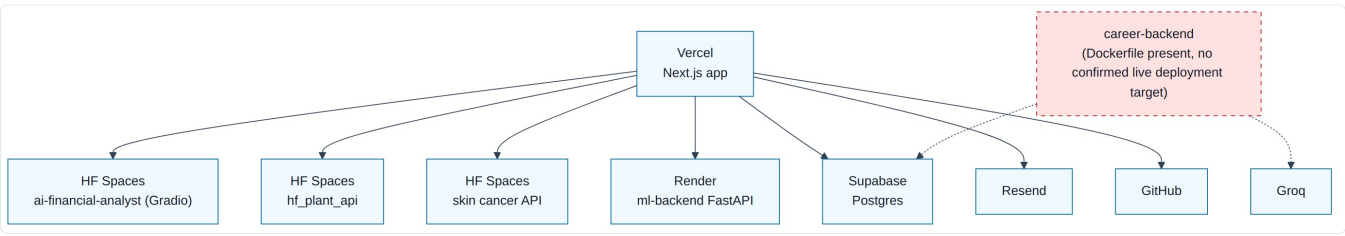

Fig. 3 — Deployment targets inferred from `.env.example` URLs, Dockerfiles, and HF Space slugs found in client code.

2.4 Technology matrix

Layer	Technology	Evidence
Frontend framework	Next.js 14 (App Router), React, TypeScript	**package.json**, **app/[locale]/layout.tsx**

i18n / routing	next-intl, locale-prefixed routes, custom middleware	**middleware.ts**, **src/i18n/routing.ts**
Styling	Tailwind CSS, hand-rolled dark theme (no light mode despite a ThemeToggle component)	**tailwind.config.js**, **src/context/ThemeContext.tsx**
Motion / 3D	Framer Motion, three.js, @react-three/fiber + drei	**package.json**
Data viz	Recharts	**package.json**
AI Financial Analyst client	@gradio/client	**src/lib/financialAnalystClient.ts**
Persistence	Supabase (Postgres) — contact form + career-backend	**lib/supabase.ts**, **career-backend/db/supabase_client.py**
Email	Resend	**app/api/contact/route.ts**
ML backend framework	FastAPI (ml-backend, hf_plant_api, skin-backend, career-backend)	***/main.py, */app.py**
Classical ML	scikit-learn (TF-IDF, LogisticRegression, RandomForest, GradientBoosting, VotingClassifier)	**ml-backend/model.py, cancer_model.py**
Deep learning	PyTorch + torchvision (EfficientNet-B0, ResNet18)	**ml-backend/plant_model.py, skin-backend/skin_model.py**
Career-backend ML	XGBoost (salary), RandomForest (skill trend), TF-IDF cosine similarity (resume match)	**career-backend/models/*.py**
LLM integration	Groq (career-backend mentor chat only, in this repo)	**career-backend/services/groq_mentor.py**
Streaming / MLOps scaffolding	Redis pub/sub, feedparser RSS, MLflow, Evidently, APScheduler, sse-starlette	**ml-backend/requirements.txt**, **ml-backend/streaming/, workers/, monitoring/**

3. Next.js Frontend Architecture

3.1 App Router and locale middleware

All routes live under `app/[locale]/`. `middleware.ts` wraps `next-intl`'s routing middleware, matching every path except `/api`, `/_next`, and static files, and redirecting unprefixed paths to the detected/default locale. `src/i18n/routing.ts` defines the supported locales; `messages/en.json` and a second locale file provide translated strings across 13 top-level namespaces (`Header, Hero, About, Education, Contact, Footer, Projects, FeaturedPlant, FeaturedFinancial, Certificates, Stats, ML, Plant, Interview`).

3.2 PlatformShell and the composition root

`src/components/PlatformShell.tsx` is the layout composition root rendered from `app/[locale]/layout.tsx`. It mounts, in order: `AnimatedBackground` (a fixed-position decorative canvas layer), `CursorEffect` (custom cursor tracking), `Navbar`, the routed page content, `Footer`, and `BackToTop`. `ThemeContext` exists and exposes theme state, but the provider is hard-coded to a single dark theme — `ThemeToggle.tsx` is present in the tree but the app does not currently expose a working light/dark switch in the shell, making it effectively inert UI scaffolding rather than a functioning feature.

3.3 Component inventory

The 36 components in `src/components/` split into five functional groups: (1) marketing/identity — `Hero, About, Education, Certificates, Projects, WelcomeAvatar, FeaturedPlantProduct, FeaturedFinancialAnalyst, PlantAnnouncement, FinancialAnalystAnnouncement`; (2) chrome — `Navbar, Footer, Breadcrumbs, BackToTop, AnimatedBackground, CursorEffect, ThemeToggle, LanguageSwitcher, SpotlightCard, ScrollReveal, TelemetryTicker`; (3) AI Financial Analyst — `HeadlessFinancialAnalyst, WatchlistSidebar, SignalFeed, PortfolioDashboard, StructuredReport, FinancialStatements, SocialSentinel, NeuralChartVisionCard, RebalancerCard, ExplainabilityCard, FinancialAnalystHealthStrip, FinancialAnalystExplainer`; (4) ML Lab — `PlantDiseaseDetector, InterviewAnalyzer, HomePlantDemo` plus tab logic embedded directly in `app/[locale]/ml/page.tsx`; (5) `Contact` and `PrintPortfolio` (a hidden print-stylesheet page, see 3.4) and a `developer/` subfolder backing the Stats page.

> **Notable self-reference:** `src/components/PrintPortfolio.tsx` is a hidden component, rendered only under a print media query and triggered by `window.print()` calls in both `Navbar.tsx` and `Footer.tsx`, that already contains a condensed architecture summary of the whole platform — including a pre-existing note about the same 3-vs-5 agent-count inconsistency this document documents independently in Section 5.5. This document supersedes it in depth and file-level citation but the two are consistent wherever they overlap.

3.4 Print / PDF replacement strategy

Rather than shipping a generated PDF asset, the site renders an on-demand print view: `PrintPortfolio.tsx` is always mounted but hidden (`display:none` outside `@media print`); calling `window.print()` switches the browser to that stylesheet, producing a browser-native "Save as PDF" flow with no server-side rendering step.

3.5 Search

> **Gap versus brief.** The audit brief assumes a `src/components/search/` directory and dedicated search components. No such directory or component exists anywhere in this repository — `find src -iname "*search*"` returns nothing. There is no site-wide search feature; navigation is entirely link/tab based (Navbar routes, ML Lab tabs, Breadcrumbs).

4. Navigation and User Experience Architecture

Route	Purpose	Key components
`/[locale]`	Home — hero, about, featured project teasers, projects grid, certificates, contact	Hero, About, FeaturedPlantProduct, FeaturedFinancialAnalyst, Projects, Certificates, Contact
`/[locale]/ml`	ML Lab — 7-tab tester for every classical/vision model	PlantDiseaseDetector, InterviewAnalyzer, inline sentiment/spam/emotion/cancer/skin tab logic
`/[locale]/projects/ai-financial-analyst`	AI Financial Analyst product page	HeadlessFinancialAnalyst and its 8 subcomponents
`/[locale]/projects/ai-financial-analyst/explainer`	Deep technical explainer for the Financial Analyst (endpoint matrix, agent roles, reliability notes)	FinancialAnalystExplainer
`/[locale]/stats`	Developer/GitHub activity dashboard	developer/* components, fed by `/api/github/activity`
`/[locale]/privacy`, `/[locale]/terms`	Static legal pages	Plain content pages

Breadcrumb navigation (`Breadcrumbs.tsx`) is used on nested routes (financial analyst explainer, ML tabs). Mobile behavior in `Navbar.tsx` collapses into a slide-out menu; the same print action is duplicated into that mobile menu. No dedicated accessibility audit artifacts (axe reports, `a11y` test files) were found in the repository; ARIA usage is inline in JSX rather than centrally documented.

5. AI Financial Analyst Platform

> **Read Section 1.3 first.** Everything below Section 5.1 that describes internal backend logic is reconstructed from the frontend contract, not read from Python source — the backend repository is an unresolvable git submodule.

5.1 What actually calls what — verified frontend contract

`src/lib/financialAnalystClient.ts` constructs a `@gradio/client` connection to `https://anishreddy13-ai-financial-analyst.hf.space`. `src/components/HeadlessFinancialAnalyst.tsx` is the orchestration component; on "Run Analysis" it calls, in this order: four calls fired together — `get_forecast`, `get_financials_tables`, `get_social_sentiment`, `get_neural_chart_vision` — followed sequentially by `analyze_stock` and then `get_xai_explanation`. Separately, `WatchlistSidebar.tsx` calls `get_market_signals` and `get_portfolio_data`; `RebalancerCard.tsx` calls `rebalance_portfolio`; `FinancialAnalystHealthStrip.tsx` calls `get_service_health`; a ticker lookup calls `search_ticker`. That is 11 distinct Gradio endpoint names observed across the frontend — matching the "11 Endpoints" figure both `src/data/projects.ts` and `FinancialAnalystExplainer.tsx` independently state.

