ChatGPT

ClutchFactor MVP Prototype Research Report

Executive summary

ClutchFactor is an NFL live win-probability experience that pairs **continuous win-probability updates** with **play-level explanations** (via SHAP) so fans can answer two questions in real time: “What are the chances my team wins?” and “What just changed and why?”

The core MVP thesis is that *live probability without interpretability feels like a black box*, and *interpretability without a stable probability signal feels like trivia*. The MVP therefore focuses on: (a) reliable ingestion of live play-by-play, (b) a calibrated win-probability model suitable for probability outputs, and (c) a lightweight explainer payload that can be served fast enough for real-time UI.

Key research outcomes and recommendations:

- **Data reality check:** Truly “official, real-time” NFL play-by-play and proprietary Next Gen Stats distribution is controlled through licensed commercial channels. Publicly accessible “official” material exists (e.g., NFL Game Books PDFs) but is not a convenient real-time API. ClutchFactor’s MVP should explicitly support two modes: (1) *Commercial mode* (licensed feed) and (2) *Developer mode* (open/unofficial data for prototyping). [1]
- **Primary official path:** Genius Sports [2] publicly states it remains the NFL’s exclusive distributor of real-time, official play-by-play statistics, proprietary Next Gen Stats data, and the league’s official sports betting data feed to media and betting operators. [3]
- **Best MVP architecture pattern:** Ingestion worker(s) → operational store (PostgreSQL) + hot cache (Redis) → prediction service → real-time client updates (SSE/WebSocket). This isolates vendor APIs server-side (a common requirement for B2B sports data providers) and provides stable latency under traffic spikes. [4]
- **Model strategy:** Start with a transparent baseline (logistic regression) and ship a stronger production model using gradient-boosted trees (e.g., XGBoost with probability objective), enabling fast “Tree SHAP” explanations. [5]
- **Explanation serving strategy:** Compute and store SHAP values for the *latest* play (or latest N plays) asynchronously; serve “top-K contributors” to the UI. This keeps the UI responsive while still delivering meaningful interpretability.

Assumptions (because not provided): target users are fans watching live games on web/mobile; MVP runs for a single week of games; training period uses multiple prior seasons of play-by-play; budget is startup-scale; no paid user accounts in MVP.

Product goals and MVP definition

Product goals (MVP): - Provide a **live win probability** after every meaningful play (or event) with a stable update cadence. - Provide **explainable deltas**: why the probability moved on the most recent event (and

optionally the last few). - Provide a UI that is “TV companion fast”: the probability and explanation should appear *quickly enough* to feel live (explicit latency targets defined below).

Primary users and user stories: - **Casual fan (mobile companion):**
Wants a single, simple number (home/away win%) and a short explanation (“Turnover + field position + time remaining”).
- **Analytics-minded fan:**
Wants a win-probability chart through time and a ranked breakdown of the drivers on each key play (top positive/negative features).
- **Developer/tester (MVP builder):**
Wants deterministic replays for integration testing. A “replay mode” is particularly valuable because live games are seasonal and hard to test end-to-end without historical time-stamped outputs. [6]

Success metrics (MVP): - Reliability: sustained live session without missed updates for a selected game. - Latency: p95 end-to-end under target (see “Realtime architecture”). - Explainability: >90% of plays have an explanation payload (top-K features + baseline value). - Model quality: calibrated probabilities (measured via Brier score/log loss + calibration curves). [7]

Prioritized MVP feature list: The MVP should intentionally avoid “everything football.” The priority is a narrow, delightful loop: *select game → watch win% update → understand why*.

Must-have: - Live games list (today/week) with status. - Game detail view: score, clock, possession, win probability. - Win-probability time series chart (updates as plays arrive). - “Why it changed” panel: SHAP top contributors for the latest play. - Replay mode (developer-focused) for deterministic tests in the off-season. [6]

Should-have: - Key play markers (turnovers, 4th-down conversions, TDs). - Odds-aware prior (point spread / moneyline) if available (improves calibration and realism); nflfastR’s win-probability calculator explicitly expects a spread line in its required inputs, underscoring the common practice of incorporating pregame strength info. [8]

Nice-to-have (post-MVP): - Multi-game “RedZone mode.” - Player-level context (QB, injuries) and richer features like proprietary tracking (Next Gen Stats) if licensed. The NFL states Next Gen Stats captures player tracking data (location/speed/acceleration) at 10Hz and produces many derived datapoints per play through ML on AWS. [9]

Data sources and licensing considerations

Prioritized sources

ClutchFactor’s data plan should be explicit about **(a) legality/terms**, **(b) real-time access**, and **(c) development ergonomics**.

