Phase 2 v2 – Feature Engineering
(Merged & Expanded from Phase 2 v1)

---

1. Purpose of Phase 2
In Phase 1, you built the raw data layer: players, defenders, teams, games, offensive stats, team defense stats, defender stats, and (optionally) coverage events.
In Phase 2, you transform that raw data into features that a prediction model can actually understand and use.
Raw stats like:
"RB123 rushed for 72 yards in Week 5"
by themselves are not sufficient. The model and baseline logic need context, such as:

- How has this player been performing recently vs their own baseline?
- How strong or weak is the opposing defense against this position?
- How has this specific defense performed *recently* vs historical?
- Which defenders are likely to be involved in coverage or run fits?
- What archetype does this player fit (deep WR, slot WR, pass-catching RB, etc.)?
- What archetype does the defense or specific defenders fit (press-man corner, soft zone, etc.)?
- How much did the team throw/run in similar game scripts in the past?

Phase 2 is where we define and compute all those contextual variables and compress them into a single "giant table" of numerical inputs ("features") per player-game instance.
That table will be called:
player_game_features
Each row = one player in one game.
Each column = one engineered feature (plus label columns for training).

---

2. Conceptual View of Feature Engineering
You can think of Phase 2 in "spreadsheet" terms:

- Rows: (player_id, game_id) → "this player in that game"
- Columns: Features that capture:
    - player's role, usage, efficiency
    - recent performance vs long-term baseline
    - opponent team-defense strength
    - opponent *defender-specific* behavior
    - archetypes (offense + defense)
    - matchup context (home/away, total, spread, etc., if available later)

For example, for a running back in Week 8:

- season_rush_yards_per_game_up_to_week7
- last3_rush_yards_exp (exponentially weighted average of last 3 games)
- def_season_rush_yards_allowed_to_rb_up_to_week7
- def_last3_rush_yards_allowed_to_rb_exp
- rb_snap_share_last3
- rb_target_share_last3

- effective_defender_run_metric (weighted composite of run defenders)
- etc.

The goal is:
For each (player, game) pair, compute a *complete feature vector* containing everything the downstream models need to make a prediction.

---

3. How Phase 2 Connects to Other Phases
Phase 2 sits directly between:

- Phase 1 (Data Ingestion) – raw tables:
  - players, defenders, teams, games
  - player_game_stats
  - team_defense_game_stats
  - defender_game_stats
  - optional coverage_events

and

- Phase 3 (Baseline Predictor) + Phase 4 (ML Training):
  - Both take player_game_features as input.
  - Baseline uses more hand-crafted weighted formulas.
  - ML models learn data-driven mappings from features → stats.

Phase 2 must therefore:

- Provide all features used by the baseline model.
- Provide rich, leak-safe features for ML models.
- Provide specialized features for:
  - Archetype modeling
  - Defender-aware modeling
  - Ensemble meta-learning

If you under-build Phase 2, the later phases are crippled.
If you over-build, you can always select subsets, but you must never leak future information.

---

4. Core Feature Families
Phase 2 v2 retains all the conceptual feature groups from v1 and expands them substantially. At a high level, we will build:

1. Player Usage Features
   - How much the player is on the field and involved (snap share, route share, target share, carry share).
2. Player Efficiency Features
   - How effective the player is when used (yards per route, yards per carry, TD rate, etc.).
3. Player Form Features (Recent vs Long-Term)
   - Rolling, windowed, and exponentially weighted metrics describing current form vs baseline.
4. Team Context Features
   - Offensive tendencies (run/pass ratio, pace, red-zone usage, etc.).
5. Opponent Team-Defense Features + Deltas
   - How the opposing defense performs vs positions (WR, RB, TE, QB), both long-term and recently.
6. Defender-Level Features + Deltas (New in v2)

    - Per-defender performance, alignment usage, coverage stats, and their recent vs baseline deltas.
7. Coverage Probability Features (New in v2)
    - Estimated probability that a given WR/TE/RB is primarily handled by certain defenders or defensive shells.
8. Effective Defender Metrics (New in v2)
    - Weighted composite defender metrics based on coverage probabilities, used for matchup-sensitive prediction engines.
9. Archetype-Related Features
    - Features that animate archetype clustering and later allow the archetype-response model to work (e.g., depth usage, slot/boundary, target profile).
10. Meta-Features for Ensemble & Interpretability
    - Simple interpretable aggregates (like “how much did recent form influence expected value vs defender difficulty”) that will feed the ensemble and explanation layer.

