

MSAP.ai

msap.ai

# LiteLLM 제품 소개

LLM 호출 경로를 하나로 모으는 Gateway

PROJECT CODE

**LiteLLM-YT-14**

DATE

**2026.10.08**

MSAP.ai

# Agenda

01

## 흘어진 LLM 호출 경로의 관리 문제

에이전트 확산과 키·비용·모델 교체의 관리 사각지대

02

## LLM 운영 과제와 벤더 AI 통합의 한계

호출 경로의 제약과 게이트웨이가 필요한 이유

03

## LiteLLM의 개념과 OpenAI 호환 API

프로바이더 통합 방식, SDK와 Proxy, 최근 변화

04

## LiteLLM Proxy의 핵심 기능

별칭, Virtual Key, 예산, 라우팅, 가드레일, 캐싱

05

## SDK·Proxy 선택과 설치 구성

두 형태의 운영 차이, 최소 설치, 클라이언트 연결

06

## LLM Gateway 요건 평가와 대안 비교

요건 8개, Enterprise 버전 경계, 오픈소스·국내 대안

07

## 로컬 LLM과 클라우드의 혼합 구성

기밀 데이터 대응을 위한 로컬 모델 연결 방식

08

## LiteLLM 도입 사례 네 건

벤고시닷컴, RAKUS, IVRy, primeNumber의 구성과 한계

09

## 사례 구성 진단: Docker 기반 Proxy

컨테이너 한 대 구성의 별칭·폴백·키 관리 현황

10

## 사례 구성 진단: 에이전트와 요청 흐름

에이전트 배치, 사용량 분포, 메시지 한 건의 요청 경로

11

## LiteLLM Proxy 모니터링

Prometheus 지표, TTFT, 로깅 연동, 사례 구성의 사각지대

12

## LLM Gateway 보안과 공급망 사고

2026년 침해 사고, 취약점, 점검 항목, 에이전트 위험

13

## 쿠버네티스 배포와 이중화

운영 배포 구성, Helm, Redis, Postgres 전환 절차

14

## 도입 판단 기준과 실행 순서

LiteLLM Proxy의 전제 조건, 결정 항목, 조치 순서

## SECTION 01

# 흩어진 LLM 호출 경로의 관리 문제

에이전트 확산과 키·비용·모델 교체의 관리 사각지대



## KEY INSIGHT

프로그램에 내장된 에이전트는 사람 없이 LLM을 반복 호출함  
— 챗봇이나 코딩 어시스턴트와는 설계 기준이 다름

2017

2022

2024

2025

### Transformer Google Research

'Attention Is All You Need' 논문으로  
