DS-2D-02-2.1 — Diseño de Contrato YAML para Robustez 2D

1. Estructura YAML Propuesta (`configs/robustness_2D.yaml`)

```
# configs/robustness_2D.yaml
meta:
  version: "2D.1"
  description: "Escenarios de robustez para validación anti-overfit
de parámetros de riesgo"
  generated_date: "2025-12-27"
  schema_version: "1.0"

# ----------------------------------------------------------------
------------
# 1. BASELINE - Configuraciones de referencia
# ----------------------------------------------------------------
------------
baseline:
  risk_configs:
    current: "risk_rules.yaml"                # Config actual
(post-2C)
    best: "configs/best_params_2C.json"       # Mejores parámetros
calibrados

  backtest_defaults:
    start_date: "2024-01-01"
    periods: 252                              # Días hábiles (~1
año)
    assets_mode: "multi"                      # "multi" o "single"
(para pruebas rápidas)
    initial_capital: 10000.0

# ----------------------------------------------------------------
------------
# 2. ENGINE - Configuración del motor de evaluación
# ----------------------------------------------------------------
------------
engine:
  reproducibility:
    default_seed: 42                          # Seed principal para
generación
    seed_list: [42, 7, 13, 21, 99, 123, 777, 1001]  # Lista de seeds
```

```
para robustez

  walkforward:
    enabled: true
    folds: 4                                    # Número de folds
(ventanas)
    validation_mode: "rolling"                  # "rolling" o
"expanding"
    min_train_days: 126                         # Mínimo días
entrenamiento

  execution:
    max_workers: 1                              # Para futura
paralelización (actualmente 1)
    timeout_per_scenario_seconds: 300           # Timeout por
escenario

# ------------------------------------------------------------------
------------
# 3. RISK CONSTRAINTS - Límites que NO deben violarse
# ------------------------------------------------------------------
------------
risk_constraints:
  hard_limits:
    max_drawdown_absolute: 0.15                 # DD máximo absoluto
permitido en cualquier escenario
    min_ok_ratio: 0.95                          # % mínimo de
escenarios exitosos por variante
    max_nan_ratio: 0.0                          # % máximo de NaN en
métricas clave

  fragility_gates:
    param_sensitivity_threshold: 0.05           # Cambio máximo
permitido en métricas con ±5% perturbación
    worst_case_dd_multiplier: 1.5               # Multiplicador
máximo del DD baseline en worst-case

# ------------------------------------------------------------------
------------
# 4. SWEEP - Definición de perturbaciones por escenario
# ------------------------------------------------------------------
------------
```

```
sweep:
  # Modos de ejecución (definen granularidad)
  modes:
    quick:
      max_scenarios: 30                        # Límite para
ejecución rápida
      subset_seeds: [42, 7, 13]                # Seeds para modo
quick

    full:
      max_scenarios: 1000                      # Límite para
ejecución completa
      use_all_seeds: true

  # Perturbaciones de datos (aplicadas a la serie de precios)
  data_perturbations:
    noise:
      enabled: true
      sigmas: [0.0, 0.001, 0.0025]            # Desviación estándar
del ruido gaussiano

    gaps:
      enabled: true
      probabilities: [0.0, 0.005, 0.01]       # Probabilidad de gap
por día
      gap_size_days: [1, 2]                    # Duración del gap en
días

    missing:
      enabled: true
      probabilities: [0.0, 0.005]              # Probabilidad de
dato faltante

    volatility_regimes:
      enabled: true
      multipliers: [1.0, 1.5, 2.0]             # Multiplicadores de
volatilidad
      regime_duration_days: [63, 126]          # Duración del
régimen en días

  # Perturbaciones de parámetros (para sensibilidad local)
  param_perturbations:
```

```
    enabled: true
    variations:
      - type: "individual"
        params: ["atr_multiplier", "min_stop_pct", "soft_limit_pct",
"hard_limit_pct", "size_multiplier_soft", "cap_factor"]
        percentages: [-0.05, -0.10, 0.05, 0.10]  # ±5%, ±10%

      - type: "random_combinations"
        count: 10                                # Número de
combinaciones aleatorias
        max_percentage_change: 0.15              # Cambio máximo por
parámetro
        seed: 42                                 # Seed para
aleatoriedad controlada

# ------------------------------------------------------------------
------------
# 5. OUTPUT - Configuración de salida y reportes
# ------------------------------------------------------------------
------------
output:
  base_dir: "report/robustness_2D"

  files:
    results_csv: "results.csv"                # Resultados
detallados por escenario
    summary_md: "summary.md"                  # Resumen ejecutivo
    run_meta_json: "run_meta.json"            # Metadatos de
ejecución
    raw_metrics_jsonl: "raw_metrics.jsonl"    # Métricas brutas
(JSONL)
    errors_jsonl: "errors.jsonl"              # Errores (JSONL)
    walkforward_csv: "walkforward_results.csv" # Resultados walk-
forward
    sensitivity_csv: "sensitivity_results.csv" # Resultados de
sensibilidad

  reporting:
    generate_plots: false                     # Para futura
extensión (gráficos)
    include_param_details: true
    format_numbers:
```

```
      decimals: 4
      percent_decimals: 2
```

