

RadarPremios — Arquitectura lógica y plan de integración de señales

Objetivo: definir cómo RadarPremios genera candidatos **el mismo día del sorteo**, evalúa **al día siguiente** contra los resultados reales, propone **mejoras de señales** y vuelve a generar candidatos de forma continua, con trazabilidad, reproducibilidad y reportes automatizados.

1) Alcance y principios

- **Rigor estadístico:** probabilidad (Kolmogórov), procesos dependientes (Markov), simulación (Montecarlo).
 - **Reproducibilidad:** toda generación guarda **semilla, señales y pesos** usados.
 - **Ética & claridad:** no hay garantías de acierto. Los reportes separan **hechos** (datos) de **hipótesis** (señales).
 - **Resiliencia:** si un fetch falla (HTTP 500, etc.), se marca el sorteo como *failed* pero el pipeline sigue con el resto.
-

2) Calendario de sorteos y orquestación (hora local)

Se generan **candidatos** con antelación *el mismo día* y se evalúan **T+1**.

Frecuencias (proveídas): - **Super Astro Luna:** L-S 22:30; D/Fest 20:30 (4 dígitos + signo) - **Tolima:** L 23:00 - **Huila:** M 22:30 - **Manizales:** X 22:30 - **Quindío:** J 22:30 - **Medellín:** V 22:30 - **Boyacá:** V 23:00 - **Baloto/Revancha:** L, X, S 23:05 y 23:15 (ventana)

Estrategia de orquestación - Pre-sorteo (Generación): tarea por juego para el **mismo día** con margen razonable antes del sorteo (p.ej. 18:00 para vespertinos; 12:00 para diarios). - **Post-sorteo (Evaluación):** **①** *pull* de resultados en ventana del sorteo (ya configurado en tu Programador) y **②** evaluación a **T+1 07:00**. - **Reportes:** diario (T+1 07:10) y semanal (Dom 08:00).

Nota: ya tienes schtasks para la **recolección** en ventana de sorteo. Aquí añadimos sólo las tareas **de generación** (pre) y **evaluación/reportes** (post). Ver §11.

3) Flujo E2E por día

```
D 08:00  -> Actualiza señales base (ventanas móviles, hot/cold, Markov).
D 12:00+ -> Genera candidatos "del día" por juego según agenda (§2).
D 22:30 -> (Tu Programador) scrapea resultados según cada sorteo.
T+1 07:00 -> Evalúa candidatos vs resultados reales.
T+1 07:10 -> Reporte diario + propuesta de ajustes de señales.
```

4) Estructura de carpetas y variables

- %RP_ROOT% = C:\RadarPremios
 - %RP_SCRIPTS% = C:\RadarPremios\scripts
 - %RP_DB% = C:\RadarPremios\radar_premios.db
 - data\crudo → CSV raw de scrapers
 - data\limpio → CSV depurados
 - logs → bitácoras por etapa
 - reports → HTML diarios/semanales (incl. four_d_light.html, four_d_advanced.html)
 - backups → backups versionados de DB
-

5) Datos clave y DB

Tablas ya presentes (relevantes para esta lógica) - Resultados & premios: baloto_resultados, revancha_resultados, *_premios, astro_luna, boyaca, huila, manizales, quindio, medellin, tolima. - **Vistas de soporte:** v_digit_hotcold, v_digit_hotcold_win, v_last_seen_digit_pos(_win), v_last_seen_exact, v_4d_pos_expanded(_win), v_4d_markov_support, v_baloto_features_alineado. - **Ejecuciones y candidatos:** rp_runs, run_candidates, rp_predictions. - **Metadatos:** meta_kv.

Extensiones propuestas - rp_eval — resultados de evaluación por juego/fecha/sorteo (hit exacto, parciales, métricas). - signals_config — pesos por señal (nombre, valor, estado, fecha_vigencia). - signals_history — tracking de valores agregados de señales por fecha (auditoría). - rp_daily_report — punteros a archivos HTML/CSV por fecha (para navegación rápida).

No se reemplaza nada existente; se añade lo mínimo para cerrar el ciclo generar→evaluar→ajustar→reportar.

