Estrategias de Resiliencia: Fallback Automático vs Proxy LiteLLM en IA

Estantería: Art de Coté

Subtema: Arquitectura

mercedev.es — 2026-06-18 | Epic 9 - Fase QA

Estrategias de Resiliencia en Ecosistemas de IA

En el despliegue de Agentes Autónomos y asistentes como Merci, garantizar la disponibilidad cuando las cuotas de la API (como los tokens de Google Gemini) se agotan es una prioridad arquitectónica. Actualmente conviven dos filosofías de contingencia en nuestro ecosistema.

El Proxy de Capa de Red (LiteLLM)

El sistema clásico implementado en el alias de terminal `mercedev` se basa en un paradigma de **interceptación de red**:

1. **Transparencia de la Aplicación**: El IDE (o el script cliente) no sabe que se están usando distintos modelos. Está configurado para apuntar a un *endpoint* de OpenAI unificado (`http://localhost:4000`).
2. **Lógica Centralizada**: Es el servidor LiteLLM el que mantiene la lógica. Cuando recibe un HTTP 429 (Too Many Requests) o un 400 por agotamiento de tokens en Gemini, el propio proxy enruta la petición automáticamente hacia Anthropic (Claude) o Groq (Llama3).
3. **Ventaja**: Es universal. Cualquier script antiguo o nuevo que se conecte a `localhost:4000` heredará esta resiliencia sin modificar una sola línea de su código fuente.

Fallback Nativo a Nivel de SDK (Google Antigravity)

La nueva implementación ("la nueva forma de manejar las IAs") reside en la **Capa de Aplicación** mediante las capacidades avanzadas del SDK de Google Antigravity:

1. **Consciencia del Agente**: En lugar de ocultarle el error al agente, el SDK captura la excepción de límite de tokens directamente.
2. **Enrutamiento Determinista**: El código del agente posee una lista ordenada de modelos alternativos. Si el modelo primario (`gemini-1.5-flash`) falla, el agente sabe instanciar una conexión con el secundario y reintentar la inferencia por su cuenta.
3. **Observabilidad Nativa**: ¿Cómo sabes que te ha derivado? Al ser una decisión de la aplicación, el SDK de Antigravity (y por extensión, el script de Python) emitirá un log visible en la terminal (ej. `[INFO] Fallback triggered: switching to gemini-1.5-pro`). Además, la telemetría del propio agente reflejará el cambio de contexto, permitiendo auditorías precisas sobre qué modelo generó cada fragmento de código.

Veredicto Arquitectónico

Mientras que **LiteLLM** actúa como un escudo protector ciego (excelente para scripts legados y herramientas estándar de terceros), el **Fallback del SDK** permite al Agente Autónomo mantener la consciencia situacional, adaptar su *system prompt* a las capacidades del modelo secundario, y registrar detalladamente el evento de contingencia en la bitácora del proyecto.