2. Convención de `scenario_id`

Cada escenario debe tener un identificador único y determinista basado en sus parámetros:

```
# Formato: SCENARIO_TYPE|SEED|PERTURBATION_HASH
# Ejemplos:
#   "BASELINE|42|NO_PERTURBATION"
#   "DATA_PERTURB|7|NOISE_0.001_GAP_0.005"
#   "PARAM_PERTURB|42|ATR_MULT_+5%"

def generate_scenario_id(scenario_type: str, seed: int, params_dict:
dict) -> str:
    """
    Genera un scenario_id legible y único.

    Args:
        scenario_type: Uno de ["BASELINE", "DATA_PERTURB",
"PARAM_PERTURB", "WALKFORWARD"]
        seed: Semilla numérica
        params_dict: Diccionario con parámetros del escenario

    Returns:
        scenario_id string
    """
    if scenario_type == "BASELINE":
        return f"BASELINE|{seed}|NO_PERTURBATION"

    elif scenario_type == "DATA_PERTURB":
        # Ejemplo:
"DATA_PERTURB|42|NOISE_0.001_GAP_0.005_MISSING_0.0_VOL_1.5"
        parts = []
        if params_dict.get("noise_sigma", 0) > 0:
            parts.append(f"NOISE_{params_dict['noise_sigma']:.4f}")
        if params_dict.get("gap_prob", 0) > 0:
            parts.append(f"GAP_{params_dict['gap_prob']:.3f}")
        if params_dict.get("missing_prob", 0) > 0:
```

```
parts.append(f"MISSING_{params_dict['missing_prob']:.3f}")
        if params_dict.get("vol_multiplier", 1.0) != 1.0:
            parts.append(f"VOL_{params_dict['vol_multiplier']:.1f}")

        if not parts:
            parts.append("NO_PERTURBATION")

        return f"DATA_PERTURB|{seed}|{'__'.join(parts)}"

    elif scenario_type == "PARAM_PERTURB":
        # Ejemplo: "PARAM_PERTURB|42|ATR_MULT_+5%__CAP_FACTOR_-10%"
        changes = []
        for param, value in params_dict.items():
            if param.endswith("_change_pct"):
                base_param = param.replace("_change_pct", "")
                changes.append(f"{base_param.upper()}_{value:+d}%")

        return f"PARAM_PERTURB|{seed}|{'__'.join(changes)}"

    elif scenario_type == "WALKFORWARD":
        # Ejemplo: "WALKFORWARD|42|FOLD_2_OOS"
        fold = params_dict.get("fold", 0)
        return f"WALKFORWARD|{seed}|FOLD_{fold}"
```

Reglas para `scenario_id`:

1. **Determinista**: Mismos inputs → mismo `scenario_id`
2. **Legible**: Humanos pueden identificar el tipo de perturbación
3. **Único**: No colisiones dentro de una ejecución
4. **Máximo 128 caracteres**: Para compatibilidad con sistemas de archivos