Tier A: Official/licensed real-time feeds (production path) - Official, real-time distribution is described publicly as being handled through Genius Sports [2] for play-by-play, Next Gen Stats, and official betting data feeds (for media and betting operators). [3]

- Practical implication: an independent MVP usually **won't** obtain this feed without a commercial relationship/sub-license; therefore, design should abstract "provider adapters" so the app works with alternative feeds during prototyping.

Tier B: Commercial sports-data APIs (fastest path to real-time MVP if you can pay) - Sportradar [10] offers an NFL play-by-play endpoint that provides a live timeline including possession/location and play-earned stats, and also documents push-based options (server-side) for continuous updates. [11]
- SportsDataIO [12] documents an NFL API accessed via API key headers and provides workflow guidance including odds/line movement; it also offers "Replay" to play back real, time-stamped archived data for testing. [13]

Tier C: Official public artifacts (useful but not a "live API") - NFL Game Books (game summary PDFs) are accessible via NFL game centers ("Download Game Book PDF"). This is an official artifact for stats/play-by-play review, but it is not optimized for sub-minute live ingestion. [14]

Tier D: Unofficial/open tools (developer mode; prototyping and offline training) - nflfastR is an R ecosystem that scrapes/loads NFL play-by-play data (not an official API) and explicitly describes itself as accessing play-by-play data from nfl.com; it also provides win probability utilities and field descriptions. [15]
- ESPN [16] endpoints are often used unofficially by developers, but stability and terms vary; treat as a fallback and do not couple core production logic to it. [17]

Practical comparison table for MVP selection

Sources informing this comparison: vendor docs (play-by-play/push/auth patterns), NFL support article for Game Books, and odds API docs. [18]

Option	Real-time	Play-by-play	Odds	Dev ergonomics	Key constraints
Official licensed distribution (via Genius Sports)	Yes	Yes (official)	Yes (official)	Low for small teams	Requires commercial relationship; expensive/ negotiated
Sportradar	Yes	Yes	Yes (separate odds products)	Good docs; push options	B2B; server-side integration recommended; licensing/cost
SportsDataIO	Yes	Yes	Yes (aggregated odds/line movement)	Strong "Replay" testing	API key access; product tiers/add-ons
NFL Game Books PDFs	Not truly	Post-game artifact	No	OK for validation	Not a live feed; parsing PDFs is brittle

Option	Real-time	Play-by-play	Odds	Dev ergonomics	Key constraints
nflfastR / nflverse	Not live	Historical	Limited	Excellent for training	Unofficial scraping; not guaranteed for live

Odds sources for “spread-aware” win probability

If you want a spread-aware prior (highly recommended for model realism/calibration), likely options include: - SportsDataIO betting/odds structures (including line movement/props guidance). [19]
- The Odds API [20] for bookmaker odds (moneyline/spreads/totals) with documented sport keys and markets. [21]
- Sportradar odds products (separate from NFL play-by-play feeds). [22]

Backend and realtime architecture

System architecture (MVP)

```
flowchart LR
  subgraph External_Data[External Data Providers]
    PBP[Play-by-play API]
    ODS[Odds API]
  end

  subgraph Backend[ClutchFactor Backend]
    ING[Ingestion Worker]
    FE[Feature Builder]
    ML[Prediction Service]
    API[REST API]
    RT[Realtime Gateway\n(SSE/WebSocket)]
    CACHE[(Redis Cache)]
    DB[(PostgreSQL)]
  end

  subgraph Frontend[Web Client]
    UI[SPA UI\n(Games list + Game detail)]
  end

  PBP --> ING
  ODS --> ING
  ING --> DB
  ING --> CACHE
  DB --> FE --> ML --> DB
  CACHE --> API
  DB --> API
```

```
API --> RT --> UI
UI --> API
```

Why this structure is MVP-appropriate: - It keeps vendor keys and B2B contracts **server-side** (explicitly recommended in at least some sports data provider guidance). [23]
- It allows "hot reads" of the latest game state and win probability from Redis while persisting the authoritative event log in Postgres. Redis TTL primitives (EXPIRE/TTL) make it straightforward to auto-expire transient state. [24]

Realtime ingestion: polling vs streaming

For a live win-probability app, the ingestion design is the backbone.

