IACADEMY

DE FUNDAMENTOS A ARQUITECTURA DE IA

MÓDULO 07

Observabilidad y calidad

Aplicación

iacedemy.com — 2026

MÓDULO 07

Observabilidad y calidad

Nivel: Aplicación

Autor: Ricardo Gutierrez

Publicación: Mayo 2026

Plataforma: iacedemy.com

Logging de operaciones IA

Todos hablan de usar IA. Nadie habla de medir si funciona. Si no mides, no sabes si tu prompt mejoro o empeoro con el ultimo cambio. No sabes si tu agente tarda 3 segundos o 30. No sabes cuanto dinero estas gastando por operacion. Estas ciego.

Los 10 campos obligatorios

Cada vez que tu IA hace algo, necesitas registrar estos 10 campos:

```
 log_entry = {
    "timestamp": "2026-05-12T10:30:00Z",     # 1. Cuando ocurrio
    "operation_id": "op_abc123",               # 2. ID unico para rastrear
    "agent": "content-generator",              # 3. Quien lo hizo
    "action": "generate_blog_post",            # 4. Que accion ejecuto
    "model": "claude-sonnet-4-5-20250514",     # 5. Que modelo uso
    "input_tokens": 1250,                      # 6. Tokens de entrada
    "output_tokens": 3400,                     # 7. Tokens de salida
    "latency_ms": 4200,                        # 8. Cuanto tardo en ms
    "status": "success",                       # 9. Funciono o fallo
    "cost_usd": 0.018                          # 10. Cuanto costo
}
```

El timestamp te permite hacer graficas temporales. El operation_id te permite rastrear una operacion especifica cuando algo falla. Input y output tokens son tu principal palanca de coste. Un prompt inflado con 5000 tokens de contexto innecesario te esta costando dinero cada vez que se ejecuta.

Implementacion

```
 import json
import time
from datetime import datetime, timezone

def log_ai_operation(agent: str, action: str, model: str,
                     input_tokens: int, output_tokens: int,
                     latency_ms: int, status: str, cost_usd: float):
    entry = {
        "timestamp": datetime.now(timezone.utc).isoformat(),
```

```
        "operation_id": f"op_{int(time.time()*1000)}",
        "agent": agent,
        "action": action,
        "model": model,
        "input_tokens": input_tokens,
        "output_tokens": output_tokens,
        "latency_ms": latency_ms,
        "status": status,
        "cost_usd": cost_usd
    }
    with open("ai_operations.jsonl", "a") as f:
        f.write(json.dumps(entry) + "\n")
    return entry
```

JSONL porque cada linea es un JSON valido independiente. Puedes hacer append sin romper el formato. Puedes leer linea a linea sin cargar todo en memoria. Y Claude lo parsea perfecto.

> **Consejo practico**
>
> Empieza a logear desde el dia uno. Incluso si no miras los datos durante un mes. Cuando los necesites (y los vas a necesitar), estaran ahi. Reconstruir datos historicos es imposible.

Eval datasets

Un eval dataset es el examen que le pones a tu IA. Sin eval, no sabes si tu prompt es bueno o malo. Solo tienes una sensacion. "Me parece que funciona bien." Eso no es ingenieria. Eso es esperanza.

Estructura de un eval

```
 {"input": "Clasifica este email: 'Tu factura esta lista'", "expected":
"transaccional", "category": "clasificacion"}
{"input": "Clasifica este email: 'Gana un iPhone gratis'", "expected":
"spam", "category": "clasificacion"}
```

```
{"input": "Resume este parrafo en 1 frase: [texto largo]", "expected": "El
PIB crecio un 2.3% en Q1 2026", "category": "resumen"}
```

Tres campos minimos: input (que le das a la IA), expected (que deberia responder), category (para agrupar metricas).