All of these must be produced without any leakage: only data from weeks *before* the target week may be used.

---

5. Label Construction (What We Are Predicting)

Although Phase 4 will handle training in detail, Phase 2 must also prepare label columns in player_game_features to avoid duplication of logic later.

For each (player, game) row we want to be able to predict:

- For WR/TE:
    - targets
    - receptions
    - rec_yards
    - rec_tds
- For RB:
    - carries
    - rush_yards
    - rush_tds
    - targets, receptions, rec_yards, rec_tds (if used as receiver)
- For QB:
    - pass_attempts
    - completions
    - pass_yards
    - pass_tds
    - interceptions
    - rush_yards, rush_tds (for mobile QBs)

These are the label columns.

Phase 2 is not “training” yet, but must attach each feature row to the matching label values so training can simply say:

X = feature columns, y = label columns.

---

6. Anti-Leakage Rules (Critical)

The most important rule in Phase 2:

You must never use stats from Week N to predict Week N.

Everything is cut off at week-1.

For each target game (season = S, week = W):

- All rolling averages, exponential weights, deltas, archetypes, defender metrics, etc. must be computed using data from:
  - Weeks < W in the same season, and possibly
  - Previous seasons (up to the rolling window limit), but never using the target week itself.

Practically:

- If you're building features for Week 8:
  - You can use Weeks 1–7 of that season.
  - You can also use prior seasons if you choose (with decaying weight).

This rule applies to:

- Player stats
- Team-defense stats
- Defender stats
- Archetype clustering inputs
- Coverage probability calibration
- Any derived performance metrics

You should explicitly build helper functions that take:

```
def build_features_for_week(season: int, week: int, max_history_seasons: int = 3):
    # 1. Filter all raw stats to those with (season < season) OR (season == season and week < target_week)
    # 2. Compute all rolling averages, deltas, archetypes, and defender metrics using only that history.
    # 3. Join context features with the (season, week) schedule and player list to produce feature rows for that upcoming week.
```

---

7. Player Usage Features

Player usage is about *how much opportunity* the player has.

Typical usage metrics:

- Snap share:
  - snap_share_season = player_snaps / team_offensive_snaps (up to previous week)
  - snap_share_last3_exp = exponential weighted average over last 3 games
- Route share (for WR/TE/RB running routes):
  - route_share_season
  - route_share_last3_exp
- Target share:
  - target_share_season = targets / team_pass_attempts
  - target_share_last3_exp
- Carry share (for RBs):
  - carry_share_season = carries / team_rush_attempts
  - carry_share_last3_exp
- Red-zone usage share:
  - rz_target_share_lastN
  - rz_carry_share_lastN

Implementation idea:

1. For each (player_id, game_id) in player_game_stats, we also need the team-level totals for that game (e.g., total snaps, total passes, total rushes).
2. Aggregate per-player and per-team across all *past* games.
3. Compute ratios per player.

These usage features were already present conceptually in v1; v2 emphasizes:

- Using both season-long and short-window (last N / last3_exp) metrics.
- Ensuring the features are available for all players with enough historical usage.

---

8. Player Efficiency Features

Usage tells us how often the player is involved.
Efficiency tells us how effective they are when they get touches.
Typical efficiency metrics:

- WR/TE:
  - yards_per_route_run
  - yards_per_target
  - yac_per_reception
  - tds_per_target
- RB:
  - yards_per_carry
  - yards_after_contact_per_carry (if available)
  - tds_per_carry
- QB:
  - yards_per_attempt
  - tds_per_attempt
  - interceptions_per_attempt

We compute:

- eff_season_... versions (over all prior games in the season).
- eff_last3_exp... versions (weighted to recent games).

Example:

```
wr_yprr_last3_exp =
    weighted_sum(yards / routes in last 3 games)
```

These were in v1 conceptually (efficiency vs volume); v2 keeps them and expands explicit definitions and variants.

---

9. Player Form Features (Recent vs Long-Term)

“Form” in v1 was described as how a player is doing relative to their norm.
In v2, we formalize this as deltas between short-window metrics and long-window (or season) metrics:

- delta_rush_yards_last3_vs_season
- delta_rec_yards_last3_vs_season
- delta_targets_last3_vs_season
- delta_snap_share_last3_vs_season
- etc.