6) Señales (features) y fundamentos

Base matemática - Kolmogórov: tratamos los eventos como sucesos en un espacio de probabilidad; evitamos “overfitting” incorporando una penalización por **complejidad** (proxy de complejidad algorítmica): secuencias excesivamente simples o repetitivas bajan score. - **Markov (1er orden por posición):** matrices de transición por dígito y posición (y para signo en Astro Luna) con ventanas móviles (7, 14, 30, 90 días). Apoyado en v_4d_markov_support y v_4d_pos_expanded(_win). - **Montecarlo:** simulación para líneas base y para stress-test de coberturas: qué tasa de acierto esperar si sólo muestreamos al azar vs. con señales.

Señales concretas (por juego) - 4D (Boyacá, Huila, Manizales, Quindío, Medellín, Tolima) - Hot/Cold por posición: v_digit_hotcold(_win) con suavizado exponencial. - *Last-seen distance* por dígito/posición: v_last_seen_digit_pos(_win). - **Markov posicional** (transiciones de dígitos por posición t-1 → t). - **Estructura de patrón:** repetidos, consecutivos, sumatoria y módulo 9, paridad. - **Complejidad (proxy):** compresión LZ o RLE del cuatrígrafo; menor compresión → mayor score (penaliza patrones triviales). - **Super Astro Luna (4 dígitos + signo)** - Todas las anteriores + **Markov de signo** (12 signos) y co-ocurrencia dígito/signo. - **Baloto/Revancha (combinatorias)** - Frecuencias marginales y conjuntas

(pares/ternas), *last-seen*, Markov de apariciones (presencia/ausencia), equilibrio par/impar y bajo/alto, distancia en el círculo 1..43/1..16 (si aplica), y penalización por *pick humano típico* (p. ej., 1-31).

Normalización y mezcla - Cada señal se escala a [0,1] por juego y ventana. - Score final por candidato: - $\text{score} = \sum (\text{w}_i * \text{s}_i)$ con w_i de `signals_config`. - *Fusión ventanas*: media ponderada (peso mayor a ventanas cortas si buscamos momentum, o largas si buscamos estabilidad).

7) Generación de candidatos (pre-sorteo del día)

RNG y reproducibilidad - RNG por defecto: `SystemRandom` / `secrets` (CSPRNG). Se guarda **semilla derivada** en `rp_runs` (hash de hora del juego + jitter + hardware/OS entropy). - Se registra en `rp_runs`: `run_id`, `ts`, `juego`, `target_fecha`, `semilla`, `version_señales`.

Algoritmo general 1. Cargar dataset del juego (N últimos sorteos configurables). 2. Calcular señales ($\S 6$) y normalizar. 3. Generar un *pool* grande de combinaciones válidas vía muestreo ponderado por `score` con **softmax($\lambda \cdot \text{score}$)** para explorar/explotar. 4. Aplicar **filtros de cobertura**: - No duplicados de días recientes. - Cobertura de rangos (paridad, alto/bajo, etc.). - Límite de combinaciones por jugador: configurable. 5. Ordenar por `score` y seleccionar Top-K (K por juego). 6. Persistir en `run_candidates` (*pool*) y `rp_predictions` (Top-K) con `signals_snapshot` (JSON por candidato: valores por señal) + `semilla` + λ .

Tamaños sugeridos - 4D/Astro Luna: Top-20 a Top-50 por juego/día. - Baloto/Revancha: Top-30 a Top-100 (según presupuesto).

8) Evaluación (T+1) y ajuste de señales

Métricas - *Hit exacto*, *aciertos parciales* (4/3/2/1 dígitos en posición para 4D; conteo de coincidencias para combinatorias), *top-k recall* (si hubo hit dentro de Top-k). - *Lift vs azar* (Montecarlo baseline): $\text{lift} = \text{tasa_acierto_modelo} / \text{tasa_azar}$. - *Stability* de señales: correlación entre `score` y acierto por banda.

Cierre diario - Grabar en `rp_eval` por juego/sorteo. - Calcular **sugerencia de ajuste** de `signals_config`: - Subir peso a señales con $\text{lift} > 1$ y estabilidad > umbral. - Bajar o desactivar señales con aporte nulo/negativo. - *Rate limiting*: cambios $\leq \pm 10\%$ por día; consolidación semanal. - Guardar snapshot en `signals_history`.

9) Reportes

Reporte diario (HTML) - Candidatos de ayer por juego, resultados reales, métricas de acierto y *lift*. - Tabla de **señales** por juego (pesos actuales vs propuestos), con semáforos. - "Qué funcionó / Qué falló" (bullets automáticos) y *checklist* de acciones.