If your provider supports streaming/push:
- Sportradar documents push feeds as a way to open a single connection and receive continuous live updates, reducing pull calls; it also notes push use requires appropriate real-time access. [25]
- Practical MVP approach: run a single ingestion service that connects to push and writes events to Postgres/Redis.

If your provider is polling-only (or you choose polling for MVP simplicity): - Poll a "Game Play-by-Play" endpoint at a frequency that respects provider TTL/caching and rate limits. Sportradar's live-game retrieval guidance references a short TTL once a game is in progress and provides recommended retrieval timing patterns. [26]
- Practical MVP polling cadence (proposal):
- Pregame: every 30–60 seconds (status scheduled)
- In-game: every 2–5 seconds, with backoff when idle
- Final: every 1–5 minutes for late stat corrections (or consult provider change logs if available)

Latency targets (explicit MVP SLOs)

Define SLOs so you can test and tune:

- **Ingestion freshness:** p95 "provider event time → stored event" < 3 seconds (polling) or < 1 second (push), excluding provider-side delays. (Proposal informed by vendor TTL guidance.) [26]
- **Prediction update:** p95 "event stored → win prob computed and available" < 1 second for baseline model, < 2 seconds with SHAP on latest play (using async compute if needed). (Proposal.)
- **Client update:** p95 "prediction available → UI updated" < 1 second (SSE/WebSocket). (Proposal.)

REST API design

The API should be versioned and documented with OpenAPI 3.1. The OpenAPI Specification provides the standard structure for describing paths and schemas, and FastAPI is a common Python framework that auto-generates OpenAPI schemas and interactive docs for endpoints. [27]

Base URL: `/api/v1`

Core endpoint list (MVP):

Method	Route	Purpose	Auth
GET	`/games`	List games by date/week/status	Optional (public)
GET	`/games/{game_id}`	Game details + latest state + latest WP	Optional (public)
GET	`/games/{game_id}/plays`	Incremental play feed	Optional (public)
POST	`/predict`	Predict WP for a provided game state	Required (token)
GET	`/models/current`	Current model metadata	Optional
GET	`/health`	Health/readiness	None
GET	`/stream/games/{game_id}`	Server-Sent Events stream of updates	Optional/Token

Auth design (MVP-ready, expandable): - Use a short-lived anonymous session token (JWT) in an HttpOnly cookie or Authorization header. - JWT is a standardized, compact representation of claims and is designed for use in HTTP contexts. [28]
- If/when you add accounts: add OAuth login, refresh tokens, and per-user quotas.

Rate limits (proposal): - Public GET endpoints: 60 requests/min/IP, burst 120. - Streaming endpoint: max 4 concurrent streams per token. - `/predict`: 30 requests/min/token (because SHAP can be compute-expensive).

These limits are also aligned with OWASP's emphasis on "lack of resources & rate limiting" as a recurring API security risk category. [29]

Sample JSON for key endpoints

1) GET `/api/v1/games?date=2026-02-08` (example response)

```
{
  "date": "2026-02-08",
  "games": [
    {
      "game_id": "2026020801",
      "season": 2025,
      "week": "POST-SB",
      "start_time_utc": "2026-02-08T23:30:00Z",
      "status": "final",
      "home_team": { "team_id": "NE", "name": "New England" },
      "away_team": { "team_id": "SEA", "name": "Seattle" },
      "score": { "home": 31, "away": 24 },
      "win_prob": { "home": 1.0, "away": 0.0, "as_of_play_seq": 182 }
    }
  ],
```

```
  "meta": { "source": "provider_adapter", "cache": "hit" }
}
```

