GameLens.ai — Predictive Analytics Requirements (NFL, NBA, NCAAF, NCAAB)

0) Goals & Non-Negotiables

Primary goal: Produce reliable, explainable per-game probabilities (Win, ATS, O/U) and score distributions,
then expose them via API and a UI "GameLens Analysis" card.
Leagues: NFL, NCAAF, NBA, NCAAB (MLB optional/next).
Design tenets: Data-first architecture (single feature vector per game), reproducible simulations, cached results, clear narratives.
Outputs (per matchup): Win%, Cover%, Over%, median & bands for scores/total, and sensitivity to line movement.

Phase 1 — Data Lake, Feature Vectors, and Monte Carlo MVP

Phase 1 establishes the foundation of GameLens' predictive analytics system — the data lake, unified feature vectors,
and the Monte Carlo simulation engine to generate per-game outcome probabilities.

- Data Architecture & Storage: Postgres (Supabase) with schemas for leagues, teams, players, games, odds, injuries, weather, rolling_stats, feature_vectors, simulation_jobs, simulation_results_summary, narratives_cache.
- ETL & Data Freshness: Historical + daily/hourly refresh for schedule/scores/odds/injuries/weather via SportsDataIO with idempotent UPSERTS and job logs.
- Rolling Team Metrics: Compute and store season-to-date and last-N (5/10) offensive/defensive efficiency, pace, rest days, home-edge, and recent form views.
- Unified Feature Vector: Create a JSON per game aggregating all relevant data, stored in feature_vectors for downstream models.
- Monte Carlo Engine (v1): Simulate 10,000+ game iterations using expected points and standard deviations adjusted for team efficiency, injuries, weather, and home advantage.
- API Endpoints: /api/games, /api/odds/latest, /api/simulate, providing per-game predictions and summaries.
- Exit Criteria: Data lake live, Monte Carlo model returning stable results, and APIs with narrative generation deployed.

Phase 2 — Add Classical Models (Stacked with Monte Carlo)

Build supporting statistical models that complement and feed into the Monte Carlo engine and produce independent probability estimates.

- Elo Rating Model: Team strength updated each game; produces win probabilities and rating deltas.
- Poisson/Skellam Model: Uses offensive/defensive rates and pace to model score distributions and total points.
- Logistic Regression Model: Uses derived features (Elo diff, efficiency diff, rest, home, etc.) to classify Win/Loss/ATS outcomes.
- Pythagorean Expectations Model: Estimates expected win % from points scored/allowed as a regression baseline.
- Bayesian Hierarchical Model: Applies partial pooling to balance team priors and league trends, generating posterior win probabilities with uncertainty.
- Meta-Ensemble Model: Stacks Monte Carlo, Elo, Poisson, Logistic, and Bayesian outputs into a single GameLens Composite Probability (GLP).

Exit Criteria: Models deployed and accessible via /api/predictions with calibration metrics visualized in the dashboard.

Phase 3 — Situational & Live Models

Integrate situational models and in-game live updates to enhance prediction precision and responsiveness.

- Markov Chain Model: Models possession-level transitions for basketball tempo and shot efficiency.
- Drive-Level EPA Model: Calculates Expected Points Added per drive for football to refine offensive efficiency.
- Dynamic Bayesian Updating: Incorporates play-by-play in-game data to update live win probabilities.
- Environmental Models: Weather and fatigue adjustments applied pre-simulation.

Phase 4 — Machine Learning & AI Enhancements

Leverage modern ML algorithms to improve model accuracy and discover complex feature relationships.

- Gradient Boosted Trees (XGBoost/LightGBM): Train ensemble classifiers for Win/ATS/O/U outcomes.

- LSTM / Temporal CNN: Learn sequential performance trends and momentum over time.
- Graph Neural Networks (GNN): Model player and team interactions to predict synergy and outcomes.
- Reinforcement Learning: Optimize edge discovery and staking strategy policies.
- Meta-Learner: Combine classical and ML model outputs into a unified ensemble with rolling retraining and calibration.

Deliverables by Phase

- Phase 1: Schema, ETL, feature vectors, Monte Carlo, APIs, narratives, runbooks.
- Phase 2: Elo, Poisson, Logistic, Pythagorean, Bayesian, Stacking, Calibration Dashboard.
- Phase 3: Markov, EPA, live updating, environmental models.
- Phase 4: ML/AI models, meta-learner, retraining automation.

