Aquí tienes el informe de investigación solicitado, diseñado para estructurar la fase 2D con rigor y claridad.

[EXTENDED THOUGHT :: ANALYSIS]

1. **Separación de Responsabilidades:** El archivo `robustness_2D.yaml` no debe contener la *lógica* del backtest, sino la *definición del espacio de búsqueda* (el "qué", no el "cómo").
2. **Determinismo:** Para que los resultados sean reproducibles (G1), la generación de IDs de escenarios debe ser independiente del orden de ejecución. Usaré un enfoque de "Canonical String + Hash" para los IDs.
3. **Compatibilidad:** El schema debe anticipar el "Walk-Forward" (paso 3) y la "Sensibilidad" (paso 4) sin obligar a implementarlos todos en el paso 2. Usaremos secciones opcionales.
4. **Interfaz con Runner:** El runner actuará como un orquestador que inyecta dependencias. La clave es cómo pasar las perturbaciones (noise, gaps) al generador de datos sin ensuciar la lógica principal del bot.

[EDITOR :: report/GM-2D-02-2.1_research.md]

````
# GM-2D-02-2.1 — Research: Schema de Robustez 2D

**ID:** GM-2D-02-2.1
**Fecha:** 2025-12-27
**Objetivo:** Definir el contrato de configuración para la
validación de robustez (Fase 2D).

---

## 1. Propuesta de Schema YAML (`configs/robustness_2D.yaml`)

Este diseño centraliza la definición de escenarios, separando la
configuración base de las perturbaciones estocásticas.

```yaml
meta:
  version: "1.0"
  description: "Definición de escenarios de estrés para validación
2D (Anti-overfit)"
  target_config: "configs/best_params_2C.json"  # Configuración
'Best' a validar
````
````

```
  base_config: "risk_rules.yaml"                # Configuración
'Baseline' (control)

# Configuración Global de Semillas
# La seed global (42) se usa para procesos deterministas internos si
no se especifica otra.
# 'sweep_seeds' define la lista explícita para probar estabilidad
estadística.
seeds:
  global_default: 42
  sweep_list: [42, 7, 13, 21, 99, 123, 777, 1001]

# Definición del Entorno de Datos Sintéticos
environment:
  assets_mode: "single"      # single | multi
  default_duration_days: 365 # Ventana base
  initial_capital: 10000.0

# Espacio de Perturbaciones (Cartesian Product)
# El runner generará escenarios combinando seeds x estas listas.
perturbations:
  noise_sigma:           [0.0, 0.001, 0.0025] # Ruido gaussiano
añadido al precio
  gap_prob:              [0.0, 0.005, 0.01]   # Probabilidad de
saltos de precio
  missing_prob:          [0.0, 0.005]         # Probabilidad de
velas perdidas (NaNs)
  vol_regime_multiplier: [1.0, 1.5, 2.0]      # Multiplicador de
volatilidad base

# Configuración de Walk-Forward (Opcional, para Paso 3)
walk_forward:
  enabled: false
  n_folds: 4
  train_ratio: 0.7  # No se usa para re-entrenar en 2D, sino para
evaluar OOS puro

# Gates de Aprobación (Constraints)
# Si se violan, el escenario se marca como FAILED.
constraints:
  risk:
    max_drawdown_hard_limit: 0.15  # 15% MaxDD absoluto permitido
```

```
(overrideable)
    max_drawdown_tolerance: 1.1    # No puede ser peor que 1.1x del
baseline
  stability:
    min_ok_ratio: 0.95             # 95% de los escenarios deben
pasar sin error técnico
    max_nan_ratio: 0.0             # Métricas críticas no pueden ser
NaN

# Configuración del Runner
execution:
  mode_defaults:
    quick:
      max_scenarios: 10            # Limita el producto cartesiano
para tests rápidos
      seeds_subset: [42, 7]
    full:
      max_scenarios: 1000
      seeds_subset: "all"
```

2. Convención de Identificación de Escenarios

Para garantizar reproducibilidad y trazabilidad en los logs, cada escenario tendrá un ID único determinista.