Reporte semanal (HTML) - Tendencias de acierto, estabilidad de señales, *drift*. - Resumen de cambios aplicados y plan de la semana siguiente.

Ubicación: `reports\YYYY\MM\informe_YYYYMMDD.html` y `reports\semanal_YYYYwwW.html`. Índices en `rp_daily_report`.

10) Manejo de errores y resiliencia

- **Fetch 500:** política 2 reintentos + **fallback URL** (ya implementado). Si persiste, `status=failed` y `heartbeat` en log/DB; se reintenta en el próximo ciclo.
 - **Datos faltantes:** generación no se bloquea; se usan ventanas disponibles. Las métricas de evaluación excluyen sorteos sin dato.
 - **Backups:** posterior a cada ciclo completo (`backups\radar_premios_YYYYMMDD_HHMMSS.db`).
 - **Versionado:** `meta_kv` guarda `model_version`, `signals_version` y `schema_version`.
-

11) Tareas programadas (añadidos)

Mantienes tus tareas de **recolección** en ventana de sorteo. Añadir:

Generación (mismo día) - Astro Luna: L-S 18:00 , D 12:00

```
schtasks /Create /TN "RP_GEN_AstroLuna_LS_1800" /TR "\"%MASTER% gen astro_luna\"" /SC WEEKLY /D MON,TUE,WED,THU,FRI,SAT /ST 18:00 /F schtasks /Create /TN "RP_GEN_AstroLuna_D_1200" /TR "\"%MASTER% gen astro_luna\"" /SC WEEKLY /D SUN /ST 12:00 /F - 4D (cada una el día correspondiente 18:00): Tolima L, Huila M, Manizales X, Quindío J, Medellín V, Boyacá V (18:00) . - Baloto/Revancha L,X,S 18:00.
```

Evaluación/Reportes (T+1) - Eval: schtasks /Create /TN "RP_EVAL_Diario_0700" /TR "\"%MASTER% eval all\"" /SC DAILY /ST 07:00 /F - Reporte: schtasks /Create /TN "RP REP_Diario_0710" /TR "\"%MASTER% report diario\"" /SC DAILY /ST 07:10 /F - Semanal: schtasks /Create /TN "RP REP_Semanal_Dom_0800" /TR "\"%MASTER% report semanal\"" /SC WEEKLY /D SUN /ST 08:00 /F

"%MASTER% gen|eval|report" son nuevas sub-acciones del **master.bat** (ver §12) que encadenan scripts Python.

12) Módulos/Scripts (nuevos) y contratos

1) `scripts/generate_candidates.py` - **Args:** `--juego <astro_luna|baloto|revancha|boyaca|...>` `--fecha YYYY-MM-DD` `--topk N` `--lambda <float>` - **Out:** Inserta en `run_candidates` (pool) y `rp_predictions` (Top-K). Guarda snapshot de señales y semilla en `rp_runs`.

2) `scripts/evaluate_predictions.py` - **Args:** `--fecha YYYY-MM-DD` `--juego all|<juego>` - **Out:** `rp_eval` con métricas; actualización de `rp_daily_report` (paths a HTML/CSV).

3) `scripts/update_signal_weights.py` - **Lógica:** lee `rp_eval` (últimos 7/14/30 días), estima *lift* por señal, aplica ajuste acotado ±10% y versiona en `signals_config` + `signals_history`.

4) `scripts/report_daily.py` y `scripts/report_weekly.py` - **Out:** HTML con tablas/plots (reutiliza estilo de `four_d_light.html`). Indexación anual/mensual.

5) `scripts/orchestrate_today.py` - **Función:** dado `--fecha` y el calendario (§2), dispara `generate_candidates.py` por cada juego con sorteo ese día (idempotente).

6) `master.bat` (**extensión**) - Nuevas sub-acciones: `gen`, `eval`, `report` que encadenan los scripts anteriores con logs y manejo de errores homogéneo.