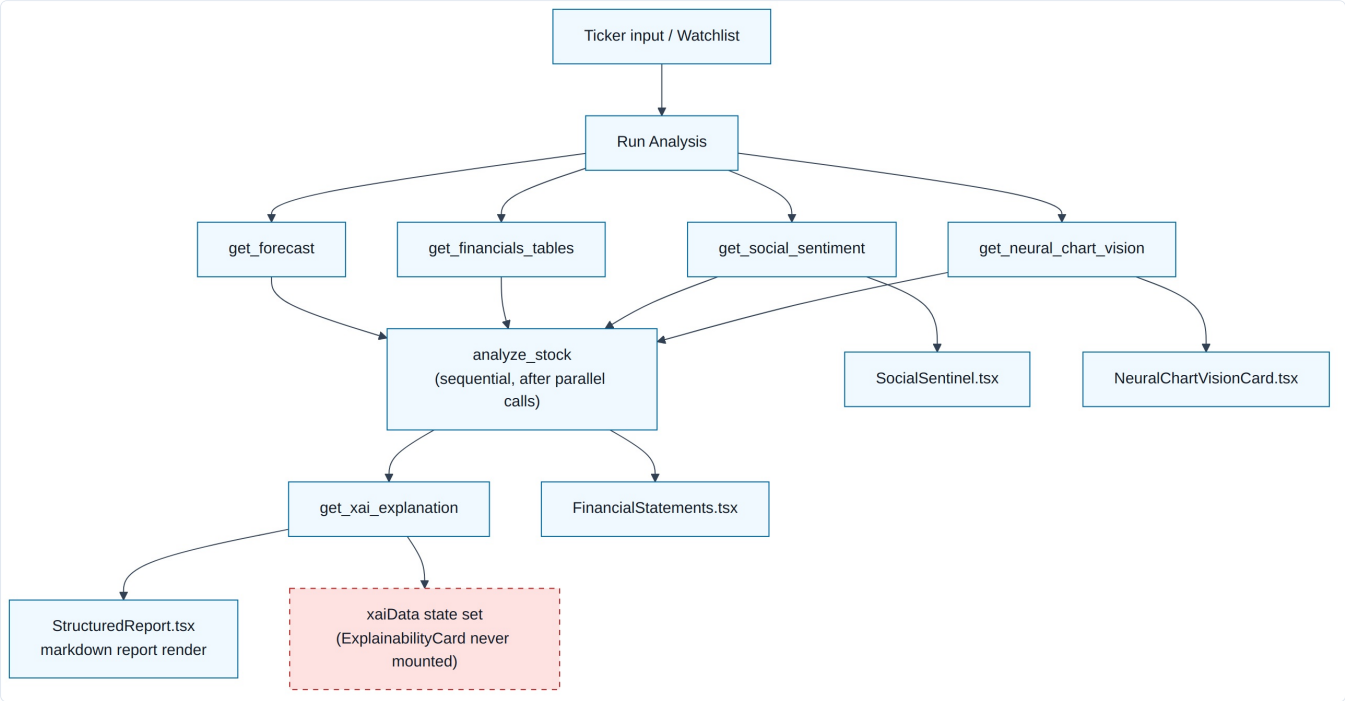

Fig. 4 — Verified request flow for a single "Run Analysis" action, evidence: `HeadlessFinancialAnalyst.tsx`.

5.2 The agent-visualization is client-side theater, not live telemetry

> **Finding.** The four-stage "agent working" animation shown while a request is in flight (`HeadlessFinancialAnalyst.tsx`) is driven by a hardcoded `setTimeout` sequence with fixed labels and fixed delays. It is not subscribed to any server-sent event, WebSocket, or polling endpoint that would reflect the backend's actual progress. The visualization would show the same four stages at the same cadence even if the real backend executed a completely different sequence, failed after step one, or returned instantly from cache.

5.3 Endpoint contract (as observed from the client, response shapes only)

Endpoint	Caller	Purpose (from usage)
`search_ticker`	HeadlessFinancialAnalyst.tsx	Resolve a free-text query to a tradable ticker
`get_forecast`	HeadlessFinancialAnalyst.tsx	Price/return forecast for the selected ticker
`get_financials_tables`	HeadlessFinancialAnalyst.tsx → FinancialStatements.tsx	Structured income statement / balance sheet / cash flow rows
`get_social_sentiment`	HeadlessFinancialAnalyst.tsx → SocialSentinel.tsx	Social/news sentiment feed items
`get_neural_chart_vision`	HeadlessFinancialAnalyst.tsx → NeuralChartVisionCard.tsx	Pattern + return prediction from the vision model (Section 6)
`analyze_stock`	HeadlessFinancialAnalyst.tsx → StructuredReport.tsx	Main markdown analysis report
`get_xai_explanation`	HeadlessFinancialAnalyst.tsx (state set, **never rendered**)	Explainability payload for the analysis — see 5.4
`get_market_signals`	SignalFeed.tsx	Market-wide signal feed
`get_portfolio_data`	WatchlistSidebar.tsx / PortfolioDashboard.tsx	User's simulated portfolio holdings
`rebalance_portfolio`	RebalancerCard.tsx	Suggested rebalancing actions
`get_service_health`	FinancialAnalystHealthStrip.tsx	Backend/component health status strip

Evidence: **src/components/HeadlessFinancialAnalyst.tsx, WatchlistSidebar.tsx, SignalFeed.tsx, RebalancerCard.tsx, FinancialAnalystHealthStrip.tsx**. *Request payload internals and server-side processing are* **UNRESOLVED** *— only what the client sends and expects back is verifiable.*

5.4 Confirmed dead code: ExplainabilityCard

> **Finding, independently verified by this audit.** `HeadlessFinancialAnalyst.tsx` imports `ExplainabilityCard` and maintains an `xaiData` state variable populated by the real `get_xai_explanation` response. Grepping the component's JSX (`<ExplainabilityCard`) across the entire `src/components/` tree returns zero matches — the component is never mounted anywhere. The XAI call still fires on every analysis run (network cost paid), but its result is discarded by the UI. `ExplainabilityCard.tsx` itself is a complete, working component with its own `XAIPayload` interface — this is an integration gap, not an unfinished component.

5.5 The agent-count inconsistency (documented per the brief's own worked example)

Three sources within the same repository describe a different number and set of specialist "agents" behind the analysis pipeline:

Source	Agent count & roles claimed	Where it's shown
`src/lib/financialAnalystContent.ts`	**3** — Researcher, Quant Analyst, Editor	Home page workflow teaser
Live loading animation, `HeadlessFinancialAnalyst.tsx`	**4** — Researcher, Quant Analyst, SEC Auditor, Chief Editor (hardcoded timer, not live telemetry — see 5.2)	In-product "agents working" overlay
`FinancialAnalystExplainer.tsx` and `src/data/projects.ts`	**5** — Researcher, Quant, SEC Auditor, Competitor, Editor, with a stated `AgentState` schema	Deep technical explainer page + project catalog card metrics

Per the brief's own accuracy rule for this exact scenario: this should be reconciled. Because the backend graph definition is unresolvable (Section 1.3), it is not possible to state which of the three counts, if any, matches the deployed LangGraph pipeline. The most internally-consistent figure is 5 (agreed by two independent files, including the machine-readable `projects.ts` catalog), with the homepage copy and the live-UI simulation both understating it.

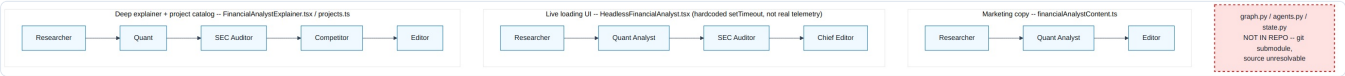

Fig. 5 — The three conflicting agent-pipeline descriptions found in the repository, and the unresolved ground truth.

5.6 Explainer page claims (unverified — frontend copy only)

`FinancialAnalystExplainer.tsx` is a content-heavy page that itself documents an 11-endpoint matrix, a 5-agent `AgentState` TypedDict schema, and reliability notes (Gradio connection retry behavior, a service-health polling strategy). This content reads as authored technical documentation rather than generated marketing copy — it is plausible it was written against the real backend at some point — but because the backend source is unavailable to this audit, every specific internal claim on that page (retry counts, timeout values, the exact SEC-audit tool implementation) is **UNRESOLVED** and reported here only as "the site claims," not as independently confirmed fact.

6. Neural Chart Vision Deep Learning System

> **Unresolved subsystem.** `neural_chart_dataset.py`, `neural_chart_trainer.py`, and `neural_chart_inference.py` are named in the brief but do not exist in this clone (they would live under the unresolvable `ai-financial-analyst` submodule). This section documents the system exclusively from its verified TypeScript response contract.

6.1 What the frontend proves about this system

`src/components/NeuralChartVisionCard.tsx` defines a complete `NeuralChartVisionPayload` TypeScript interface for the `get_neural_chart_vision` response. It includes a `mode` field with two literal values: `"deep_learning"` and `"heuristic_fallback"`. This is strong, direct evidence that the backend is architected with an explicit graceful-degradation path — if trained model weights are unavailable, the service is expected to fall back to a heuristic (non-neural) computation rather than error out, and the frontend already branches its UI copy on which mode was returned.