2) GET `/api/v1/games/{game_id}` **(example response)**

```
{
  "game_id": "2026020801",
  "status": "inprogress",
  "season": 2025,
  "week": "POST-SB",
  "start_time_utc": "2026-02-08T23:30:00Z",
  "teams": {
    "home": { "team_id": "NE", "name": "New England" },
    "away": { "team_id": "SEA", "name": "Seattle" }
  },
  "score": { "home": 17, "away": 14 },
  "game_state": {
    "quarter": 3,
    "clock": "09:41",
    "possession_team_id": "NE",
    "down": 2,
    "ydstogo": 6,
    "yardline_100": 38,
    "timeouts": { "home": 3, "away": 2 },
    "score_differential_home": 3,
    "game_seconds_remaining": 1181,
    "half_seconds_remaining": 581
  },
  "latest_play": {
    "play_seq": 121,
    "type": "pass",
    "description": "Pass complete short left for 8 yards",
    "event_time_utc": "2026-02-09T01:12:19Z"
  },
  "win_prob": {
    "home": 0.63,
    "away": 0.37,
    "as_of_play_seq": 121,
    "model_version": "wp_xgb_v1"
  },
  "explanation": {
    "base_value_home": 0.51,
    "top_positive": [
      { "feature": "score_differential_home", "value": 3, "shap": 0.08 },
      { "feature": "yardline_100", "value": 38, "shap": 0.05 }
    ],
```

```
    "top_negative": [
      { "feature": "game_seconds_remaining", "value": 1181, "shap": -0.03 }
    ],
    "computed_at_utc": "2026-02-09T01:12:20Z"
  }
}
```

3) POST `/api/v1/predict` **(example request/response)**

```
{
  "game_id": "2026020801",
  "model_version": "wp_xgb_v1",
  "game_state": {
    "quarter": 4,
    "game_seconds_remaining": 132,
    "half_seconds_remaining": 132,
    "possession_team_is_home": true,
    "down": 1,
    "ydstogo": 10,
    "yardline_100": 25,
    "score_differential_home": -4,
    "timeouts_home": 2,
    "timeouts_away": 1,
    "receive_2h_ko": 0,
    "spread_line_home": -2.5
  },
  "explain": true,
  "explain_top_k": 6
}
```

```
{
  "prediction": {
    "win_prob_home": 0.28,
    "win_prob_away": 0.72,
    "model_version": "wp_xgb_v1",
    "timestamp_utc": "2026-02-09T02:41:10Z"
  },
  "explanation": {
    "method": "tree_shap",
    "base_value_home": 0.54,
    "top_contributors": [
      { "feature": "score_differential_home", "value": -4, "shap": -0.19 },
      { "feature": "game_seconds_remaining", "value": 132, "shap": -0.11 },
      { "feature": "yardline_100", "value": 25, "shap": 0.04 },
      { "feature": "timeouts_home", "value": 2, "shap": 0.03 },
```

```
      { "feature": "down", "value": 1, "shap": 0.02 },
      { "feature": "spread_line_home", "value": -2.5, "shap": 0.01 }
    ]
  }
}
```

Data model and PostgreSQL schema

Postgres is a strong fit for an MVP because it supports robust relational integrity for games/plays and also supports `jsonb` and GIN indexing for semi-structured vendor payloads if you store "raw provider events" alongside normalized columns. [30]
Use B-tree indexes for common ordered lookups like `(game_id, play_seq)` and for equality/range filters. [31]

Schema goals: - Preserve a canonical event log (auditable, replayable). - Support fast reads of latest state and WP series. - Support SHAP explainability lookups keyed by prediction/play.