3. Invariantes (Reglas que NO deben violarse)

3.1. Invariantes de Configuración

1. **No modificar `risk_rules.yaml` original**:
    a. El runner debe trabajar con copias temporales o configuraciones en memoria.
    b. Si se requiere modificar, crear `risk_rules_candidate.yaml` y validar antes de escribir.
2. **Semillas controladas**:

    a. Cada escenario debe usar exactamente la seed especificada.
    b. Reset de `np.random.seed(seed)` al inicio de cada escenario.
3. **Rangos válidos de parámetros**:
    a. `atr_multiplier > 0`
    b. `0 < min_stop_pct < 1`
    c. `0 < soft_limit_pct ≤ hard_limit_pct < 1`
    d. `0 ≤ size_multiplier_soft ≤ 1`
    e. `0 ≤ cap_factor ≤ 1`

3.2. Invariantes de Ejecución

4. **Aislamiento de escenarios**:
    a. Un escenario fallido no debe afectar la ejecución de otros.
    b. Cada escenario debe ejecutarse en un contexto limpio (reset de variables globales).
5. **Consistencia de métricas**:
    a. Si `max_drawdown = 0`, `calmar_ratio` debe ser 0 (no NaN o Inf).
    b. `win_rate ∈ [0, 1]`
    c. `pct_time_hard_stop ∈ [0, 1]`
6. **Trazabilidad completa**:
    a. Cada escenario debe registrar todos los inputs (seed, perturbaciones, parámetros).
    b. Cada resultado debe ser reproducible dado el `scenario_id`.

3.3. Invariantes de Output

7. **Estructura de archivos**:
    a. El directorio de salida debe crearse si no existe.
    b. Los archivos CSV deben tener headers consistentes.
    c. Los JSON deben ser válidos (parseables).
8. **Manejo de errores**:
    a. Errores no deben interrumpir la ejecución completa (continue-on-error).
    b. Los errores deben registrarse en `errors.jsonl` con stack trace.

4. Tests Sugeridos

4.1. Unit Tests para el Runner

```
# tests/test_robustness_runner_2D.py
import pytest
from pathlib import Path
import tempfile
```

```
import yaml

def test_runner_loads_config():
    """Verifica que el runner puede cargar el YAML de
configuración."""
    config_path = Path("configs/robustness_2D.yaml")
    assert config_path.exists()

    with open(config_path, 'r') as f:
        config = yaml.safe_load(f)

    assert 'baseline' in config
    assert 'sweep' in config

def test_scenario_id_generation():
    """Verifica que la generación de scenario_id es determinista."""
    from tools.run_robustness_2D import generate_scenario_id

    # Test baseline
    sid1 = generate_scenario_id("BASELINE", 42, {})
    sid2 = generate_scenario_id("BASELINE", 42, {})
    assert sid1 == sid2 == "BASELINE|42|NO_PERTURBATION"

    # Test data perturbation
    params = {"noise_sigma": 0.001, "gap_prob": 0.005}
    sid3 = generate_scenario_id("DATA_PERTURB", 7, params)
    assert sid3.startswith("DATA_PERTURB|7|")

def test_runner_creates_output_files():
    """Smoke test: ejecuta runner en modo quick y verifica archivos
de salida."""
    import subprocess

    with tempfile.TemporaryDirectory() as tmpdir:
        cmd = [
            "python", "tools/run_robustness_2D.py",
            "--mode", "quick",
            "--out-dir", str(Path(tmpdir) / "output")
        ]

        result = subprocess.run(cmd, capture_output=True, text=True)
```

```
        # Verifica que no haya errores fatales
        assert result.returncode == 0, f"Runner failed:
{result.stderr}"

        output_dir = Path(tmpdir) / "output"
        assert (output_dir / "results.csv").exists()
        assert (output_dir / "summary.md").exists()
        assert (output_dir / "run_meta.json").exists()

def test_perturbation_ranges():
    """Verifica que las perturbaciones estén dentro de rangos
válidos."""
    from tools.run_robustness_2D import validate_perturbation_params

    # Casos válidos
    valid_params = {
        "noise_sigma": 0.001,
        "gap_prob": 0.01,
        "missing_prob": 0.005,
        "vol_multiplier": 1.5
    }
    assert validate_perturbation_params(valid_params) == []

    # Casos inválidos
    invalid_params = {"noise_sigma": -0.1}  # Negativo
    errors = validate_perturbation_params(invalid_params)
    assert len(errors) > 0
```