Como crear un eval dataset

1. **Recopila 50 ejemplos reales.** No inventados. De tu produccion real. Los ejemplos inventados no reflejan la realidad.
2. **Anota la respuesta correcta manualmente.** Tu, humano, decides que es correcto. Tedioso pero indispensable.
3. **Categoriza.** Agrupa por tipo o dificultad para detectar donde falla.
4. **Diversifica.** Incluye edge cases, errores comunes, inputs malformados.
5. **Mide.** Ejecuta tu prompt contra el dataset y calcula accuracy. Ese numero es tu baseline.

Runner de evaluacion

```
 import json

def run_eval(eval_path: str, classify_fn) -> dict:
    correct = 0
    total = 0
    results_by_category = {}

    with open(eval_path) as f:
        for line in f:
            item = json.loads(line)
            prediction = classify_fn(item["input"])
            is_correct = prediction == item["expected"]
            correct += is_correct
            total += 1

            cat = item.get("category", "unknown")
            if cat not in results_by_category:
                results_by_category[cat] = {"correct": 0, "total": 0}
            results_by_category[cat]["correct"] += is_correct
            results_by_category[cat]["total"] += 1

    return {
```

```
        "accuracy": correct / total,
        "total": total,
        "by_category": results_by_category
    }
```

Resultado: accuracy 87%, faciles 95%, dificiles 72%. Ahora sabes EXACTAMENTE donde falla tu prompt. No vibes. Datos.

Prompt versioning

Los prompts son codigo. Y el codigo se versiona. Si cambias un prompt y no guardas la version anterior, no puedes hacer rollback. No puedes comparar. No puedes saber que version funcionaba mejor.

Estructura de carpetas

```
docs/prompts/
 classifier_support_v1.0.txt
 classifier_support_v1.1.txt
 classifier_support_v2.0.txt
 summarizer_blog_v1.0.txt
 content_generator_linkedin_v1.0.txt
```

Naming convention: `{agente}_{tarea}_v{major}.{minor}.txt`

- **Major:** Cambio estructural. Nueva seccion, nuevo formato de output.
- **Minor:** Ajuste de redaccion, ejemplo anadido, clarificacion.

Metadata del prompt

```
 # Prompt: classifier_support_v2.0
# Autor: Ricardo Gutierrez
# Fecha: 2026-05-12
# Modelo target: claude-sonnet-4-5-20250514
# Eval accuracy: 91% (50 samples)
# Cambios: Anadida categoria "complaint", instruccion de uncertainty
# Dependencias: Ninguna
```

Esta metadata es oro. Cuando dentro de 3 meses alguien pregunta "por que usamos este prompt?", la respuesta esta ahi. Sin metadata, es un archivo misterioso que nadie se atreve a tocar.

Medir el impacto

```
 Prompt v1.0: accuracy 82% (50 evals)
Prompt v1.1: accuracy 87% (50 evals) — +5 puntos
Prompt v2.0: accuracy 91% (50 evals) — +4 puntos
```

Sin versionado, nunca sabrias que v1.1 mejoro 5 puntos al anadir dos categorias y la instruccion de uncertainty.

Dashboard de operaciones IA

Seis metricas que necesitas ver de un vistazo:

1. **Operaciones/dia:** Cuantas llamadas a IA haces. Si sube mucho, alguien esta generando en exceso. Si baja a cero, algo esta roto.
2. **Latencia p50/p95:** Tu IA responde en 2 segundos normalmente (p50), pero el 5% de las veces tarda 15 (p95). Ese p95 es lo que tus usuarios sienten.
3. **Coste acumulado:** Cuanto llevas gastado este mes. Si tu presupuesto es 100 dolares y llevas 80 el dia 15, problema.
4. **Accuracy por agente:** Cada agente con su score del eval. Si uno baja, investigas.
5. **Errores/dia:** Tasa de fallos. Deberia estar bajo 5%.
6. **Tokens/operacion:** Media de tokens por llamada. Detecta prompts inflados.