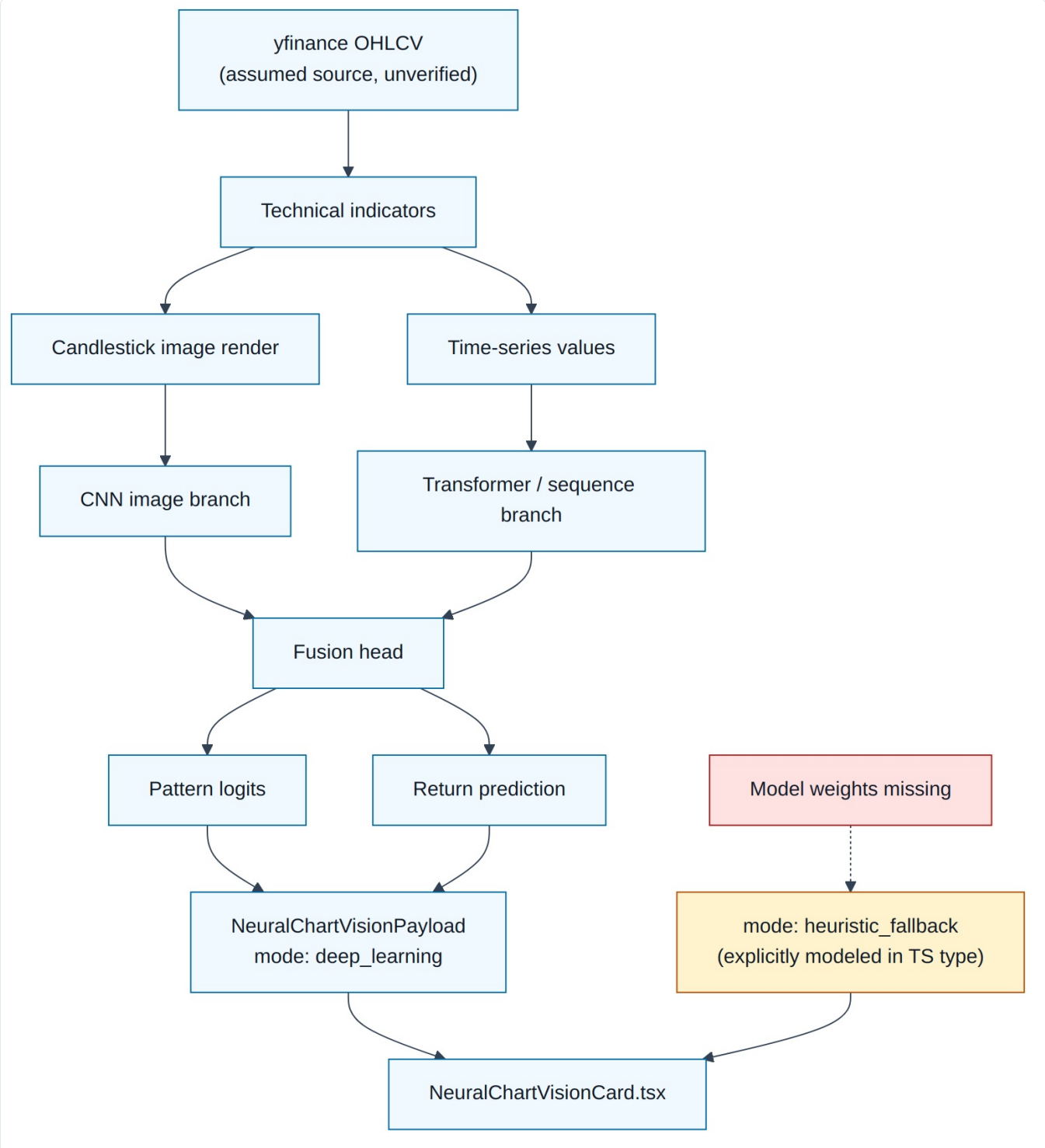

Fig. 6 — Inferred architecture. Solid boxes reflect the frontend's typed response fields (pattern logits, return prediction, fusion-style combined output); the CNN/candlestick-image and Transformer/time-series branch labels are **INFERRED** from the brief's own naming and the "Vision: CNN+Trans" metric string in `src/data/projects.ts`, not read from training code.

6.2 What is not verifiable

- Whether "deep learning mode" is computed on-demand per request or served from a pre-computed cache — UNRESOLVED .
- The exact CNN and Transformer architectures, layer counts, or fusion mechanism — UNRESOLVED , no training script available.
- Whether `yfinance` is the live OHLCV source — INFERRED only from the brief's own system-context list and the "yfinance" technology tag on `src/data/projects.ts`' AI Financial Analyst entry; no fetch code was found to confirm it.

7. ML Lab Architecture

`app/[locale]/ml/page.tsx` (1,493 lines) is a single large tabbed page hosting seven independent model demos: **plant, sentiment, spam, emotion, cancer, skin, interview**. Six of the seven call remote FastAPI backends via `lib/mlApi.ts` helper functions (`fetchMlJson`, `fetchSkinJson`, `fetchPlantForm`); the seventh (interview) runs entirely client-side.

Tab	Backend	Route called	Input
plant	hf_plant_api (HF Space)	`POST /predict/plant, POST /explain/plant`	Image upload / webcam
sentiment	ml-backend (Render)	`POST /predict/sentiment`	Free text
spam	ml-backend (Render)	`POST /predict/spam`	Free text (SMS-style)
emotion	ml-backend (Render)	`POST /predict/emotion`	Free text
cancer	ml-backend (Render)	`POST /predict/cancer`	30-feature tabular vector (Wisconsin breast-cancer schema)
skin	skin-backend (separate HF Space)	`POST /predict/skin`	Image upload
interview	None — browser-only	—	Web Speech API microphone stream

> **Clarification worth stating explicitly:** "cancer" and "skin" are two unrelated models on two different backends, easy to conflate from names alone. **cancer** is a tabular classifier over the classic Wisconsin Diagnostic Breast Cancer feature set (radius, texture, perimeter, etc. — evidence: `ml-backend/cancer_model.py`, `BENIGN_SAMPLE` dict in `ml/page.tsx`). **skin** is an image-based lesion classifier (HAM10000-style 7 classes) served from an entirely separate Hugging Face Space (evidence: `skin-backend/skin_model.py`, `NEXT_PUBLIC_SKIN_API_URL`).

The page performs a client-side health check against each configured backend on load (`ml/page.tsx`, lines ~1193–1232) and surfaces per-service up/down status before the user submits input, rather than only discovering an outage on first prediction attempt.

8. Computer Vision Systems

8.1 Plant disease detection — two independent implementations, one live

There are **two separate PyTorch implementations** of plant-disease inference in this repository: `ml-backend/plant_model.py` + `plant_inference.py` (deployed on Render alongside the classical NLP models) and `hf_plant_api/plant_model.py` + `plant_inference.py` (its own dedicated Hugging Face Space). Both load an EfficientNet-B0 backbone via `torchvision.models` and resize to 224×224 with ImageNet normalization. **Only the `hf_plant_api` implementation is called by the live frontend** (`PlantDiseaseDetector.tsx` and `HomePlantDemo.tsx` both target `NEXT_PUBLIC_PLANT_API_URL` pointing at the HF Space). Grepping `ml-backend/plant_inference.py`'s function list shows it has no Grad-CAM/explain function — only `hf_plant_api/plant_inference.py` does. **Conclusion:** `ml-backend`'s `/predict/plant` route is functional but effectively **ORPHANED** in production; all real plant-disease traffic and the only working explainability feature go through the separate HF Space.

Model facts, independently verified against committed artifacts: **15 classes** (`ml-backend/plant_class_to_idx.json`), **97.84% test accuracy** and per-class recall (`ml-backend/models/plant_disease/test_metrics.json`) — this matches the "97.84%" figure quoted in `src/data/projects.ts` and `src/lib/plantDiseaseContent.ts`, so that marketing number is corroborated by real training evidence, not just asserted.