Conceptually:

```
delta_stat = recent_stat (last N or exp-weighted) - season_baseline
```

We can do this for:

- Volume stats (targets, carries, snaps, routes)
- Efficiency stats (yards per route, yards per carry, etc.)

These deltas become behavioral features:
is the player trending up or down vs their norm?
This was baked into v1 as “form,” but v2 explicitly defines it as delta_* features and ensures:

- Use of both season baseline and multi-season baseline (if needed).
- Use of exponentials where appropriate (last3_exp is recommended for recency).

---

10. Team Context Features
Team context features capture *how the offense behaves as a whole*.
Examples:

- team_pass_rate_season
- team_pass_rate_last3_exp
- team_rush_rate_season
- team_plays_per_game_season (pace)
- team_rz_pass_rate_season (how often they throw in red zone)
- team_target_concentration_index (are targets concentrated among 1–2 players or spread out?)

These features help the model anticipate:

- Whether volume is likely to be constrained by a run-heavy or pass-heavy scheme.
- Whether there is room for a WR3 to produce, for example.

This concept existed in v1 as “team context” — v2 simply requires you to formalize specific metrics and attach them as columns.

---

11. Opponent Team-Defense Features + Deltas
Now we move into the strength of the opposing defense, which v1 already had but in simpler form.
For each opposing defense, we want to know:
Against WR:

- def_wr_yards_per_game_season
- def_wr_targets_per_game_season
- def_wr_yards_per_target_season
- def_wr_tds_per_game_season
- def_wr_yac_allowed_per_reception_season

Against RB:

- same concepts: yards per game, per carry, TDs allowed, receiving yards allowed, etc.

Against TE:

- same structure.

We also want recent defensive form vs their own baseline:
delta_def_wr_yards_allowed_last3_vs_season
delta_def_rb_yards_allowed_last3_vs_season
...
And we can add advanced metrics from team_defense_game_stats:

- delta_def_pass_epa_allowed_last3_vs_season
- delta_def_success_rate_allowed_last3_vs_season

- delta_def_explosive_pass_rate_last3_vs_season

The idea is:
If a defense has suddenly started getting roasted by WRs or RBs, the model should “know” that beyond season averages.

---

12. Defender-Level Features + Deltas (NEW in v2)
This is one of the major v2 changes.
Using defender_game_stats, we compute:
Per defender:

- def_cov_snaps_season (coverage snaps)
- def_targets_allowed_per_game_season
- def_yards_allowed_per_target_season
- def_yac_allowed_per_reception_season
- def_tds_allowed_per_target_season
- Alignment usage:
  - alignment_boundary_pct
  - alignment_slot_pct
  - alignment_deep_pct
  - alignment_box_pct
- Coverage style:
  - man_coverage_pct
  - zone_coverage_pct

And form deltas:

- delta_def_yards_allowed_last3_vs_season
- delta_def_tds_allowed_last3_vs_season
- delta_def_targets_allowed_last3_vs_season

These metrics will feed into:

- defender archetypes (press-man lockdown vs soft zone etc.)
- effective defender metrics (below)
- defender-aware model (Phase 4)

---

13. Coverage Probability Features (NEW in v2)
We want features that describe:
“How likely is it that this offensive player will be primarily influenced by a specific defender or set of defenders?”
For a WR, we might define:

- p_covered_by_primary_cb
- p_covered_by_slot_cb
- p_covered_by_safety_help
- p_covered_by_zone_shell

If we have precise coverage event data (coverage_events), we can empirically estimate these probabilities over past games. If not, we will approximate based on:

- Offensive player alignment (slot vs boundary)
- Defender alignment usage
- Team’s man/zone tendencies
- Target depth

These probabilities form the weights used in the next step.

---

14. Effective Defender Metrics (NEW in v2)

Once we have:

- per-defender metrics (Section 12)
- coverage probabilities (Section 13)

we can define an effective defender metric:

```
effective_defender_yards_allowed_metric_for_off_player =
    Σ_over_defenders [ p_player_covered_by_defender_i *
def_i_yards_allowed_per_target ]
```

Similarly for TDs allowed, YAC allowed, etc.

We can build:

- effective_def_yards_allowed_per_target
- effective_def_tds_allowed_per_target
- effective_def_yac_allowed_per_reception
- effective_def_delta_yards_allowed_last3_vs_season

These features give the model and baseline engine a single numeric handle on:

“How scary is the likely matchup, after considering coverage shares and defender form?”