Dashboard basico con Python

```
 import json
from collections import defaultdict

metrics = defaultdict(list)
with open("ai_operations.jsonl") as f:
    for line in f:
        entry = json.loads(line)
        metrics["latency"].append(entry["latency_ms"])
        metrics["cost"].append(entry["cost_usd"])
```

```
        metrics["status"].append(entry["status"])

total = len(metrics["status"])
errors = sum(1 for s in metrics["status"] if s != "success")
latencies = sorted(metrics["latency"])

print(f"Total operaciones: {total}")
print(f"Tasa de error: {errors/total*100:.1f}%")
print(f"Latencia p50: {latencies[total//2]}ms")
print(f"Latencia p95: {latencies[int(total*0.95)]}ms")
print(f"Coste total: ${sum(metrics['cost']):.2f}")
```

Para produccion: Grafana + OpenTelemetry. Pero para empezar, un script de 20 lineas te da el 80% del valor. No dejes que la perfeccion te impida empezar.

Evals en CI

```
 name: AI Eval Gate
on:
  push:
    paths: ['docs/prompts/**']

jobs:
  eval:
    runs-on: ubuntu-latest
    steps:
      - uses: actions/checkout@v4
      - name: Run evals
        run: python scripts/run_evals.py
      - name: Check threshold
        run: |
          ACCURACY=$(cat eval_results.json | jq '.accuracy')
          if (( $(echo "$ACCURACY < 0.85" | bc -l) )); then
            echo "Eval failed: accuracy $ACCURACY < 0.85"
            exit 1
          fi
```

Cada vez que alguien cambia un prompt, CI ejecuta los evals. Si la accuracy baja del 85%, el PR no se puede mergear. Sin excusas. Los datos dicen que no.

Cuando retrain vs reprompt

```
Accuracy bajo del threshold?
 SI → Errores de comprension del formato?
   SI → REPROMPT: anade ejemplos, clarifica instrucciones
   NO → Errores de conocimiento del dominio?
     SI → Tienes 1000+ ejemplos anotados?
       SI → FINE-TUNE
       NO → RAG o mas few-shot examples
     NO → CAMBIAR MODELO o REVISAR EVAL DATASET
```

En el 90% de los casos, repromptear resuelve el problema. Fine-tune es el ultimo recurso, no el primero.

Ejercicio practico

Ejercicio M07: Implementa observabilidad

1. Implementa la funcion `log_ai_operation` en tu proyecto.
2. Crea un eval dataset de 50 items para una tarea que tu IA haga (clasificacion, resumen, etc.).
3. Ejecuta el eval y obtiene tu baseline de accuracy.
4. Versiona tu prompt actual como v1.0 con metadata completa.
5. Haz un cambio al prompt, guardalo como v1.1, y vuelve a ejecutar el eval.
6. Compara v1.0 vs v1.1 con datos.
7. Crea el script de dashboard basico y ejecutalo con tus logs.

Bonus: Configura un GitHub Action que ejecute evals automaticamente cuando cambies un prompt.

Conclusiones clave

Key takeaways del M07

1. 10 campos obligatorios en cada log de operacion IA. Empieza desde el dia uno.
2. Eval datasets miden calidad con datos, no con vibes. 50 ejemplos reales, anotados manualmente.
3. Prompts versionados con naming convention y metadata. Cada cambio es rastreable y comparable.
4. Dashboard de 6 metricas: operaciones, latencia, coste, accuracy, errores, tokens.
5. Evals en CI impiden mergear prompts que empeoran la calidad. Automatico.
6. El 90% de los problemas se resuelven repromoteando. Fine-tune es el ultimo recurso.

Ahora sabes medir

En el M08 entramos en desarrollo: TDD asistido por IA, CLAUDE.md enterprise, agentes arquitecto y debugger, y code review automatizado en CI.

Ir al Modulo 08

IACADEMY

iacedemy.com

De fundamentos a arquitectura de IA.
12 módulos prácticos. 24 recursos descargables.
Quizzes con certificado. Vídeos profesionales.

Empieza gratis en iacedemy.com/free