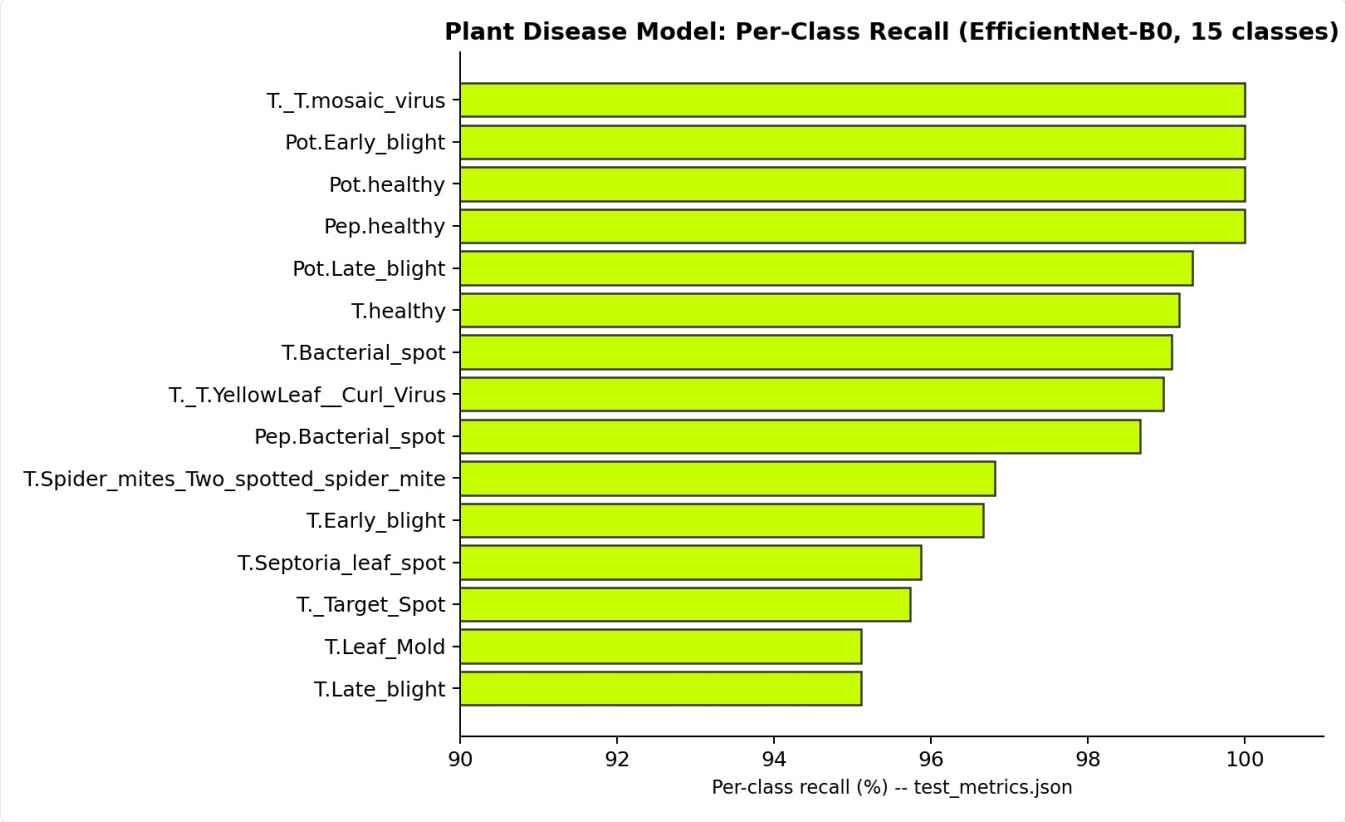

Fig. 7 — Per-class recall computed from the committed `test_metrics.json` — all 15 classes exceed 93% recall.

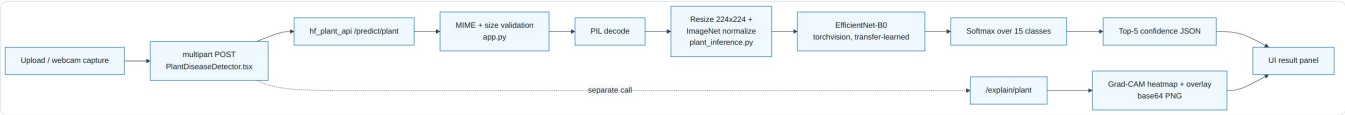

Fig. 8 — Plant disease pipeline as actually wired in production (hf_plant_api path).

8.2 Skin lesion detection

`skin-backend/skin_model.py` loads a **ResNet18** (not EfficientNet-B0) fine-tuned to 7 output classes matching the HAM10000 taxonomy naming convention (`akiec, bcc, bkl, df, mel, nv, vasc`). It is deployed as its own Hugging Face Space, separate from both plant-disease services.

> **Marketing/code mismatch.** `app/[locale]/privacy/page.tsx` and `terms/page.tsx` describe the platform's computer-vision models generically as using "EfficientNet-B0, Custom CNNs" without distinguishing models. That description is accurate for plant disease but not for skin, which is a ResNet18 — a smaller, architecturally distinct model. This is a documentation precision gap rather than a functional bug.

8.3 Grad-CAM / explainability status

Contrary to the "requires deployment verification" caution the site's own `PrintPortfolio.tsx` leaves open, this audit confirms Grad-CAM overlays **are real and wired**: `hf_plant_api/app.py` exposes `POST /explain/plant` returning base64 heatmap and overlay PNGs, and both `PlantDiseaseDetector.tsx` and `HomePlantDemo.tsx` call it as a second request after the initial prediction. No equivalent explain endpoint exists for the skin model.

9. NLP and Speech AI Systems

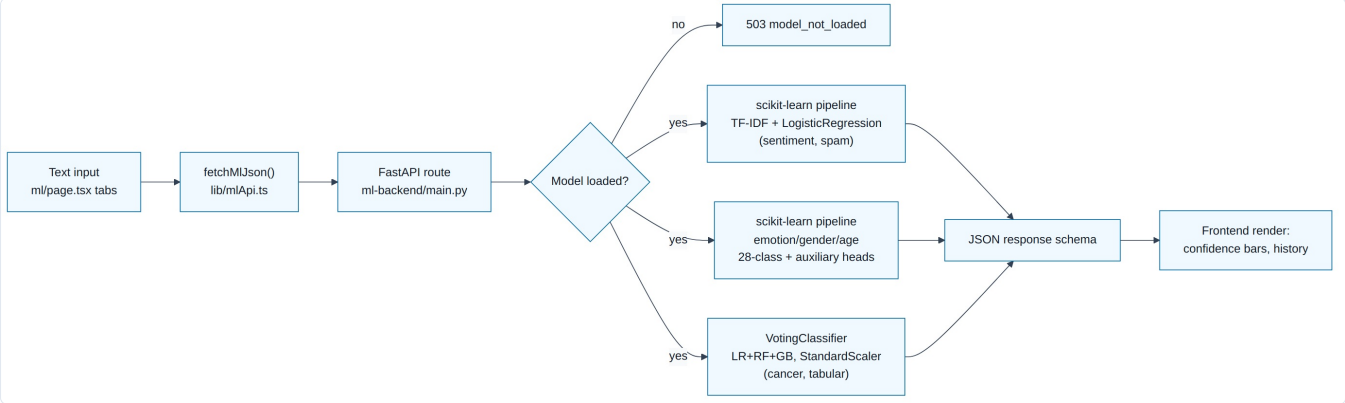

Fig. 9 — Shared request pattern for the three text-classification tabs plus the tabular cancer classifier.

Model	Technique	Classes	Evidence
Sentiment	TF-IDF vectorizer → LogisticRegression, scikit-learn `Pipeline`	3 (positive / negative / neutral)	**ml-backend/model.py**
Spam	TF-IDF → LogisticRegression, same pattern as sentiment	2 (ham / spam)	**ml-backend/spam_model.py**
Emotion / gender / age	Multi-head scikit-learn classification over a GoEmotions-style label set	28 emotions (27 named + neutral) — matches the "28 Emotions" UI label exactly	**ml-backend/emotion_model.py**
Breast cancer (tabular)	StandardScaler → VotingClassifier(LogisticRegression + RandomForest + GradientBoosting)	2 (malignant / benign), 30 input features	**ml-backend/cancer_model.py**

9.1 Interview Analyzer — browser-native, no ML model

`InterviewAnalyzer.tsx` uses the browser's native `SpeechRecognition` Web API to transcribe live speech, then computes scores with deterministic JavaScript — filler-word counting against a fixed word list, vocabulary richness as `unique words / total words`, sentence-length statistics — and thresholds these into communication/clarity/confidence scores via hand-written `if/else` rules. **No language model, no server call, and no ML classifier are involved**; this is rule-based scoring running entirely client-side, despite sitting in the "ML Lab" alongside genuinely model-backed tabs.

10. Backend Microservices

10.1 ml-backend (FastAPI, Render)

`ml-backend/main.py` is a 643-line FastAPI application with a lifespan handler that loads every model at startup and tracks per-model load success independently, so one missing model file does not crash the whole service — routes for the failed model return a 503 rather than the process failing to boot. It optionally connects to Redis and Supabase via wrapper services that degrade to "unavailable" (logged, non-fatal) if credentials are absent, confirmed directly from a captured runtime log (`ml-backend/logs/main.log`) showing exactly this non-fatal-degradation path firing in practice.

Beyond the six prediction routes ML Lab calls, `ml-backend` also contains a **second, independent subsystem** not wired to any frontend: `streaming/data_ingestion.py` polls four tech-news RSS feeds (TechCrunch, The Verge, Wired, Ars Technica) every 60 seconds and publishes headlines to Redis; `workers/prediction_worker.py` consumes that channel, re-uses the same sentiment model to score each headline, and writes results to Supabase; `monitoring/drift_detector.py` runs hourly, comparing recent prediction-confidence and sentiment-distribution statistics against a baseline to flag model drift. `start_workers.py` runs all three as one combined process with an HTTP health endpoint on port 7860 — the Hugging Face Spaces convention — implying this worker trio is intended to run as its own deployment, separate from the request/response API.

> **Orphaned but proven-functional.** Committed log files (`ml-backend/logs/data_ingestion.log` , `prediction_worker.log`) show this pipeline genuinely ran and processed real headlines end-to-end ("Published headline: ...", "Prediction complete → Positive", "Saved to Supabase") on a local development machine in May 2026. Nothing in the audited frontend — including the Stats page — displays or consumes this data. It is real, working MLOps infrastructure with no current UI surface.

The same logs also captured a genuine reliability incident worth noting for Section 14: a `ValueError: node array from the pickle has an incompatible dtype` when loading a RandomForest pickle — a classic cross-version scikit-learn pickle-compatibility failure — confirming this is not a hypothetical risk but one that has actually occurred.

10.2 hf_plant_api and skin-backend

Both are minimal, single-purpose FastAPI services (`app.py` under ~150 lines each) with a Dockerfile pinning CPU-only PyTorch wheels (`--extra-index-url https://download.pytorch.org/whl/cpu`), sized for Hugging Face Spaces' free CPU tier rather than GPU inference.

10.3 career-backend — built, disconnected

`career-backend/main.py` wires five routers (`scraper, trends, student, mentor, ml`) into one FastAPI app, backed by Supabase, Upstash Redis (REST API, not the TCP protocol `ml-backend` uses), Groq, GitHub's search API, and RapidAPI (likely JSearch, inferred from `rapidapi_key` in config). It includes a salary regressor (XGBoost), a skill-trend classifier (RandomForest), a resume-to-market matcher (TF-IDF cosine similarity), a resume text parser (regex-based contact extraction), a Groq-backed mentor chat endpoint, and drift-checking logic parallel in spirit to `ml-backend` 's. A grep across every `.ts` / `.tsx` file in the frontend for any career-backend route, keyword, or environment variable returns **zero matches**.