13) Contratos de datos (propuestos)

`rp_runs` (extender si hace falta): `run_id (PK) | ts | juego | target_fecha | kind(gen/eval/report) | seed | status | notes | model_version | signals_version`

`run_candidates`: `run_id (FK) | juego | combo | score | rank | signals_snapshot(JSON) | created_ts`

`rp_predictions`: `run_id (FK) | juego | combo | score | rank | target_fecha | created_ts`

`rp_eval`: `eval_id | juego | sorteo_id | fecha_sorteo | actual | topk | hits_exactos | hits_parciales(JSON) | lift | resumen | created_ts`

`signals_config`: `signal | juego | weight | enabled | window | updated_ts | version`

`signals_history`: `ts | signal | juego | weight | metric | value`

14) Métricas clave de éxito (KPIs)

- **Disponibilidad:** % de sorteos del día con candidatos generados.
 - **Trazabilidad:** % de runs con semilla + snapshot de señales guardadas.
 - **Lift vs azar** por juego y horizonte (7/30 días).
 - **Tiempo** de generación y de reporte.
-

15) Roadmap de implementación

Fase 1 (rápida) - Orquestación: `orchestrate_today.py`, tareas GEN/EVAL/REPORT. - Señales: Hot/Cold, Last-seen, Reglas de patrón, Top-K y reportes diarios.

Fase 2 - Markov posicional (4D/Astro Luna) y co-ocurrencia de signo. - Montecarlo baseline y *lift*.

Fase 3 - Ajuste automático de pesos (online), versión de señales y dashboard semanal.

Fase 4 - Coberturas optimizadas por presupuesto; A/B de λ en softmax.

16) Notas específicas por juego

- **Astro Luna:** salida **4 dígitos + signo**. Candidatos deben incluir ambos; Markov de signo + dígito.
 - **4D semanales:** la generación sólo se activa **el día del sorteo** de cada lotería.
 - **Baloto/Revancha:** dos ejecuciones de **recolección** (23:05, 23:15) ya configuradas; generación a las 18:00 de L/X/S.
-

17) Mensajería y transparencia

- Cada reporte incluye: "RadarPremios es un sistema de análisis estadístico. No garantiza premios. Juega responsablemente".
 - Número/local de ayuda (si aplica) en pie de página.
-

Fin del documento

Axioma de Elección, medibilidad y selección reproducible (aplicado a Radar de Premios)

Resumen práctico: todo lo que hacemos para **elegir** candidatos debe ser **constructivo, medible y auditble**. Evitamos "elecciones arbitrarias" (tipo axioma de elección) y particiones patológicas (Vitali/Banach-Tarski) asegurando que todo criterio provenga de una **σ -álgebra finita/contable** sobre un espacio de estados discreto (combinaciones y ventanas finitas de sorteos) y que la **aleatoriedad** sea **pseudorandom reproducible**.

1) Espacio muestral y σ -álgebra segura

- Juegos discretos \Rightarrow espacio finito/contable:
- Super Astro Luna: $\Omega = \{0000..9999\} \times \{\text{signo} \in 12\} \rightarrow 120\,000$ puntos.
- Otros (quinielas/loterías) análogo.
- Señales/funciones **medibles**: dependen sólo de **ventanas finitas** de historia (p. ej., últimos W sorteos) y de funciones explícitas (frecuencias, Markov de orden k , rachas, gaps). Nada de comprensiones "no constructivas".

2) Regla de bien-orden constructiva

- Definimos un **orden total determinista** para romper empates y evitar "elección sin regla":
- **Score** (desc): suma ponderada de señales.
- **Tiebreak A:** lexicográfico por dígitos (0000...9999).
- **Tiebreak B:** orden fijo de signos (  ... ).
- Así existe siempre un "primer" candidato sin apelar a elección no constructiva.

3) Pseudorrandómico reproducible y auditado

- Usamos PRNG **PCG64** (o `numpy.random.Generator(PCG64DXSM)`) con **semilla determinista** por `fecha_juego + juego_id + versión_modelo`.
- Guardamos en BD: `seed`, versión de señales, vector de pesos y **log de decisiones**.
- Si se requiere muestreo (p. ej., *top-K* con ponderación), se hace **weighted sampling without replacement** con ese PRNG.

4) Señales compatibles (constructivas/medibles)

- **Frecuencias** por dígito/signo (hot/cold) en ventana W .
- **Markov(k)** sobre signo y/o últimos m dígitos (transiciones contables).
- **Gaps/racha**: tiempo desde última aparición por dígito/signo.
- **Penalizaciones**: repeticiones triviales (ej. 0000) según política.
- **Monte Carlo**: simulación de histórico bootstrap (semilla fija) para estimar estabilidad del score; **no** altera selección final, sólo da intervalos.