4.2. Tests de Integración

```
# tests/test_robustness_integration_2D.py
def test_walkforward_fold_generation():
    """Verifica que walkforward genera el número correcto de
folds."""
    from tools.run_robustness_2D import generate_walkforward_folds

    prices = pd.DataFrame(...)  # Serie sintética de 252 días
    folds = generate_walkforward_folds(prices, n_folds=4)

    assert len(folds) == 4
    for train, test in folds:
```

```
        assert len(train) > 0
        assert len(test) > 0
        # No overlap entre train y test
        assert train.index.max() < test.index.min()

def test_param_sensitivity_calculation():
    """Verifica cálculo de sensibilidad de parámetros."""
    from tools.run_robustness_2D import calculate_sensitivity

    baseline_metrics = {"sharpe_ratio": 1.0, "max_drawdown": -0.1}
    perturbed_metrics = {"sharpe_ratio": 0.95, "max_drawdown": -
0.12}

    sensitivity = calculate_sensitivity(baseline_metrics,
perturbed_metrics)

    assert "sharpe_ratio_change_pct" in sensitivity
    assert "max_drawdown_change_pct" in sensitivity
    assert sensitivity["sharpe_ratio_change_pct"] == -5.0  # -5%
```

4.3. Edge Cases a Considerar

1. **Seed 0 y valores límite**:
    a. Probar con seed=0 (válido pero poco común).
    b. Probar con `gap_prob=0.0` y `gap_prob=1.0` (límites).
2. **Perturbaciones extremas**:
    a. `noise_sigma` muy alto (ej: 0.1) que puede hacer que los precios sean negativos.
    b. `vol_multiplier=0.5` (reducción de volatilidad, no solo aumento).
3. **Series temporales muy cortas**:
    a. Probar con `periods=10` (insuficiente para cálculos de métricas).
    b. Walkforward con más folds que días disponibles.
4. **Parámetros en límites**:
    a. `hard_limit_pct = soft_limit_pct` (zona de transición).
    b. `cap_factor=0.0` (ninguna posición permitida).
    c. `cap_factor=1.0` (máxima exposición).
5. **Manejo de NaN/Inf**:
    a. Serie de precios con valores NaN (debería limpiarse o rechazarse).
    b. Métricas que devuelvan Inf (ej: Sharpe con volatilidad 0).
6. **Consistencia entre modos**:
    a. Verificar que `quick` mode es subconjunto de `full` mode.

b. Resultados deben ser idénticos para escenarios comunes.

4.4. Test de CI (Continuous Integration)

```
# .github/workflows/robustness_2D.yml (extensión del CI existente)
name: Robustness 2D Quick Check

on:
  push:
    branches: [feature/2D_param_robustness]
  pull_request:
    branches: [main]

jobs:
  robustness-quick:
    runs-on: ubuntu-latest
    steps:
      - uses: actions/checkout@v3

      - name: Set up Python
        uses: actions/setup-python@v4
        with:
          python-version: '3.12'

      - name: Install dependencies
        run: |
          python -m pip install --upgrade pip
          pip install -r requirements.txt

      - name: Run robustness quick check
        run: |
          python tools/run_robustness_2D.py \
            --mode quick \
            --out-dir report/.ci/robustness_2D_quick

      - name: Validate outputs
        run: |
          # Verificar que se generaron los archivos esperados
          test -f report/.ci/robustness_2D_quick/results.csv
          test -f report/.ci/robustness_2D_quick/summary.md

          # Verificar que no hay errores críticos
          ! grep -q '"status":"error"'
```

`report/.ci/robustness_2D_quick/results.csv`

5. Recomendaciones de Implementación

1. **Modularidad**: Separar el runner en módulos:
    a. `scenario_generator.py`: Genera escenarios desde el YAML
    b. `perturbation_engine.py`: Aplica perturbaciones a datos y parámetros
    c. `metrics_aggregator.py`: Calcula métricas agregadas y checks
2. **Configuración validada**: Usar `pydantic` o `marshmallow` para validar el esquema YAML.
3. **Logging estructurado**: Usar JSON logging para fácil análisis posterior.
4. **Cache de resultados**: Para desarrollo, cachear resultados de escenarios ya ejecutados.
5. **Paralelización futura**: Diseñar para soportar `multiprocessing` o `concurrent.futures`.

Este diseño proporciona una base sólida para la validación de robustez 2D, con énfasis en reproducibilidad, trazabilidad y detección temprana de fragilidad en los parámetros calibrados.