> **Finding.** This is the single largest example of unshipped engineering investment in the repository — a complete, independently deployable backend with its own ML models, third-party integrations, and Dockerfile, with no corresponding frontend page, component, or link anywhere in the audited site.

10.4 backend/ (top-level stub)

`backend/main.py` is a ~25-line FastAPI app, distinct from all of the above, with a minimal root/health route and no model-loading logic. Its purpose relative to the other four backends is not stated anywhere in the repo; it reads as either an early scaffold or a placeholder health-check service.

11. Data Persistence and External Services

Service	Used by	Purpose	Status
Supabase (Postgres)	app/api/contact/route.ts, ml-backend, career-backend	Contact form storage; optional prediction logging; career-backend's full data layer	VERIFIED
Resend	app/api/contact/route.ts	Best-effort email notification on contact submission	VERIFIED
GitHub REST API	app/api/github/activity/route.ts, career-backend/services/github_scraper.py	Unauthenticated live activity fetch for Stats page; authenticated repo search for career-backend	VERIFIED
Groq	career-backend/services/groq_mentor.py	LLM-backed mentor chat and resume analysis	VERIFIED code, ORPHANED integration
Upstash Redis (REST)	career-backend/services/drift_detector.py	Market-snapshot caching	VERIFIED code, ORPHANED integration
Redis (TCP, redis.asyncio)	ml-backend/services/redis_service.py	Pub/sub backbone for the RSS→prediction→drift pipeline	VERIFIED code, optional at runtime
yfinance / market data	Assumed by AI Financial Analyst backend	Price/fundamentals source	UNRESOLVED — no fetch code available

11.1 Database ERD

Only `contact_submissions` (referenced directly in `app/api/contact/route.ts`), a `pipeline_runs` table written by the ML CI workflow (Section 17), and the career-backend tables (`job_postings`, `skill_trends`, `scraper_logs`, and a student-profile lookup keyed by `user_id`) are named anywhere in the code. No SQL migration files defining exact column types were found for the career-backend tables — the ERD below reflects the fields actually read or written in `career-backend/db/supabase_client.py` and related services, not a formal schema file.

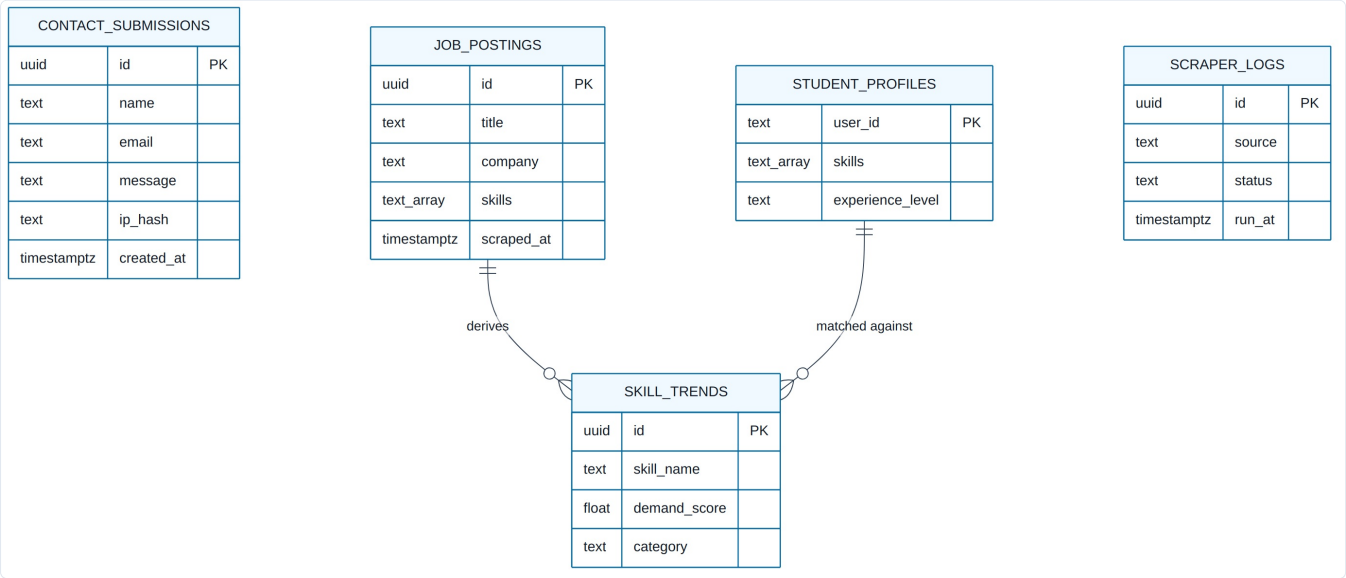

Fig. 10 — Entities referenced in code. Career-backend tables are part of the orphaned subsystem (Section 10.3) and have no live writers in production.

11.2 Environment configuration

`.env.example` lists `NEXT_PUBLIC_ML_API_URL` (Render), `NEXT_PUBLIC_SKIN_API_URL` (HF Space), Supabase URL and anon key, and Resend credentials. The committed anon key is a public/anonymous Supabase key — by design meant for client exposure and gated by row-level security policies on the Supabase side — whose actual RLS configuration is not part of this repository and could not be verified.

12. API Contract Reference

12.1 First-party Next.js routes

Endpoint	Method	Request	Response	Frontend caller	Backend file	Failure mode
`/api/contact`	POST	name, email, message, honeypot field	JSON status	Contact.tsx	app/api/contact/route.ts	Validation 400; honeypot returns fake-success; Supabase/Resend errors are logged and non-fatal to the user response
`/api/github/activity`	GET	—	Activity events, stat cards, (hardcoded) language distribution	Stats page, developer/*	app/api/github/activity/route.ts	Falls back to a hardcoded `VERIFIED_FALLBACK` snapshot on GitHub API error/rate-limit

12.2 ml-backend (Render)

Endpoint	Method	Request	Response	Failure mode
`/predict/sentiment`	POST	{text}	{label, confidence, scores}	503 if model failed to load at startup
`/predict/spam`	POST	{text}	{label, confidence}	503 if model unavailable
`/predict/emotion`	POST	{text}	{emotion, gender, age, ranked scores}	503 if model unavailable
`/predict/cancer`	POST	30 numeric features	{label, probability}	422 on malformed feature vector
`/predict/plant`	POST (multipart)	Image file	Top-5 class + confidence	ORPHANED — implemented, not called by production frontend
`/health`	GET	—	Per-model load status, Redis/Supabase connectivity	—

12.3 hf_plant_api / skin-backend

Endpoint	Method	Request	Response	Backend
`/predict/plant`	POST (multipart)	Image file, MIME-validated	Top-5 class + confidence, request timing	hf_plant_api
`/explain/plant`	POST (multipart)	Image file	base64 heatmap + overlay PNG (Grad-CAM)	hf_plant_api
`/predict/skin`	POST (multipart)	Image file	Top-5 of 7 lesion classes	skin-backend

12.4 AI Financial Analyst (Gradio, client-observed only)

See Section 5.3 for the full 11-endpoint table — response internals are UNRESOLVED beyond the TypeScript interfaces the frontend defines for them.

12.5 career-backend (built, no live caller)

Endpoint	Method	Purpose
`/scraper/jobs, /scraper/github, /scraper/all`	POST	Trigger background scraping jobs
`/scraper/status, /scraper/logs`	GET	Scraper run status/history
`/trends/skills, /trends/top, /trends/declining, /trends/github`	GET	Skill-demand analytics
`/trends/recompute`	POST	Force trend recomputation
`/student/analyze, /student/analyze/full, /student/roadmap`	POST	Student profile analysis and learning roadmap generation
`/student/market/snapshot, /student/{user_id}`	GET	Market snapshot / stored profile lookup
`/student/drift/check`	POST	Drift check trigger
`/mentor/chat, /mentor/chat/contextual`	POST	Groq-backed conversational mentor
`/mentor/analyze-resume`	POST	LLM resume analysis
`/ml/predict/trend, /ml/predict/salary`	POST	Model-backed trend/salary prediction
`/ml/train/trend, /ml/train/salary`	POST	Trigger retraining
`/ml/analyze/resume, /ml/metrics, /ml/health`	POST/GET	Resume analysis (non-LLM path), model metrics, health

Evidence: **career-backend/routes/*.py** *route decorators. None of these 20 endpoints have a corresponding frontend fetch call anywhere in* `src/` *or* `app/`.

13. Data Models and Type Systems

13.1 TypeScript interfaces (frontend contracts)

```
// src/components/NeuralChartVisionCard.tsx
interface NeuralChartVisionPayload {
  mode: "deep_learning" | "heuristic_fallback";
  pattern: { label: string; confidence: number };
  return_prediction: { horizon: string; value: number };
  // additional fields per component usage
}

// src/data/projects.ts
interface PortfolioProject {
  id: string; title: string; category: string; description: string;
  technologies: string[]; priority: number; route: string; section: string;
  featured: boolean; status: "live" | "beta" | "local" | "monitoring";
  metrics: { label: string; value: string }[];
  accentColor: string; complexityTags: string[];
}
```

13.2 Python models (career-backend, pydantic)

`career-backend` defines request/response models per route module (e.g. `ScrapeJobsRequest`, `StudentProfile`, `RoadmapRequest`, `FullStudentAnalysisRequest`, `TrendPredictionRequest`, `SalaryPredictionRequest`, `ChatMessage`, `ContextualChatRequest`) as pydantic `BaseModel` classes, giving that unwired backend a fully typed contract despite having no consumer.