Proposed MVP schema (tables, keys, indexes):

Table	Purpose	Primary key	Key foreign keys	Suggeste
`teams`	Team reference	`team_id`	—	unique or
`players`	Player reference (optional MVP)	`player_id`	`team_id → teams`	(`team_i` optional
`games`	Game metadata	`game_id`	`home_team_id → teams`, `away_team_id → teams`	(`start_` (`season`
`plays`	Normalized play-by-play rows	(`game_id` , `play_seq`)	`game_id → games`	btree (`ga` (`event_`
`play_raw`	Raw provider payloads	`raw_id`	`game_id → games`	GIN on `p` (optional) (`game_i`
`game_state_snapshots`	Derived "state after play" features	(`game_id` , `play_seq`)	`game_id → games`	btree (`ga`
`model_versions`	Model registry	`model_version`	—	(`traine`

Table	Purpose	Primary key	Key foreign keys	Suggeste
`wp_predictions`	Win probability outputs	`prediction_id`	(`game_id`, `play_seq`) → snapshots; `model_version` → `model_versions`	(`game_i` (`model_`
`shap_values`	Per-feature SHAP attribution	(`prediction_id`, `feature_name`)	`prediction_id` → `wp_predictions`	(`predic`
`odds_snapshots`	Odds per game/time (optional MVP)	`odds_id`	`game_id` → `games`	(`game_i`

Design note: keeping `plays` normalized while storing `play_raw.payload_jsonb` preserves vendor compatibility and supports debugging and reprocessing. Postgres JSONB + GIN indexing is appropriate if you need to query across raw payload keys; otherwise, keep it simple and index only normalized columns. [30]

ML pipeline and SHAP explainability

Problem framing

Each play updates the game state. The model outputs **P(home team wins | current state)** (or equivalently away win probability as 1 − home).

Labeling (supervised learning): - For each play snapshot: label = 1 if home team wins game, else 0.

Training dataset assumptions

Not specified by the user, so the MVP can proceed with explicit assumptions: - **Assumed training period:** multiple recent seasons of play-by-play (e.g., 2016–2024), with the most recent completed season held out for final evaluation (e.g., 2025). (Assumption.) - **Assumed sample size:** “hundreds of thousands” of play snapshots after filtering non-informative events. (Assumption.)

If you use nflfastR for training data, note it is explicitly described as accessing NFL play-by-play from nfl.com and includes a rich set of fields and modeling utilities. [32]

Feature engineering (MVP feature set)

A strong MVP uses features that are: 1) available live, 2) stable across seasons/providers, and 3) interpretable.

A pragmatic MVP feature set closely overlaps with what nflfastR's win-probability calculator expects as minimum required inputs (down/distance/field position/time/score/spread/second-half receive indicator). [8]

Game context - `score_differential_home` (home − away) - `home_team`, `possession_team_id`, `possession_team_is_home` - `receive_2h_ko` (if known/derivable from game events)

Clock and phase - `quarter` - `game_seconds_remaining` - `half_seconds_remaining`

Down and distance / field position - `down` - `ydstogo` - `yardline_100` (distance to opponent end zone)

Timeouts (if available) - `timeouts_home`, `timeouts_away`

Pregame priors (optional but recommended) - `spread_line_home` (or moneyline-implied prior) - `total_points_line`

Odds integration is commonly documented by sports data providers and can be added as snapshots. [33]

Derived features (if time permits) - Possessions remaining proxy: function of timeouts + time remaining + quarter (heuristic). - Drive-level indicators: start field position, drive result flags.

Model choice and baselines

Baseline model (required for rigor): - Logistic regression on core features. (Provides interpretability and sanity checking.)

Primary MVP model: gradient-boosted trees - Use a tree ensemble so you can compute fast, instance-level SHAP values via TreeExplainer. SHAP's TreeExplainer documentation describes Tree SHAP as a fast method for tree models. [34]
- If using XGBoost, use the binary classification probability objective (commonly `binary:logistic`). XGBoost parameter docs describe using `binary:logistic` for probability outputs in binary classification. [35]

Evaluation metrics and validation strategy

Because win probability is a **probability forecasting** task, evaluate both accuracy and calibration:

Primary metrics - **Brier score loss:** mean squared error between predicted probability and outcome. scikit-learn documents this directly. [36]
- **Log loss (cross-entropy):** penalizes overconfident wrong predictions; scikit-learn defines it as negative log-likelihood of the probabilistic model. [37]

Calibration - Reliability diagrams / calibration curves. scikit-learn documents calibration curves as true vs predicted probabilities. [38]

Cross-validation strategy - **Time-based (season-forward) CV:** Train on seasons up to year N, validate on year N+1; roll forward.
Rationale: prevents leakage from future seasons and respects temporal drift (rule changes, play style changes). (Methodological recommendation.)

