IACADEMY

ENTERPRISE AGENT ENGINEERING

MÓDULO 07

Evaluación Continua

Principal

iacedemy.com — 2026

MÓDULO 07

Evaluación Continua

Nivel: Principal / Arquitecto

Autor: Alicia Sanz

Publicación: Julio 2026

Plataforma: iacedemy.com

> “No basta con que el agente funcione hoy. Debe medirse continuamente porque el mundo cambia.”

6.1 Métricas Enterprise

Por Agente

```
AGENT_METRICS = {
    "task_completion_rate": "% tareas completadas correctamente",
    "avg_latency_ms": "Latencia media de respuesta",
    "cost_per_task_eur": "Coste medio por tarea",
    "hallucination_rate": "% respuestas con info inventada",
    "groundedness_score": "% respuestas basadas en fuentes",
    "tool_accuracy": "% tool calls correctas",
    "escalation_rate": "% tareas escaladas a humano",
    "user_satisfaction": "CSAT medio (1-5)",
}
```

Por Tenant

```
TENANT_METRICS = {
    "active_agents": "Agentes activos este mes",
    "total_executions": "Ejecuciones totales",
    "total_cost_eur": "Coste total acumulado",
    "budget_utilization": "% del budget usado",
    "security_incidents": "Incidentes de seguridad",
    "compliance_score": "% controles cumplidos",
    "avg_response_quality": "Calidad media de respuestas",
}
```

6.2 Datasets a Escala

Dataset Versioning

```
from datetime import datetime

class EvalDatasetManager:
    def __init__(self, storage_path: str):
        self.path = storage_path

    def create_version(self, name: str, cases: list[dict]) -> str:
        version = datetime.now().strftime("%Y%m%d_%H%M%S")
        path = f"{self.path}/{name}/{version}.jsonl"
        with open(path, "w") as f:
            for case in cases:
                f.write(json.dumps(case, ensure_ascii=False) + "\n")
        return version

    def generate_synthetic(self, agent_spec: dict, n: int = 50) -> list[d
ict]:
        """Generar casos sintéticos con LLM basados en la spec del
agente."""
        prompt = f"""
        Given this agent spec: {json.dumps(agent_spec)}
        Generate {n} test cases in JSONL format.
        Include: 60% golden, 20% edge, 20% adversarial.
        Each case: {{"id","input","expected_tool_calls","expected_escalat
ion","output_contains","output_not_contains","tags"}}
        """
        response = llm.complete(prompt)
        return [json.loads(line) for line in
response.text.strip().split("\n")]

    def from_production(self, executions: list[dict], sample_pct: float
= 0.1) -> list[dict]:
        """Crear dataset desde ejecuciones de producción (shadow
eval)."""
        import random
        sampled = random.sample(executions, int(len(executions) *
sample_pct))
        return [{"id": f"prod-{e['id']}", "input": e["input"], "actual_ou
tput": e["output"]} for e in sampled]
```

6.3 Evaluadores

LLM-as-Judge a Escala

```
import asyncio

class ScalableJudge:
    def __init__(self, judge_model: str = "claude-sonnet-4-6",
concurrency: int = 10):
        self.model = judge_model
        self.semaphore = asyncio.Semaphore(concurrency)

    async def evaluate_batch(self, cases: list[dict]) -> list[dict]:
        tasks = [self._evaluate_one(case) for case in cases]
        return await asyncio.gather(*tasks)

    async def _evaluate_one(self, case: dict) -> dict:
        async with self.semaphore:
            result = await llm.complete(
                model=self.model,
                messages=[{"role": "user", "content": self._build_prompt(
case)}],
                temperature=0.0,
            )
            scores = json.loads(result.text)
            return {**case, "judge_scores": scores}
```

Arena Pattern (Pairwise Comparison)

```
class AgentArena:
    """Comparar 2 versiones del agente head-to-head."""

    async def compare(self, agent_a, agent_b, test_cases: list) -> dict:
        wins = {"a": 0, "b": 0, "tie": 0}

        for case in test_cases:
            resp_a = await agent_a.run(case["input"])
            resp_b = await agent_b.run(case["input"])

            winner = await self._judge_pair(case["input"], resp_a,
resp_b)
            wins[winner] += 1

        return {
            "agent_a_wins": wins["a"],
            "agent_b_wins": wins["b"],
            "ties": wins["tie"],
            "recommendation": "a" if wins["a"] > wins["b"] else "b",
        }
```

6.4 Pipeline de Evaluación Continua

CI/CD Integration

```yaml
# .github/workflows/agent-eval.yml
name: Agent Continuous Evaluation

on:
  schedule:
    - cron: '0 2 * * *'  # Nightly at 2am
  push:
    paths: ['agents/**', 'prompts/**', 'specs/**']

jobs:
  eval:
    runs-on: ubuntu-latest
    steps:
      - uses: actions/checkout@v4

      - name: Run golden evals
        run: python eval/run.py --dataset golden --threshold 85

      - name: Run adversarial evals
        run: python eval/run.py --dataset adversarial --threshold 95

      - name: Run regression
        run: python eval/run.py --dataset regression --threshold 100

      - name: Shadow eval (production sample)
        run: python eval/shadow.py --sample-pct 5 --min-score 4.0

      - name: Cost check
        run: python eval/cost.py --max-cost-per-query 0.05

      - name: Report
        if: always()
        run: python eval/report.py --output github-comment
```

Automated Rollback

```
class AutoRollback:
    async def check_and_rollback(self, agent_id: str, window_minutes:
int = 30):
        metrics = await get_recent_metrics(agent_id, window_minutes)

        if metrics.error_rate > 0.10:
            await rollback(agent_id, reason="error_rate > 10%")
        elif metrics.avg_score < 3.5:
            await rollback(agent_id, reason="quality < 3.5")
        elif metrics.data_leakage > 0:
            await rollback(agent_id, reason="data leakage detected")
            await disable(agent_id)  # Desactivar inmediatamente
```

6.5 Ejercicios

Ejercicio 1: Synthetic Dataset

Genera un dataset sintético de 50 casos para un agente de soporte usando LLM. Incluye golden, edge y adversarial.

Ejercicio 2: Arena

Implementa una Arena que compare 2 versiones de un agente. Ejecuta 20 queries y determina el ganador con LLM-as-judge.

Ejercicio 3: Shadow Eval Pipeline

Implementa shadow evaluation que muestrea 10% de queries de producción, las evalúa con LLM-as-judge, y alerta si el score baja de 4.0.

Ejercicio 4: Nightly Eval

Configura un GitHub Action que ejecute evals nightly con thresholds y reporte automático.