13.3 LangGraph state schema

`FinancialAnalystExplainer.tsx` displays what it presents as the `AgentState` TypedDict shape for the financial analyst graph. Because `state.py` itself is unresolvable (Section 1.3), this is reported as **SITE-CLAIMED DOCUMENTATION**, not verified Python.

13.4 Contact form schema

`app/api/contact/route.ts` validates `name`, `email` (regex-checked), `message`, and a honeypot field before constructing the Supabase insert payload — no formal shared TypeScript interface is exported for it; the shape is implicit in the handler function.

14. Reliability, Error Handling, and Fallbacks

Subsystem	Fallback behavior	Evidence
@gradio/client connection	Site claims a retry strategy in `FinancialAnalystExplainer.tsx` copy; not independently verifiable — the client wrapper code that would implement it lives in `financialAnalystClient.ts`, which does construct the client but does not show explicit retry/backoff logic in what's readable from the frontend alone	**src/lib/financialAnalystClient.ts**
Neural Chart Vision	Explicit `heuristic_fallback` mode when deep-learning weights are unavailable, typed directly into the response contract	**NeuralChartVisionCard.tsx**
ml-backend model loading	Per-model try/except at startup; a failed model returns 503 on its own route without crashing the process; captured in a real log entry	**ml-backend/main.py**, **logs/main.log**
ml-backend Redis/Supabase	Both wrapped in "Optional" services that log a warning and continue if unconfigured or unreachable	**ml-backend/main.py**
Contact form	Supabase insert and Resend email are both best-effort; a failure in either does not necessarily block the user-facing success response	**app/api/contact/route.ts**
GitHub stats	Hardcoded `VERIFIED_FALLBACK` snapshot used on API error or rate limiting; language distribution is **always** a hardcoded `MOCK_LANGUAGES` constant, never computed from the live repo	**app/api/github/activity/route.ts**
ML Lab frontend	Per-service health check on page load surfaces outages before submission attempts, rather than failing silently on first request	**app/[locale]/ml/page.tsx**

> **Concrete incident on record.** A captured log line shows a real `scikit-learn` pickle-compatibility `ValueError` when loading a saved RandomForest — evidence that the "models trained on one machine may fail to load on another due to library version drift" risk (explicitly called out in `monitoring/drift_detector.py`'s own docstring) is not theoretical for this codebase.

15. Performance Analysis

- **Decorative rendering cost:** `AnimatedBackground.tsx` and `CursorEffect.tsx` run continuously on every page via `PlatformShell`, adding persistent client-side rendering work (canvas/DOM tracking) site-wide rather than only on marketing pages.
- **Cold starts:** three of five backends are Hugging Face Spaces (free-tier CPU), which sleep after inactivity; a user's first request after idle time should be expected to pay a multi-second wake-up latency. This is architectural (inferred from the deployment choice, evidence: `.env.example` HF Space URLs and each service's Dockerfile) rather than directly measured.
- **Sequential + parallel mixing in Financial Analyst:** four Gradio calls fire concurrently, but `analyze_stock` and `get_xai_explanation` run strictly after them — total latency for one "Run Analysis" click is `max(4 parallel calls) + analyze_stock + get_xai_explanation`, compounded by the fact that the XAI call's result is currently discarded (Section 5.4), meaning that latency is paid with zero user-visible benefit today.
- **ml-backend startup:** loads six models synchronously in the lifespan handler before accepting traffic; a large plant-disease checkpoint (EfficientNet-B0 weights) plus five classical models means a non-trivial boot time on Render's smaller instance tiers, though exact figures weren't captured in logs.
- **No caching layer observed** between the Next.js app and any backend for repeated identical requests (e.g., re-analyzing the same ticker) — each "Run Analysis" appears to always hit the live Gradio backend fresh.

16. Security Analysis

Area	Observation	Evidence
Contact form	Server-side field validation, honeypot field, and in-memory per-IP rate limiting are all present	**app/api/contact/route.ts**
Contact form rate limit durability	The rate limiter is in-memory (a JS `Map` in the route handler), which resets on every serverless cold start / redeploy and is not shared across concurrent instances — a determined actor could bypass it by triggering redeploys or hitting a fresh instance, though this is a common and often acceptable trade-off for a low-traffic contact form	**app/api/contact/route.ts**
Supabase key exposure	The anon key committed to `.env.example` is the public/client-safe key by design; the service-role key is referenced only via environment variable, never committed. RLS policy correctness on the Supabase project itself is outside this repository and unverifiable	**.env.example**, **lib/supabase.ts**
GitHub API calls	`/api/github/activity` calls GitHub unauthenticated (no token), which is lower-risk (no credential to leak) but subject to GitHub's stricter unauthenticated rate limits, mitigated by the fallback snapshot	**app/api/github/activity/route.ts**
File upload validation	hf_plant_api and skin-backend validate MIME type/size before PIL decode, reducing (not eliminating) risk from malformed or oversized uploads	**hf_plant_api/app.py**
CORS	ml-backend's `ALLOWED_ORIGINS` is environment-configurable, defaulting to `*` in local dev per `utils/config.py` — production origin restriction depends on the deployed environment variable, not verifiable from source alone	**ml-backend/utils/config.py**
career-backend secrets surface	Config requires Supabase service-role key, Groq, RapidAPI, GitHub, HuggingFace, and Gemini API keys — a broad credential footprint for a backend with no live traffic, worth pruning or removing if the service stays unshipped	**career-backend/utils/config.py**

17. Deployment and Operations

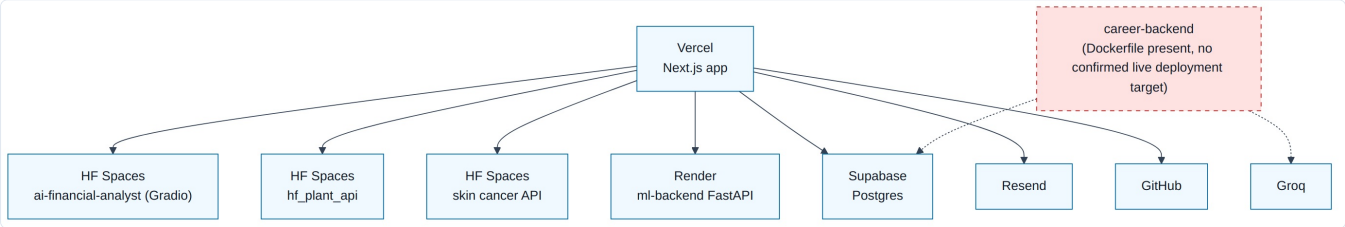

Fig. 11 — Deployment topology (repeated from Section 2 for reference in this section).

Service	Platform (inferred)	Evidence
Next.js app	Vercel	*Standard Next.js deployment convention; no explicit vercel.json found, relies on framework defaults*
ml-backend	Render	**.env.example**: `NEXT_PUBLIC_ML_API_URL=https://portfolio-pkdj.onrender.com`
hf_plant_api	Hugging Face Spaces (Docker)	**hf_plant_api/Dockerfile**, *HF Space URL in client code*
skin-backend	Hugging Face Spaces (Docker)	**skin-backend/Dockerfile**, `NEXT_PUBLIC_SKIN_API_URL`
ai-financial-analyst (Gradio)	Hugging Face Spaces	`anishreddy13-ai-financial-analyst.hf.space` *in* **financialAnalystClient.ts**
career-backend	Has a Dockerfile; no URL, env reference, or deployment evidence of it actually running anywhere reachable by this audit	**career-backend/Dockerfile** *exists; no confirmed live target*
ml-backend worker trio (streaming/prediction/drift)	Structured for HF Spaces (port 7860 health server convention) but not confirmed deployed there — only proven to have run on a local dev machine per logs	**ml-backend/start_workers.py**, **logs/*.log**

Two GitHub Actions workflows are present. `.github/workflows/frontend.yml` runs on push/PR touching `app/, lib/, src/, public/`, or config files: installs Node 20, runs `npm run lint`, then `npm run build` — a build/lint gate, with actual deployment left to Vercel's native git integration. `.github/workflows/ml_pipeline.yml` runs on push to `ml-backend/**`: installs a pinned scikit-learn/numpy/joblib/nltk set, downloads NLTK corpora, runs `pytest tests/ -v` (smoke tests in `ml-backend/tests/test_models.py` that load the sentiment and spam pipelines and assert they produce a prediction), and — regardless of test outcome, via `continue-on-error` — logs the run's status, commit, and message to a Supabase `pipeline_runs` table before finally failing the job if tests failed. This is a real, if lightweight, CI setup: it validates that both classical NLP models still load and predict on every backend change, and keeps an audit trail of pipeline runs in the same database the contact form uses.