Sanity checks - Monotonicity expectations (heuristic): All else equal, larger score deficit late in game should generally reduce win probability. - Symmetry checks: identical states mirrored for home/away should reflect complementary probabilities. (Recommendation.)

SHAP explainability integration (compute + serve)

SHAP is a game-theoretic approach for feature attribution; the library offers a unified API and TreeExplainer for tree models. [39]

Where SHAP runs - Run SHAP server-side only (never in-browser for MVP) to control CPU costs, prevent model extraction, and standardize payload size. (Security/performance recommendation.)

When SHAP runs - **Synchronous for on-demand** `/predict?explain=true` if latency budget allows (e.g., single instance call).
- **Asynchronous for live games:** enqueue SHAP compute when a new play arrives; persist top-K feature attributions for the "latest play" explanation panel.

What is served to clients - `base_value` (expected model output) - top-K positive and negative attributions: (`feature_name`, `feature_value`, `shap_value`) - optionally, a "humanized" template string generated by deterministic rules (not an LLM in MVP) to avoid adding a second AI subsystem.

Storage - Store SHAP attributions in `shap_values` keyed by `prediction_id` so you can fetch quickly for the UI and also re-render explanations historically.

ML data flow diagram

```
flowchart TD
  A[Raw play-by-play events] --> B[Normalize + derive game_state_snapshot]
  B --> C[(Training dataset)]
  C --> D[Train baseline LR]
  C --> E[Train GBDT model]
  E --> F[Calibration check\n(Brier/log loss + reliability)]
  E --> G[Register model_version]
  G --> H[Online inference\n(per new play)]
  H --> I[Compute SHAP\n(TreeExplainer)]
  H --> J[(wp_predictions)]
  I --> K[(shap_values)]
  J --> L[API serves latest WP series]
  K --> M[API serves explanation payload]
```

Frontend UX and client integration

Key screens (wireframe descriptions)

Games list ("Today / This Week") - Top bar: date selector; "In progress / Upcoming / Final." - Game cards show: teams, score, quarter/clock, and a compact win%:
- "HOME 63% • AWAY 37%"
- Clicking opens game detail.

Game detail - Header: team names/logos, score, status, possession indicator. - Main chart: win probability over time (line chart). - Event feed: latest plays at top with a small win% delta badge. - Explanation drawer (right side or below on mobile):
- Title: "Why it changed"
- Two short ranked lists: "Increased home win%" and "Decreased home win%"
- Each item: feature label + value + contribution magnitude.

Key play view (optional MVP) - Clicking a play highlights it on the chart and shows the SHAP attribution breakdown for that play.

Real-time update behavior

Two MVP-safe patterns:

1) **SSE (recommended MVP):** - Client subscribes to `GET /api/v1/stream/games/{game_id}` and receives events: - `play_added` - `wp_updated` - `explanation_ready` - SSE is simpler operationally than WebSockets and fits "server → client updates" well. (Design recommendation.)

2) **Polling (fallback MVP):** - Poll `GET /api/v1/games/{game_id}` every 2–5 seconds. - Poll `GET /api/v1/games/{game_id}/plays?since_play_seq=...` for incremental updates.

If using a provider with published TTL behavior, align polling to avoid waste and rate-limit pressure. [26]

REST calls from frontend (typical flow)

- List games: `GET /api/v1/games?date=YYYY-MM-DD`
- Open game: `GET /api/v1/games/{game_id}`
- Subscribe to live updates: `GET /api/v1/stream/games/{game_id}`
- If replay mode enabled (dev): `POST /api/v1/replay/start` (optional endpoint) and then same stream.

Delivery, deployment, security, and testing

Dev tasks and sprint plan (2–3 week MVP)

The sprint plan assumes one engineer can ship a functioning prototype in ~3 weeks (or a small team in ~2 weeks). (Assumption.)