LLM의 기반 구조 제시

### ChatGPT OpenAI

출시 2개월 만에 월간 사용자 1억 명<sup>1</sup>, LLM  
대중화

### RAG 확산

사내 문서·DB 검색을 결합해 답변 근거를 확보

### AI 에이전트 2025년의 키워드

모델이 도구를 직접 골라 다단계 작업을 수행

## 에이전트 유형별 LLM 호출 방식

### CHATBOT

1회

질문 1건당 LLM 호출<sup>5</sup>

예시  
ChatGPT, Claude.ai, Gemini

호출 주체  
사람이 질문할 때마다 한 번 응답

### CODING ASSISTANT

약 40배

버그 1건 수정 토큰, 같은 질문 채팅 대비<sup>4</sup>

예시  
GitHub Copilot, Cursor, Claude Code

호출 주체  
개발자 지시 후 읽기·실행·수정 반복

### AI PROGRAM

10-20회

Gateway 대상

작업 1건당 LLM 호출, 사람 개입 없음<sup>5</sup>

예시  
고객지원·운영 자동화, 사내 업무 봇

호출 주체  
프로그램이 이벤트·스케줄로 자동 호출

## 에이전트 확산 전망

33%

2028년 에이전트 AI를 포함할 기업용 소프트웨어 비중<sup>2</sup>

2024 1% 미만

2028E 33%

5-30배

에이전트 작업 1건의 토큰 사용량, 챗봇 대비<sup>3</sup>

타임라인은 Guillaume Laforge 발표의 네 시기 기준. 호출·토큰 수는 작업 유형에 따라 편차가 큼

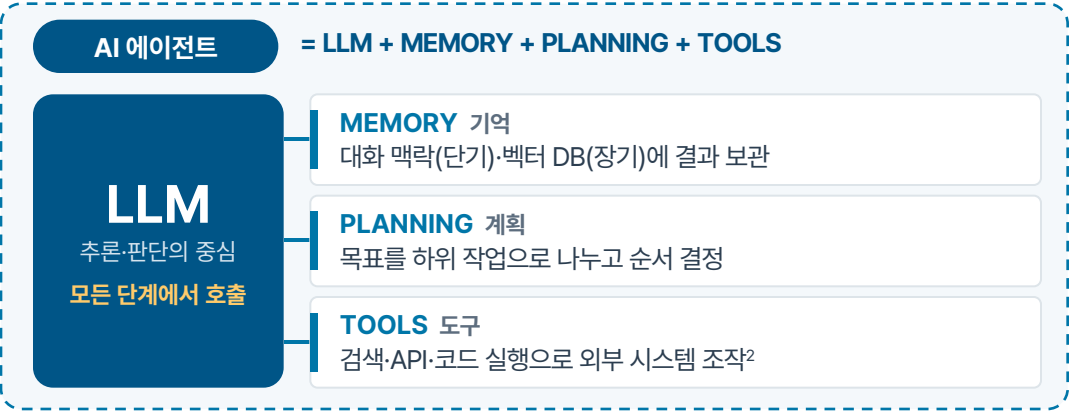
<sup>1</sup> Reuters·UBS (2023.2) <sup>2</sup> Gartner (2024.10) <sup>3</sup> Gartner (2026.3) <sup>4</sup> FavTray (2026): 버그 수정 125K-290K 토큰 vs 채팅 3K-8K <sup>5</sup> NeuralTrust, Cockroach Labs (2026)



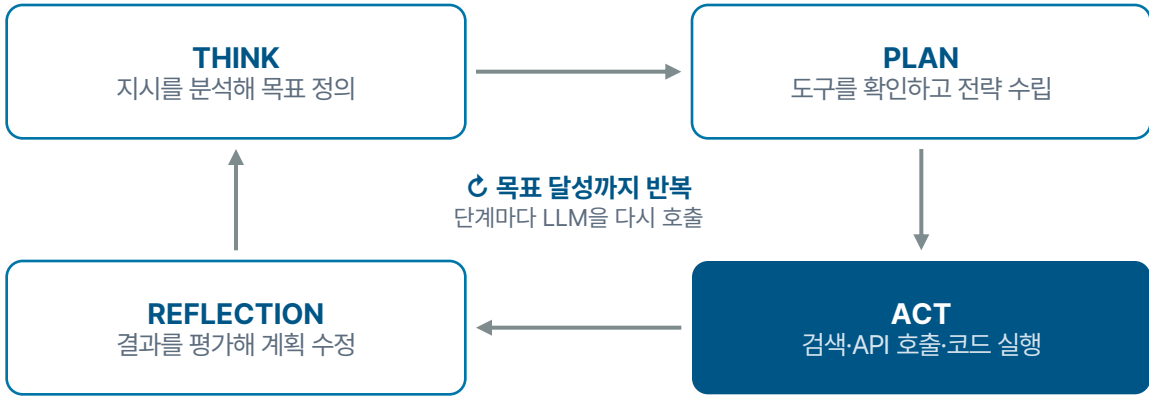
KEY INSIGHT

에이전트는 모델에 메모리·계획·도구를 더한 것임  
— 목적을 받아 도구를 골라 행동함

1 AI 에이전트의 구성 요소<sup>1</sup> LLM이 두뇌, 나머지가 손발과 기억



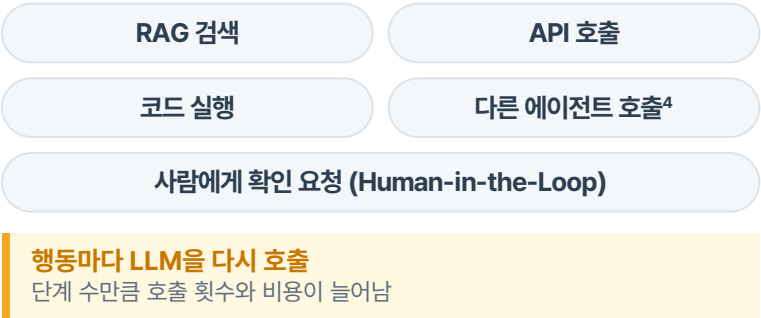
2 동작 루프<sup>3</sup> 목표를 달성할 때까지 생각과 행동을 반복



