```
"""
E-Veto Engine (Filtro de Honestidad)
Teoría Cromodinámica Sincrónica (TCDS)

Implementa el criterio de admisibilidad ontológica:
- LI ≥ 0.90
- R > 0.95
- RMSE_SL < 0.10
- ΔH ≤ -0.20

Autor: Genaro Carrasco Ozuna
ORCID: 0009-0005-6358-9910
"""

from typing import Dict, Literal
from dataclasses import dataclass
import hashlib
import json

# Umbrales canónicos TCDS (congelados)
CANONICAL_THRESHOLDS = {
    "LI_min": 0.90,
    "R_min": 0.95,
    "RMSE_SL_max": 0.10,
    "DH_max": -0.20
}

VerdictType = Literal["ACCEPT", "REJECT", "SILENCE", "NOT_EVALUABLE"]

@dataclass
class EVetoResult:
    """Resultado del Filtro de Honestidad"""
    verdict: VerdictType
    metrics: Dict[str, float]
    reason: str
    config_hash: str

    def to_dict(self) -> Dict:
        return {
            "verdict": self.verdict,
            "metrics": self.metrics,
            "reason": self.reason,
            "config_hash": self.config_hash
        }


def compute_config_hash(thresholds: Dict[str, float]) -> str:
    """
    Calcula SHA-256 del config para trazabilidad forense.
    Cualquier cambio en umbrales cambia el hash.
    """
    config_str = json.dumps(thresholds, sort_keys=True)
    return hashlib.sha256(config_str.encode()).hexdigest()[:16]


def apply_eveto(
    li: float,
    r: float,
    rmse_sl: float,
    delta_h: float,
```

```
    thresholds: Dict[str, float] = None
) -> EVetoResult:
    """
    Aplica el Filtro de Honestidad (E-Veto).

    Args:
        li: Locking Index [0, 1]
        r: Correlación temporal [0, 1]
        rmse_sl: Root Mean Square Error Slope-Lock [0, ∞)
        delta_h: Cambio de entropía normalizada (-∞, ∞)
        thresholds: Umbrales personalizados (default: CANONICAL_THRESHOLDS)

    Returns:
        EVetoResult con veredicto y razón

    Lógica de decisión:
    - ACCEPT: Cumple KPIs de coherencia Y reducción entrópica
    - REJECT: Falla KPIs Y entropía (ruido confirmado)
    - SILENCE: Cumple parcialmente (ambigüedad, prudencia)
    - NOT_EVALUABLE: Datos inválidos
    """
    if thresholds is None:
        thresholds = CANONICAL_THRESHOLDS

    config_hash = compute_config_hash(thresholds)

    # Validación de datos
    if any(x is None or not isinstance(x, (int, float)) for x in [li, r, rmse_sl, delta_h]):
        return EVetoResult(
            verdict="NOT_EVALUABLE",
            metrics={"LI": li, "R": r, "RMSE_SL": rmse_sl, "ΔH": delta_h},
            reason="Datos inválidos o ausentes",
            config_hash=config_hash
        )

    metrics = {
        "LI": float(li),
        "R": float(r),
        "RMSE_SL": float(rmse_sl),
        "ΔH": float(delta_h)
    }

    # Evaluación de KPIs de coherencia
    kpi_ok = (
        li >= thresholds["LI_min"] and
        r > thresholds["R_min"] and
        rmse_sl < thresholds["RMSE_SL_max"]
    )

    # Evaluación de reducción entrópica
    entropy_ok = delta_h <= thresholds["DH_max"]

    # Matriz de decisión
    if kpi_ok and entropy_ok:
        return EVetoResult(
            verdict="ACCEPT",
            metrics=metrics,
            reason=(
                f"Ventana Q-driven: coherencia alta (LI={li:.3f}, R={r:.3f}, "
                f"RMSE_SL={rmse_sl:.3f}) con reducción entrópica forzada (ΔH={delta_h:.3f})"
            ),
            config_hash=config_hash
        )
```

```
    elif not kpi_ok and not entropy_ok:
        return EVetoResult(
            verdict="REJECT",
            metrics=metrics,
            reason=(
                f"Ventana φ-driven: coherencia insuficiente Y entropía ascendente/nula. "
                f"Ruido confirmado."
            ),
            config_hash=config_hash
        )

    else:
        # Zona borderline: algunos criterios pasan, otros no
        if kpi_ok and not entropy_ok:
            detail = "Coherencia aparente pero sin reducción entrópica (posible apofenia)"
        else:
            detail = "Reducción entrópica sin coherencia estructural sostenida"

        return EVetoResult(
            verdict="SILENCE",
            metrics=metrics,
            reason=f"Zona borderline: {detail}. Insuficiente para admisibilidad ontológica.",
            config_hash=config_hash
        )


def batch_eveto(metrics_list: list[Dict[str, float]], thresholds: Dict = None) -> list[EVetoResult]:
    """
    Aplica E-Veto a múltiples ventanas.

    Args:
        metrics_list: Lista de dicts con claves 'LI', 'R', 'RMSE_SL', 'ΔH'
        thresholds: Umbrales (default: canónicos)

    Returns:
        Lista de EVetoResult
    """
    results = []
    for m in metrics_list:
        result = apply_eveto(
            li=m.get('LI'),
            r=m.get('R'),
            rmse_sl=m.get('RMSE_SL'),
            delta_h=m.get('ΔH'),
            thresholds=thresholds
        )
        results.append(result)
    return results


# Test de auto-validación
if __name__ == "__main__":
    print("=== Test E-Veto Engine ===\n")

    test_cases = [
        {
            "name": "Q-driven válido",
            "LI": 0.95, "R": 0.98, "RMSE_SL": 0.05, "ΔH": -0.30
        },
        {
            "name": "φ-driven ruido",
            "LI": 0.70, "R": 0.85, "RMSE_SL": 0.25, "ΔH": 0.10
```

```
    },
    {
        "name": "Borderline (coherencia sin entropía)",
        "LI": 0.92, "R": 0.97, "RMSE_SL": 0.08, "ΔH": -0.10
    },
    {
        "name": "Dataset trampa (debe fallar)",
        "LI": 0.88, "R": 0.94, "RMSE_SL": 0.15, "ΔH": 0.05
    }
]

for tc in test_cases:
    name = tc.pop("name")
    result = apply_eveto(**tc)
    print(f"[{name}]")
    print(f"  Veredicto: {result.verdict}")
    print(f"  Razón: {result.reason}")
    print(f"  Config hash: {result.config_hash}\n")
```