Sprint backlog overview

Sprint	Deliverables	Key tasks
Sprint A	Data ingestion + game state	Provider adapter; normalize plays; store in Postgres; Redis hot cache; `/games` + `/games/{id}` endpoints
Sprint B	Model v1 + evaluation harness	Build training dataset; baseline LR + tree model; offline evaluation (Brier/log loss + calibration); model registry + `/predict`; SHAP compute service
Sprint C	Frontend + realtime + polish	Games list + game detail UI; SSE streaming; win% chart + explanation panel; basic monitoring; load test; deployment

Rigor checklist per sprint - Sprint A must prove: event ordering is correct (`play_seq` monotonic) and “latest snapshot” retrieval is <50ms from Redis under normal use (target). (Proposal.) - Sprint B must prove: evaluation scripts reproducible; baseline vs model improvement documented; SHAP payload computed and stored for latest play. - Sprint C must prove: end-to-end demo using replay mode or a past game recording.

Infrastructure and deployment

Proposed MVP stack - Backend: FastAPI (OpenAPI generation + typed schemas) [40]
- DB: PostgreSQL (normalized tables + optional JSONB) [41]
- Cache: Redis (TTL-based hot cache) [24]
- Background jobs: worker queue (Celery/RQ) for ingestion and SHAP compute (design recommendation). - Deployment: containerized app to a managed platform (e.g., AWS/GCP/Fly/Render). If you later integrate Next Gen Stats-like workloads, note the NFL describes building derived stats through ML on Amazon Web Services [42] . [9]

CI/CD (MVP) - Lint + unit tests + integration tests on PR. - Build/push container image; run migrations; deploy.

Monitoring (MVP) - Metrics: request latency, SSE connection counts, ingestion lag, prediction compute time, SHAP compute time. - Logs: structured JSON logs with correlation IDs. - Alerts: ingestion stalled, error rate spikes.

Cost considerations (MVP) - Major cost drivers: paid data feed (if commercial), compute for SHAP, and bandwidth for live streaming. - Recommendation: aggressively cache and serve incremental updates; most clients don’t need full play payload every update.

Security and privacy considerations

ClutchFactor is primarily sports-data + analytics, so privacy risk is limited unless you add accounts. Still, security should be deliberate.

API security priorities - Apply OWASP API Security Top 10 guidance as a checklist: broken object-level authorization, broken authentication, and lack of rate limiting are common API risk categories. [29]
- Use JWT for session tokens (short-lived), per RFC 7519; store in HttpOnly cookies when possible to reduce

token theft via XSS. [28]
- Enforce per-route rate limits and request body size limits (especially `/predict`).

Vendor key protection - Never expose third-party API keys in the browser. Some provider documentation explicitly notes B2B APIs are not intended to be called directly from a client application. [23]

Data integrity - Validate play ordering; de-duplicate events; keep raw payloads for audit; avoid silently mutating historical events unless you can prove corrections (some providers offer change logs). [26]

Testing plan

A credible MVP needs tests across ingestion, ML, API, and UI.

Backend - Unit tests: feature builder (down/distance/time conversions), state snapshot correctness. - Contract tests: endpoint schemas vs OpenAPI definitions (OpenAPI 3.1). [43]
- Integration tests: ingestion → DB write → prediction → API read. - Load tests: simulate N concurrent SSE connections; validate Redis hit rates and Postgres read amplification.

ML - Reproducibility tests: fixed data slice → deterministic metrics. - Metric tests: Brier score and log loss computed correctly per scikit-learn definitions. [44]
- Calibration checks: reliability diagram generation produces expected binning and diagnostics. [38]
- SHAP tests: TreeExplainer produces attributions; top-K extraction stable; payload size bounded. [45]

Frontend - Component tests: chart renders incremental points; explanation panel updates on `explanation_ready`. - E2E tests: replay mode runs through a full game and produces stable UI snapshots.

Manual QA (game-day checklist) - Start a live in-progress game; verify ingestion lag; verify win% monotonic updates; verify “latest play” SHAP appears within target latency.

