README_EXECUTIVE

TCDS–EDGE | AERC (Artefacto de Evaluación de Riesgo Causal)

Paquete TRL-9 estilo demostración (reproducible, offline-friendly)

December 24, 2025

1. Propósito

Este paquete entrega una demostración **reproducible** (one-shot) de un análisis causal pre-ruptura basado en **entropía** y **persistencia temporal** (tiempo causal), usando exclusivamente **datos públicos**:

- **USGS** (metadata del evento por ID).
- **IRIS/FDSN** (traza sísmica por estación y canal; endpoint timeseries).

El objetivo no es *predecir la magnitud* (M_w) sino cuantificar una variable de **riesgo causal** previa a la ruptura, operacionalizable para infraestructura crítica bajo criterios de seguridad.

2. Qué contiene (entregables)

- `CODE/edge_demo.py`: ejecutable one-shot que genera un **JSON firmado** (SHA256).
- `tcds_edge_result.json`: salida principal con métricas, decisión y proveniencia.
- `TCDS_Overlay_Impact.pdf/.png`: gráfica estática de alto impacto (tiempo cronológico vs lectura causal).
- `GLOBAL_SUMMARY.pdf` y `TCDS_DEFENSA_PLANETARIA_SÍSMICA.pdf`: contexto operativo y lectura ejecutiva.
- `MANIFEST.sha256`: manifiesto de integridad (hash por archivo).

3. Qué demuestra (veracidad y reproducibilidad)

- **Reproducibilidad**: con el mismo evento/estación/ventana, el JSON canónico produce el mismo hash SHA256.
- **Trazabilidad**: el JSON incluye endpoints, reglas, ventanas y parámetros de cálculo.
- **Separación tM vs tC**:
  - **tM (tiempo cronológico)**: ventana fija alrededor del evento (pre/post).

  - **tC (tiempo causal)**: ventanas deslizantes de entropía y su persistencia (lock).
- **Criterio de honestidad (E-Veto)**: la señal sólo se considera operativa si existe caída de entropía suficiente y persistencia mínima.

4. Definición operacional (AERC)

El AERC mide:

- **Caída de entropía**: $\Delta H(t) = H(t) - H_{\text{baseline}}$ (bits).
- **Magnitud causal**: $M_c = |\min \Delta H|$.
- **Persistencia de bloqueo**: $t_{\text{lock}}(\theta) =$ mayor corrida continua con $\Delta H \leq \theta$.

Reglas de decisión (por defecto; ajustables):

- Umbral operativo: $\Delta H \leq -0.15$ (trigger).
- Umbral de veto: $\Delta H \leq -0.20$ (E-Veto).
- Persistencia mínima: válvulas $\geq 4\,s$; trenes $\geq 10\,s$.

5. Cómo correrlo (local)

Requisitos: Python 3.9+, `requests`, `numpy`.

```
pip install requests numpy
python CODE/edge_demo.py \
  --event us6000m0xl \
  --net IU --sta MAJO --loc 00 --cha BHZ \
  --pre 120 --post 600 \
  --win 10 --step 1 \
  --emit-series \
  --out tcds_edge_result.json
```

Salida esperada:

- JSON con campos: `event`, `station`, `windows`, `metrics`, `risk`, `decision_rules`, `provenance`, `crypto`.
- `crypto.sha256_of_canonical_json` cambia ante cualquier modificación de código o parámetros (inmutabilidad práctica).

6. Cómo correrlo (Google Colab)

1. Suba `CODE/edge_demo.py` al entorno o monte Drive.
2. Instale dependencias:

   ```
   !pip -q install requests numpy
   ```

3. Ejecute el comando del apartado anterior (en celda `!python ...`).

7. Interpretación del resultado

El paquete clasifica el riesgo causal en niveles discretos, con recomendación táctica. Ejemplo de lectura:

Campo	Significado
`metrics.min_dH_bits`	Mínimo de ΔH (caída máxima observada en ventana tM).
`metrics.Mc`	$M_c = \lvert \min \Delta H \rvert$, intensidad causal.
`decision_rules.t_persist_trigger_seconds`	Persistencia requerida bajo trigger (segundos).
`risk.e_veto_pass`	Verdadero si $\min \Delta H \leq$ umbral E-Veto.
`risk.recommended_action`	Acción sugerida: `IDLE`, `PRE_ALERT`, `KILL`, `KILL_PLUS_EVACUATE`.

8. Alcance y límites (claridad institucional)

- Este paquete es una **demostración auditada** de lectura causal pre-ruptura **con datos públicos**.
- No sustituye sistemas EEW post-P; es un enfoque **complementario** centrado en **pre-ruptura**.
- El **manejo completo de falsos positivos**, calibración por región, fusión multi-estación y gobernanza operativa forman parte de un **módulo no público** destinado a implementación bajo colaboración institucional.

9. Integridad criptográfica

- `MANIFEST.sha256` verifica integridad de archivos del paquete.
- Cada corrida produce un JSON con hash SHA256 del contenido canónico:

```
json.dumps(out, sort_keys=True, separators=(",", ":"), ensure_ascii=False).encode("utf-8")
```

- Recomendación: versionar en Git y registrar releases firmados (tags) para auditoría.

10. Contacto y siguiente paso

Si la institución desea:

- Ejecutar en privado,
- Replicar con sus estaciones/sensores,
- Integrar política de falsos positivos y gobernanza operativa,

se propone una **sesión técnica** para acordar: alcance, criterios de seguridad, datasets de validación ciega y bitácora criptográfica de corridas.

Nota: Este README no publica módulos de calibración avanzada ni criterios internos de reducción de falsas alarmas.