Monte Carlo Model — Detailed Development Scope

This section provides the full development requirements and implementation guidance for the Monte Carlo simulation model
within GameLens.ai. The purpose of this model is to estimate the probability distribution of possible outcomes for sporting
events based on team-level and contextual input variables, producing win probabilities, score distributions, and over/under metrics.

1. Objectives

The Monte Carlo simulation engine should:
• Generate probabilistic outcome predictions for each matchup across supported leagues (NFL, NBA, NCAAF, NCAAB).
• Utilize team statistics, efficiency metrics, injuries, weather, rest, and betting line data to model realistic outcomes.
• Output multiple prediction metrics including Win %, Cover %, Over %, expected scores, and sensitivity to line movement.
• Support integration into ensemble models (Elo, Poisson, Logistic, Bayesian) for improved predictive accuracy.

2. Data Inputs

The following data fields are required to build accurate simulation distributions. Each must be present in the unified feature vector
for each game in the data lake:

- Team offensive and defensive efficiency (adjusted by opponent strength).

- Pace or possessions per game (tempo adjustment).
- Home-field or home-court advantage factor.
- Injury adjustment (key player impact on expected points).
- Weather impact (for outdoor sports affecting scoring variance).
- Rest days and travel distance modifiers.
- Betting market inputs (closing spread, total points, moneyline).
- Historical variance of team scoring outcomes (standard deviation inputs).

3. Model Design

The Monte Carlo model will simulate 10,000+ iterations per game using probabilistic sampling based on derived parameters
for each team. Core design specifications include:

- Use a bivariate normal distribution to jointly sample home and away scores.
- Mean and standard deviation (μ, σ) of each team's score distribution derived from efficiency metrics and recent form.
- Correlation (ρ) parameter set between 0.2–0.35 to account for interdependence in scoring outcomes.
- Adjust μ and σ based on injury reports, weather factors, and rest-day modifiers.
- Clamp negative simulated scores at zero; round fractional values to nearest integer if desired.
- Use vectorized computation (NumPy) to ensure high performance for thousands of simulations per request.
- Store all simulation results in `simulation_results_summary` with unique game_id, version, and timestamp.

4. Outputs

Each Monte Carlo run should return the following summary statistics, both stored in the database and available via the API:

- home_win_pct, away_win_pct
- home_cover_pct, away_cover_pct (against spread)
- over_pct, under_pct (against total)
- mean and median projected scores for home and away
- p10, p50, p90 percentile scores (distribution range)
- spread_sensitivity and total_sensitivity metrics showing how small line movements affect probabilities
- model_version, simulation_run_id, and simulation_timestamp for traceability

5. Technical Implementation

Recommended stack and development standards for implementation:

- Language: Python (preferred).
- Libraries: NumPy, Pandas, SciPy for distribution sampling; PostgreSQL (Supabase) for storage; FastAPI for API endpoints.
- Use async tasks or job queues for background computation of large simulation batches (e.g., Celery or Supabase Functions).
- Cache results to avoid recomputation for the same game unless odds or injury inputs change.
- Design model to run independently per league to allow modular updates and scaling.
- Enable version control for model parameters (e.g., number of iterations, correlation constants).
- Log all model runs including parameter set and random seed for reproducibility.

6. Validation & Testing

The model should undergo extensive validation and backtesting against historical data. Key validation components include:

- Backtest Monte Carlo predictions versus historical results to measure accuracy and calibration (Brier score, LogLoss).
- Compare simulated outcomes to market-implied probabilities (moneyline and spread).
- Run sensitivity tests by varying correlation and standard deviation to observe model stability.
- Ensure that re-running simulations with the same seed produces identical results (reproducibility test).
- Perform integration tests for data ingestion and API endpoint consistency.

7. Deliverables

Upon completion, the developer must deliver:

- Fully implemented Monte Carlo simulation module integrated into the GameLens backend.
- Database schema updates for simulation_results_summary and simulation_jobs tables.
- API endpoints for triggering simulations and retrieving results.
- Unit and regression test scripts covering simulation logic and data integrity.
- Documentation and runbook describing parameters, assumptions, and troubleshooting steps.

- Demo notebook or script showing example simulation run for one NFL and one NBA game.