This is a core new building block for the defender-aware submodels and the ensemble.

---

15. Archetype-Related Features

Archetype modeling (separate doc) will cluster players/defenders into archetypes based on:

- usage profile (slot vs wide, depth of target, etc.)
- physical attributes
- role and alignment
- efficiency patterns

Phase 2 must supply the raw features used for clustering:

For offensive players (especially WR/TE/RB):

- slot_rate
- wide_alignment_rate
- route_depth_distribution
- target_depth_profile
- yards_after_catch_profile
- usage_by_down_distance (if available)
- designed_run_usage (for WRs with end-arounds, etc.)

For defenders:

- alignment_boundary_pct, alignment_slot_pct, etc.
- man_coverage_pct, zone_coverage_pct
- effectiveness vs archetype types (once computed iteratively)

Archetype ids themselves may be computed in a separate process, but Phase 2 is where the *inputs* are engineered and stored.

---

16. Schema of player_game_features

At the end of Phase 2, you will have a table, for example:

Table: player_game_features

Columns (not exhaustive, but representative):

- IDs:
  - player_id
  - game_id

    - team_id
    - opponent_team_id
    - season
    - week
    - position
    - off_archetype_id (once computed)
    - def_team_archetype_id (optional)
- Usage:
    - snap_share_season, snap_share_last3_exp, ...
    - route_share_*, target_share_*, carry_share_*
- Efficiency:
    - yprr_season, yprr_last3_exp
    - yards_per_carry_*, etc.
- Form deltas:
    - delta_targets_last3_vs_season, delta_yards_last3_vs_season, ...
- Team context:
    - team_pass_rate_season, team_pass_rate_last3_exp, ...
- Opponent team-defense:
    - def_wr_yards_allowed_per_game_season, delta_def_wr_yards_allowed_last3_vs_season, etc.
- Defender-related:
    - effective_def_yards_allowed_per_target
    - effective_def_tds_allowed_per_target
    - effective_def_yac_allowed
    - optional primary_cb_archetype_id, etc.
- Archetype inputs / metadata:
    - slot_rate, deep_route_rate, etc.
- Labels:
    - targets
    - receptions
    - rec_yards
    - rec_tds
    - carries
    - rush_yards
    - rush_tds
    - etc. (depending on position)

---

17. Implementation Approach (SQL + Python)

Phase 2 can be implemented as a set of Python scripts that:

1. Query raw data from SQLite (Phase 1 DB).
2. Use pandas to aggregate, compute rolling windows, exponential averages, and deltas.
3. Join tables into a unified feature dataframe.
4. Write the result back to SQLite as player_game_features.

Suggested script breakdown:

- build_player_usage_features.py
- build_player_efficiency_features.py
- build_player_form_features.py
- build_team_context_features.py

- build_team_defense_features.py
- build_defender_features.py
- build_effective_defender_features.py
- assemble_player_game_features.py

You can combine later, but keeping them separate while building is easier to debug.

---

18. Quality Checks for Phase 2

Before you consider Phase 2 “complete” for a given data_version:

- Randomly sample rows and verify:
  - Feature values make intuitive sense.
  - No obvious leaks (e.g., using Week 10 stats for Week 10 features).
- Check:
  - No feature columns are 100% null for players who should have history.
  - Ranges make sense (snap_share between 0 and 1, etc.).
- Build quick distributions / histograms for critical features:
  - extreme values should be rare but plausible.
- Confirm:
  - For a player who was on a bye last week, recent-form features still only use prior weeks.
  - For rookies, features gracefully degrade (less history → more reliance on limited data, but not NaNs).

---

19. Summary of Phase 2 v2

Phase 2 v1 already defined:

- Why feature engineering matters.
- How “one row per player-game, many columns per feature” works.
- High-level concepts for usage, efficiency, opponent defense, and player form.

Phase 2 v2 retains all of that and adds:

- Defender-level features and deltas
- Coverage probability features
- Effective defender metrics
- Archetype input features
- Clearer definitions and naming for all main feature families
- Strict anti-leakage rules formalized
- Clear connection to Phase 3 (baseline), Phase 4 (ML), and archetype/defender-aware submodels

At the end of Phase 2, you will have a rich, leak-safe player_game_features table that any junior engineer can hand to a modeling script and simply say:

“Use these feature columns as X, and these label columns as y.”

And every later phase—baseline, ML, ensemble, defender-aware, and explanation—will be standing on a solid, well-documented foundation.