18. Visual Feature Map

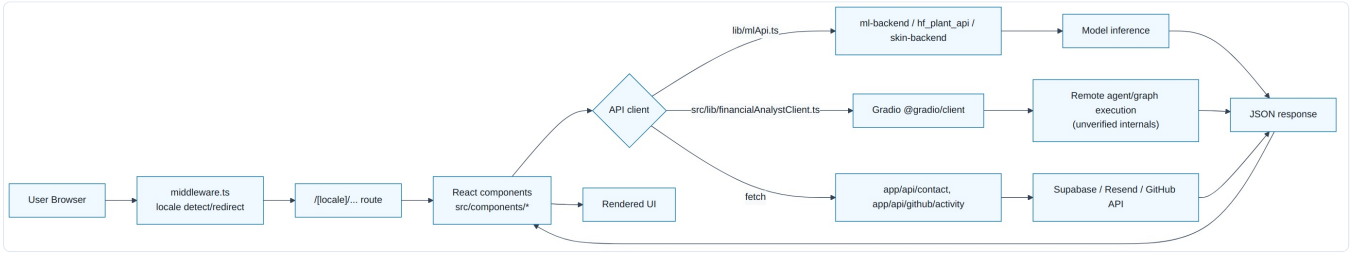

Fig. 12 — End-to-end request/dataflow: browser through middleware to every backend family.

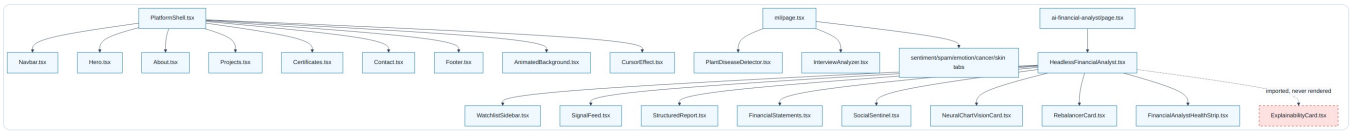

Fig. 13 — Component dependency graph for the platform shell, ML Lab, and AI Financial Analyst surfaces. Red dashed = imported but never rendered (Section 5.4).

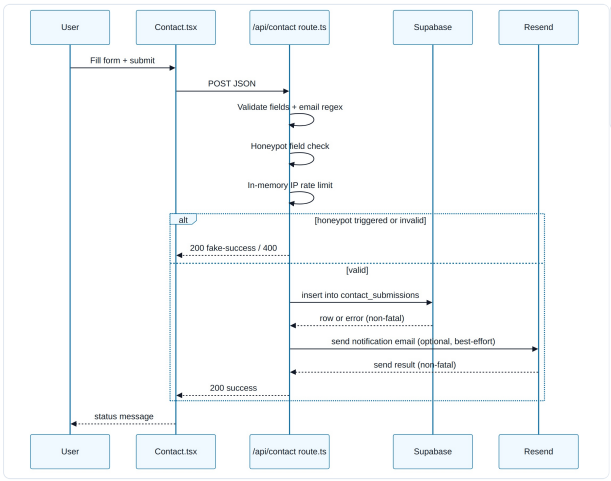

Fig. 14 — Contact form sequence.

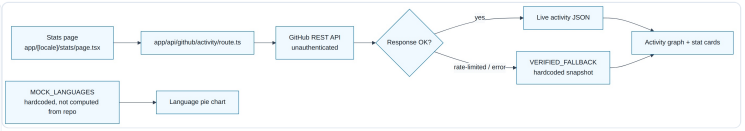

Fig. 15 — GitHub stats flow, including the two distinct fallback mechanisms.

User-facing surfaces, end to end: **Home** (identity, project teasers, contact) → **Projects grid** (links into ML Lab tabs and the Financial Analyst) → **ML Lab** (7 tabs, 6 live models + 1 browser-native) → **AI Financial Analyst** (ticker analysis, watchlist, rebalancer, explainer deep-dive) → **Stats** (GitHub activity) → **Contact** → **Print/PDF** (browser-native, via the hidden `PrintPortfolio` stylesheet, not a generated file).

19. Engineering Strengths and Limitations

19.1 Strengths

- **Genuine architectural range:** classical ML (TF-IDF/LogisticRegression), ensemble methods (VotingClassifier), deep learning (EfficientNet-B0, ResNet18 CNNs), and an orchestration layer calling a third-party multi-agent LLM system, all deployed to real, independently reachable services — not a single monolith pretending to be several projects.
- **Real MLOps instincts, not just inference endpoints:** per-model graceful degradation in `ml-backend`, a working RSS-ingestion → prediction → drift-detection pipeline with genuine runtime logs, and two functioning GitHub Actions pipelines (lint/build gate; ML smoke tests with a Supabase audit trail) are the kind of infrastructure many portfolio projects skip entirely.
- **Independently verifiable accuracy claim:** the plant-disease model's 97.84% figure, quoted in three separate places, matches the committed `test_metrics.json` exactly — the number is not just asserted copy.
- **Thoughtful fallback typing:** the `NeuralChartVisionPayload`'s explicit `deep_learning` vs. `heuristic_fallback` mode field is good API design — the frontend can react honestly to degraded backend state instead of silently pretending everything is a neural prediction.
- **Self-awareness already present in the codebase:** the hidden `PrintPortfolio.tsx` already flags the agent-count inconsistency this audit also found independently — evidence of the author (or a prior audit pass) already tracking technical debt rather than only marketing copy.

19.2 Limitations

- **Unresolvable core:** the single most architecturally interesting subsystem — the LangGraph multi-agent financial analyst — cannot be verified at all from this repository. Its submodule pointers are broken (no `.gitmodules`), which is itself a repository hygiene issue independent of what the backend actually contains.
- **career-backend represents substantial unshipped work:** a full second product (job-market intelligence, salary prediction, resume matching, LLM mentor) with no path from any page a visitor can reach. Either ship it behind a route, or remove it and its credential footprint from the deployed surface area.
- **Simulated progress UI:** the Financial Analyst's "agents working" animation not reflecting real backend state is a materially misleading trust signal for a product whose selling point is transparency/explainability (XAI).
- **Discarded XAI data:** paying for a `get_xai_explanation` call and then not rendering it is both a wasted-latency and a wasted-value bug — the fix is likely a single missing JSX line.
- **Inconsistent self-description:** the 3/4/5-agent discrepancy, plus the "EfficientNet-B0, Custom CNNs" blanket claim that doesn't distinguish the ResNet18 skin model, suggest documentation/marketing copy drifts out of sync with implementation changes over time.
- **No accessibility or automated testing artifacts** beyond the two GitHub Actions workflows and the one ML smoke-test file — no frontend unit/integration tests, no e2e tests, no a11y audit tooling were found.

19.3 Recommended next refactors

1. Restore `.gitmodules` (or fold the three submodules into the main repo) so the platform's most complex subsystem is auditable and CI-testable at all.
2. Render `ExplainabilityCard` where `xaiData` is already available, or remove the now-pointless `get_xai_explanation` call.
3. Pick one agent count and role list and propagate it to `financialAnalystContent.ts`, the live-loading animation, `FinancialAnalystExplainer.tsx`, and `projects.ts` — ideally driven by a single shared constant rather than four independent copies.
4. Either wire `career-backend` into a real route/page or remove it (and its five external API credentials) from the deployed footprint.
5. Retire or repoint `ml-backend`'s dead `/predict/plant` route, since the only actually-used, explainable implementation lives in `hf_plant_api`.
6. Replace the simulated agent-progress timers with real progress signals (SSE/polling against `get_service_health` or a dedicated status endpoint) if genuine transparency is the product's value proposition.