Formato: `S{seed}_N{noise}_G{gap}_M{miss}_V{vol}` **Ejemplo:** `S42_N0.001_G0.0_M0.005_V1.5`

Algoritmo de Generación de ID

1. Tomar los valores de la permutación actual.
2. Formatear floats con precisión fija (ej. `.4f`) para evitar ambigüedades (0.1 vs 0.10000001).
3. Concatenar con separadores.
4. *(Opcional)* Si el string es muy largo, generar un hash SHA-256 corto (8 chars) y guardarlo como `scenario_hash`, manteniendo el string legible como `scenario_label`.

3. Mapeo: YAML → Runner → Overrides

La siguiente tabla define cómo se traduce cada clave del YAML a la ejecución del código.

Sección YAML	Argumento Runner (Internal)	Efecto en Ejecución
`seeds.sweep_list`	`current_seed`	Inicializa `numpy.random.seed` y el generador de precios sintéticos.
`perturbations.noise_sigma`	`noise_std`	Se pasa al generador de datos: `price = price * (1 + N(0, noise_std))`
`perturbations.gap_prob`	`gap_prob`	Se pasa al generador: Inyecta saltos artificiales en OHLC.
`perturbations.missing_prob`	`nan_prob`	Se pasa al generador: Convierte filas aleatorias a NaN (simula fallo de API).
`perturbations.vol_regime`	`vol_mult`	Escala la volatilidad base del generador Browniano/GARCH.
`constraints.risk.max_dd`	`dd_limit`	Se usa en la fase de **validación post-run**. Si `result.max_dd < -dd_limit`, `status=FAIL`.
`meta.target_config`	`best_params_dict`	Se carga el JSON y se aplica sobre `risk_rules` en memoria usando la lógica de `apply_calibration`.

4. Reglas de Validación (Schema & Logic)

Antes de iniciar el bucle de escenarios, el runner debe validar el YAML:

Errores (Bloqueantes)

1. **Tipos Incorrectos:** `seeds.sweep_list` no es una lista de enteros.
2. **Probabilidades Inválidas:** `gap_prob`, `missing_prob` fuera de `[0, 1]`.
3. **Configuración Faltante:** No se encuentra el archivo `target_config` (best params).
4. **Constraints Ilógicos:** `min_ok_ratio > 1.0` o `max_nan_ratio < 0.0`.

Warnings (Informativos)

1. **Explosión Combinatoria:** Si el producto cartesiano de perturbaciones x seeds > 500 escenarios y el modo es `full`.
2. **Seed Global vs Sweep:** Si `global_default` no está en `sweep_list` (no es error, pero se notifica).

5. Pseudo-CLI del Runner

Interfaz propuesta para `tools/run_robustness_2D.py`:

```
# Ejecución Rápida (Smoke Test / CI)
python tools/run_robustness_2D.py \
    --config configs/robustness_2D.yaml \
    --mode quick \
    --out-dir report/robustness_2D/quick_run

# Ejecución Completa (Nightly / Release)
python tools/run_robustness_2D.py \
    --config configs/robustness_2D.yaml \
    --mode full \
    --parallel 4  # Opcional: ejecución paralela
```

Flujo Interno del Runner

1. **Load:** Carga YAML y valida schema.
2. **Plan:** Genera la lista de escenarios (objetos `ScenarioSpec`) según el `mode` (quick/full).
3. **Loop:** Para cada escenario:

    a. Generar datos sintéticos con `seed` y `perturbations`.
    b. **Run A (Baseline):** Ejecutar backtest con `risk_rules.yaml`.
    c. **Run B (Candidate):** Aplicar `best_params_2C.json` en memoria y ejecutar backtest.
    d. **Compare:** Calcular deltas y verificar constraints.
    e. **Log:** Guardar fila en `results.csv`.
4. **Report:** Generar `summary.md` y verificar Gates globales (ej. `ok_ratio`). ```

[TERMINAL :: INSTRUCTION] Usuario, he generado el diseño del schema y la arquitectura del runner.

1. Crea el archivo `report/GM-2D-02-2.1_research.md` con el contenido de arriba.
2. Luego, crea el archivo de configuración real `configs/robustness_2D.yaml` basándote en este diseño.
3. Finalmente, implementa el runner `tools/run_robustness_2D.py` siguiendo la especificación.

¿Procedemos con el paso 1?