---

[1] [3] The National Football League Expands and Extends Strategic ...
https://investors.geniussports.com/news/news-details/2025/The-National-Football-League-Expands-and-Extends-Strategic-Partnership-with-Genius-Sports-in-Multi-Year-Deal/default.aspx?utm_source=chatgpt.com

[2] [7] [10] [36] [44] brier_score_loss
https://scikit-learn.org/stable/modules/generated/sklearn.metrics.brier_score_loss.html?utm_source=chatgpt.com

[4] [23] Sportradar API Documentation
https://developer.sportradar.com/getting-started/docs/get-started?utm_source=chatgpt.com

[5] [35] XGBoost Parameters — xgboost 3.1.1 documentation
https://xgboost.readthedocs.io/en/stable/parameter.html?utm_source=chatgpt.com

[6] [16] [20] API Resources | Introduction & Testing
https://sportsdata.io/developers/apis?utm_source=chatgpt.com

[8] Compute win probability — calculate_win_probability
https://nflfastr.com/reference/calculate_win_probability.html?utm_source=chatgpt.com

9 NFL Next Gen Stats
https://operations.nfl.com/gameday/technology/nfl-next-gen-stats/?utm_source=chatgpt.com

11 18 Game Play-by-Play
https://developer.sportradar.com/football/v5/reference/nfl-play-by-play?utm_source=chatgpt.com

12 38 calibration_curve
https://scikit-learn.org/stable/modules/generated/sklearn.calibration.calibration_curve.html?utm_source=chatgpt.com

13 NFL Data API Developer Portal
https://sportsdata.io/developers/api-documentation/nfl?utm_source=chatgpt.com

14 Game Books
https://support.nfl.com/hc/en-us/articles/35869678028180-Game-Books?utm_source=chatgpt.com

15 An R package to quickly obtain clean and tidy NFL play by ...
https://nflfastr.com/?utm_source=chatgpt.com

17 Does the NFL have an API or service for people to access ...
https://www.reddit.com/r/NFLNoobs/comments/16zwdtm/does_the_nfl_have_an_api_or_service_for_people_to/?utm_source=chatgpt.com

19 33 Betting Data Integration Guide
https://support.sportsdata.io/hc/en-us/articles/4404845466519-Betting-Data-Integration-Guide?utm_source=chatgpt.com

21 NFL Odds API
https://the-odds-api.com/sports-odds-data/nfl-odds.html?utm_source=chatgpt.com

22 Odds API Overview
https://developer.sportradar.com/odds/reference/intro?utm_source=chatgpt.com

24 EXPIRE | Docs
https://redis.io/docs/latest/commands/expire/?utm_source=chatgpt.com

25 Push Feeds
https://developer.sportradar.com/football/docs/nfl-ig-push?utm_source=chatgpt.com

26 Live Game Updates
https://developer.sportradar.com/football/docs/nfl-ig-live-game-retrieval?utm_source=chatgpt.com

27 43 OpenAPI Specification v3.1.0
https://spec.openapis.org/oas/v3.1.0.html?utm_source=chatgpt.com

28 42 RFC 7519 - JSON Web Token (JWT)
https://datatracker.ietf.org/doc/html/rfc7519?utm_source=chatgpt.com

29 OWASP Top 10 API Security Risks – 2023
https://owasp.org/API-Security/editions/2023/en/0x11-t10/?utm_source=chatgpt.com

30 41 Documentation: 18: 8.14. JSON Types
https://www.postgresql.org/docs/current/datatype-json.html?utm_source=chatgpt.com

31 Documentation: 18: 65.1. B-Tree Indexes
https://www.postgresql.org/docs/current/btree.html?utm_source=chatgpt.com

32 Help for package nflfastR
https://cran.r-project.org/web/packages/nflfastR/refman/nflfastR.html?utm_source=chatgpt.com

34 45 shap.TreeExplainer — SHAP latest documentation
https://shap.readthedocs.io/en/latest/generated/shap.TreeExplainer.html?utm_source=chatgpt.com

37 log_loss — scikit-learn 1.8.0 documentation
https://scikit-learn.org/stable/modules/generated/sklearn.metrics.log_loss.html?utm_source=chatgpt.com

39 shap/shap: A game theoretic approach to explain the ...
https://github.com/shap/shap?utm_source=chatgpt.com

40 Features
https://fastapi.tiangolo.com/features/?utm_source=chatgpt.com