5) Controles LLN y sesgos (día T+1)

- **Chi-cuadrado/KS** sobre uniformidad de dígitos y signos (por juego/ventana), con *p-values* en reporte T+1.
- **Runs test** (Walsh) para chequear aleatoriedad elemental.
- **Drift monitor**: divergencia KL entre distribución esperada vs observada por semana.

6) Módulos nuevos (listos para el repo)

```
radar/
core/
    rng.py          # PRNG reproducible (PCG64) + utilidades de semilla
    choice_engine.py # orden total, muestreo ponderado SRSWOR
    signals.py      # señales constructivas (freq, Markov, gaps, etc.)
jobs/
    gen_candidates.py # usa signals + choice_engine para el día D
    eval_report.py   # T+1: compara ganadores vs generados + métricas
```

`core/rng.py`

```
from hashlib import sha256
from numpy.random import Generator, PCG64DXSM

DEFAULT_NS = "radar-v1"

def make_seed(date_iso: str, game_id: str, ns: str = DEFAULT_NS) -> int:
    h = sha256(f"{ns}|{game_id}|{date_iso}".encode()).digest()
    # tomar 8 bytes → entero 64 bits
    return int.from_bytes(h[:8], 'big', signed=False)
```

```
def make_rng(date_iso: str, game_id: str, ns: str = DEFAULT_NS) -> Generator:
    return Generator(PCG64DXSM(make_seed(date_iso, game_id, ns)))
```

core/choice_engine.py

```
from typing import List, Tuple
import numpy as np

# Candidate: (digits:str[4], signo:int[0..11], score:float)

SIGNO_ORDER = list(range(12)) # Aries..Piscis por índice

def tiebreak_key(c):
    digits, signo, score = c
    return (-score, digits, SIGNO_ORDER.index(signo))

def rank_candidates(candidatos: List[Tuple[str,int,float]]):
    return sorted(candidatos, key=tiebreak_key)

def weighted_sample_without_replacement(rng, items, weights, k):
    # método de Efraimidis-Spirakis
    w = np.asarray(weights, dtype=float)
    u = rng.random(len(items))
    keys = u ** (1.0 / w)
    idx = np.argpartition(-keys, k-1)[:k]
    return [items[i] for i in idx]
```

jobs/gen_candidates.py

```
# Pseudocódigo: generar K candidatos para juego/fecha
from radar.core.rng import make_rng
from radar.core.choice_engine import rank_candidates,
weighted_sample_without_replacement
from radar.core import signals

K = 50 # ejemplo

def generar_para(fecha_iso: str, juego_id: str):
    rng = make_rng(fecha_iso, juego_id)
    universo = signals.universo(juego_id) # p.ej. 120k combinaciones
    feats = signals.feature_table(juego_id, ventana=200) # constructivo
    scored = signals.score(universo, feats) # lista de (digits, signo,
    score)

    ranked = rank_candidates(scored)
    top = ranked[:min(2000, len(ranked))]

    pesos = [max(1e-6, c[2]) for c in top]
    elegidos = weighted_sample_without_replacement(rng, top, pesos, K)
```

```
# persistir: seed, versión, señales, elegidos, log de ranking
return elegidos
```

7) Auditoría y trazabilidad

- Persistimos: `seed`, `ns/version`, `features_hash`, `topN_snapshot`, `K_final`, PRNG usado.
- **Reproducibilidad**: rerun con misma DB/semina ⇒ mismos K.
- **Determinismo**: sin PRNG (p. ej. K=1) usar sólo `rank_candidates`.

8) Qué *no* hacemos (lecciones Axioma de Elección)

- No usamos particiones “exóticas” del espacio muestral (tipo Vitali) ni construcciones no medibles.
- No apoyamos decisiones en “elecciones” sin regla; siempre hay **regla explícita o semilla registrada**.
- No mezclamos capas continuas/geométricas (Banach–Tarski) en un dominio esencialmente discreto.

Impacto en el pipeline existente - `GEN D` : usa `gen_candidates.py` (seed diaria) → guarda lote + log. - `EVAL T+1` : añade pruebas LLN/chi-sq/runs y drift KL al informe. - `REPORT` : incluye trazabilidad de selección y estabilidad de señales.