20. Appendix

20.1 Technology distribution

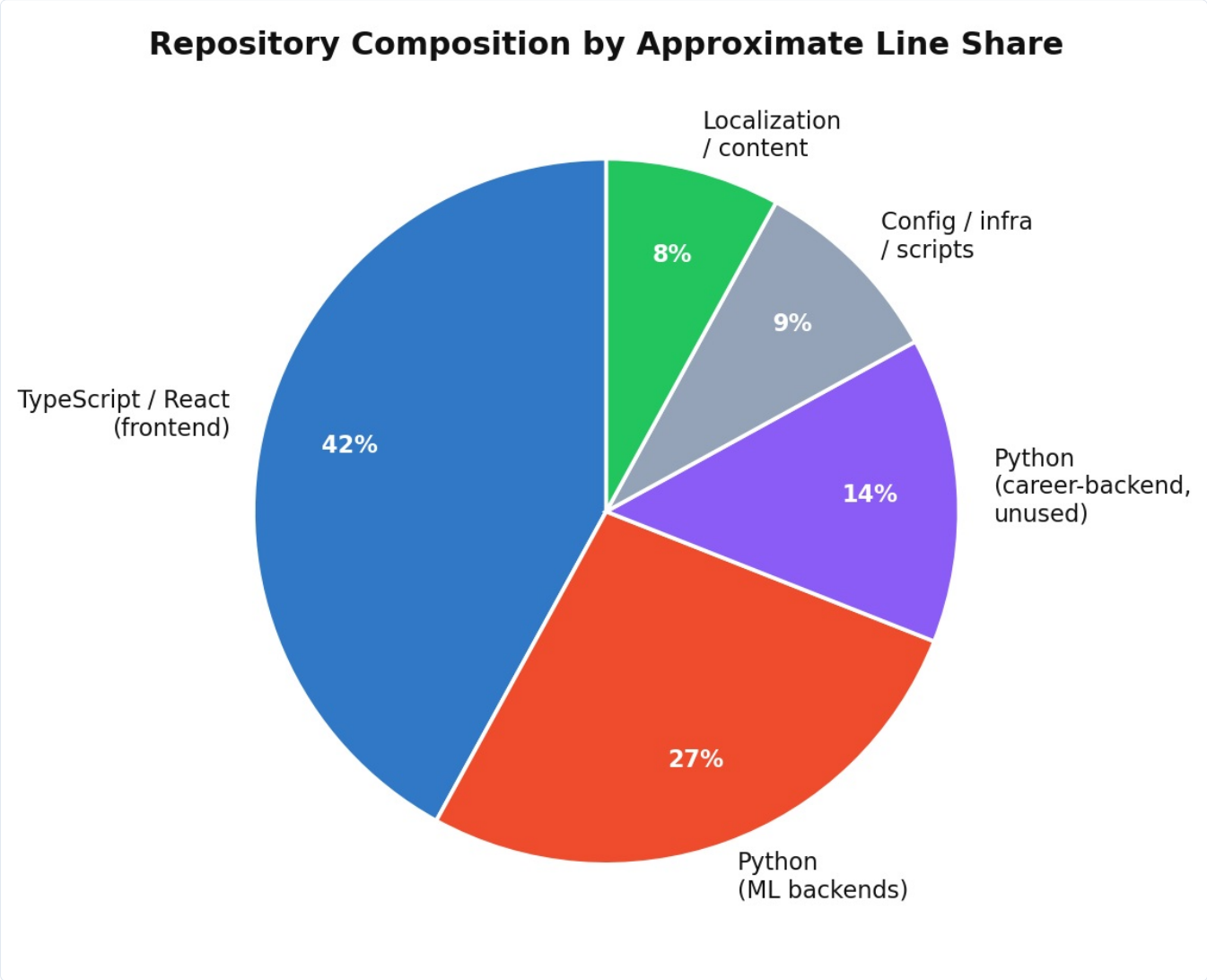

Fig. 16 — Approximate line-share by area. "Career-backend (unused)" is broken out separately from other Python to make the orphaned-code footprint visible on its own.

20.2 Feature complexity comparison

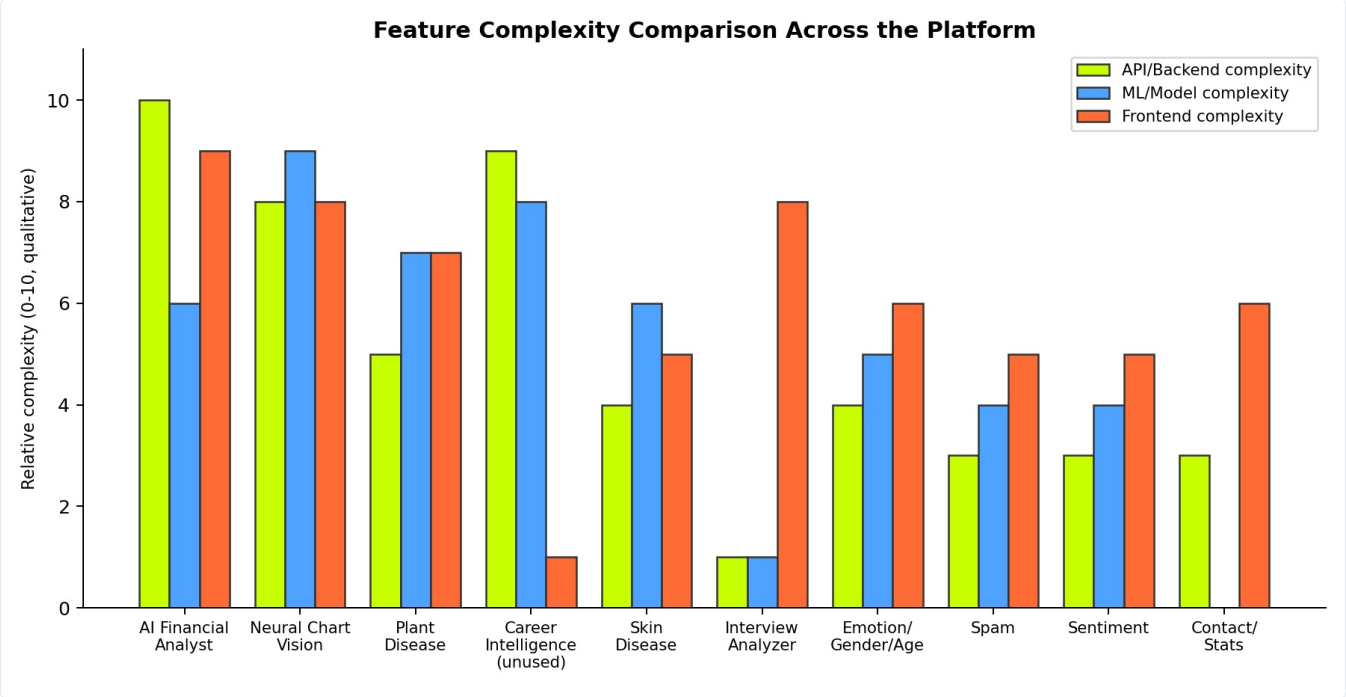

Fig. 17 — Qualitative 0–10 complexity scoring per feature across API/backend, ML/model, and frontend axes, based on this audit's reading of each feature's code (not a formal metric).

20.3 ML model inventory

Model	Type	Framework	Input	Output	File path	Deployment	Status
Sentiment	TF-IDF + LogisticRegression	scikit-learn	Text	3-class label + confidence	`ml-backend/model.py`	Render	Live
Spam	TF-IDF + LogisticRegression	scikit-learn	Text	2-class label + confidence	`ml-backend/spam_model.py`	Render	Live
Emotion/gender/age	Multi-head classifier	scikit-learn	Text	28-class emotion + gender + age	`ml-backend/emotion_model.py`	Render	Live
Breast cancer	VotingClassifier (LR+RF+GB)	scikit-learn	30 tabular features	Malignant/benign + probability	`ml-backend/cancer_model.py`	Render	Live
Plant disease (ml-backend copy)	EfficientNet-B0, transfer-learned	PyTorch/torchvision	Image (224×224)	15-class + confidence	`ml-backend/plant_model.py`	Render	ORPHANED
Plant disease (live)	EfficientNet-B0, transfer-learned, 97.84% test acc.	PyTorch/torchvision	Image (224×224)	15-class top-5 + Grad-CAM	`hf_plant_api/plant_model.py`	HF Spaces	Live
Skin lesion	ResNet18, fine-tuned	PyTorch/torchvision	Image	7-class top-5	`skin-backend/skin_model.py`	HF Spaces	Live
Salary predictor	XGBRegressor	XGBoost	Experience/location/skills features	Predicted salary	`career-backend/models/salary_predictor.py`	Not deployed	ORPHANED
Skill trend classifier	RandomForestClassifier	scikit-learn	Skill category features	Emerging/stable/declining	`career-backend/models/skill_trend_model.py`	Not deployed	ORPHANED
Resume-market matcher	TF-IDF + cosine similarity	scikit-learn	Skill lists	Match score, gap list	`career-backend/models/resume_matcher.py`	Not deployed	ORPHANED
Neural Chart Vision	CNN + Transformer fusion (claimed)	PyTorch (claimed)	OHLCV + candlestick image (claimed)	Pattern class + return prediction	`unresolved — submodule`	HF Spaces (Gradio)	UNRESOLVED
Financial multi-agent graph	LangGraph agent pipeline (claimed, 3/4/5 agents depending on source)	Unknown (claimed LangGraph)	Ticker + context	Structured markdown report	`unresolved — submodule`	HF Spaces (Gradio)	UNRESOLVED

20.4 Route inventory

`/`, `/ml`, `/projects/ai-financial-analyst`, `/projects/ai-financial-analyst/explainer`, `/stats`, `/privacy`, `/terms` — each locale-prefixed by `next-intl` middleware.

20.5 Public assets

`public/intro-audio.mp3` (played by `WelcomeAvatar.tsx` on first visit), `public/favicon.svg`, plus profile imagery and project screenshots referenced across the marketing components. Binary contents were not inspected beyond filename and referencing component.

20.6 Dependency inventory (top-level)

Frontend (package.json): next, react, react-dom, next-intl, @gradio/client, @supabase/supabase-js, framer-motion, three + @react-three/fiber + drei, recharts, react-markdown + remark-gfm, pdfjs-dist, lenis, lucide-react, clsx, uuid.

ml-backend: fastapi, uvicorn, scikit-learn, pandas, numpy, pillow, nltk, supabase, redis[hiredis], feedparser, apscheduler, httpx, mlflow, evidently, sse-starlette.

hf_plant_api / skin-backend: fastapi, uvicorn, pillow, torch (CPU wheel), torchvision (CPU wheel).

career-backend: fastapi, pydantic-settings, supabase, httpx, groq, scikit-learn, xgboost, mlflow, wandb, PyGithub.

20.7 Glossary

`Gitlink / submodule`	A git pointer to a commit in another repository; without a registered `.gitmodules` URL, cloning the parent repo does not fetch its contents.
`Grad-CAM`	Gradient-weighted Class Activation Mapping — produces a heatmap over an input image showing which regions most influenced a CNN's prediction.
`XAI`	Explainable AI — techniques and outputs intended to make a model's prediction interpretable to a human.
`Drift detection`	Monitoring a deployed model's prediction distribution over time to catch degradation from changing real-world input patterns.
`Orphaned code`	Working, deployable code with no reachable caller in the currently live product surface, as used throughout this document.

Generated by a code-level audit of anishreddy13/portfolio @ 7d08701. Markdown source, standalone diagram files, and this PDF are provided as companion deliverables. See the final message for the recommended upload path.