3 대화형 AI와의 차이

|        | 대화형 AI     | AI 에이전트       |
|--------|------------|---------------|
| 입력     | 질문         | 목적            |
| 처리     | 학습 지식으로 응답 | 정보·도구를 스스로 선택 |
| 출력     | 답변 텍스트     | 행동한 결과        |
| LLM 호출 | 1회         | 단계마다 반복       |

4 ACT 단계의 행동



5 운영 업무에서의 역할

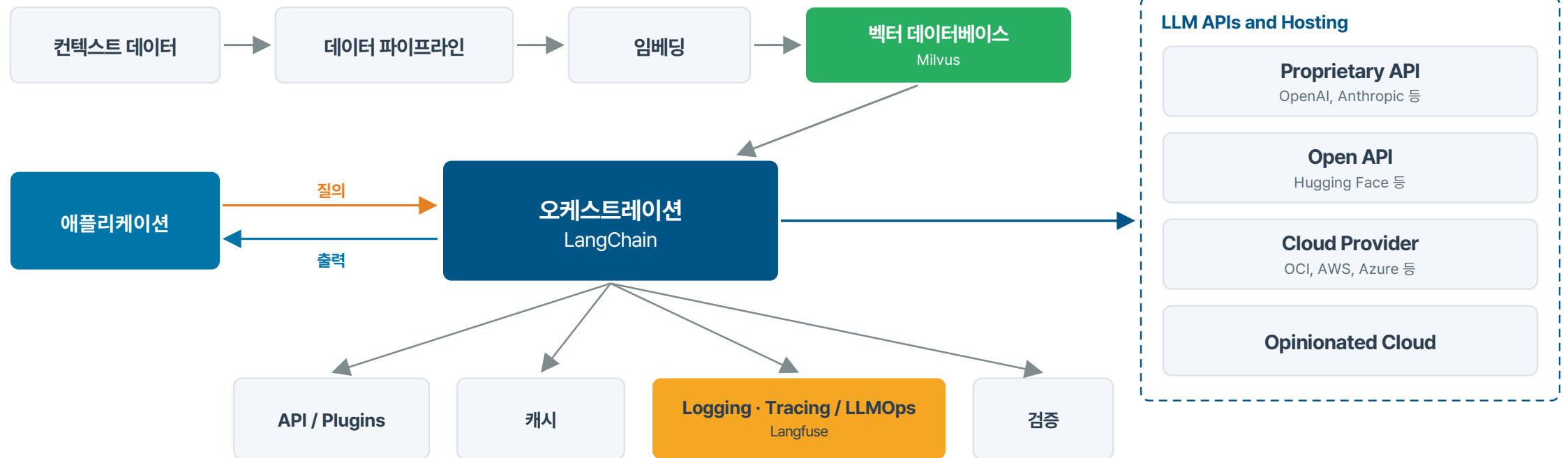
- 여러 데이터를 묶은 조사  
상태·이벤트·로그를 함께 확인
- 모호한 요청의 해석  
원인과 다음 조치를 도출
- 조사와 보고의 통일  
형식을 통일해 특정한 의존을 방지

<sup>1</sup> Lilian Weng, LLM Powered Autonomous Agents (2023) <sup>2</sup> Anthropic, Model Context Protocol (2024.11) <sup>3</sup> Yao et al., ReAct (2022) <sup>4</sup> Google, Agent2Agent 프로토콜 (2025.4)



## KEY INSIGHT

이 구성도에는 게이트웨이가 없음  
— 오케스트레이션이 여러 LLM API 를 직접 호출하는 형태



색이 채워진 상자: OCHaCafe 발표가 다루는 도구 (LangChain, Milvus, Langfuse)

이 구성도에 LLM 게이트웨이 상자는 없음



# LLM 을 호출하는 클라이언트의 세 가지 형태

01 · 흩어진 LLM 호출 경로의 관리 문제

## KEY INSIGHT

LLM을 부르는 클라이언트는 세 가지 형태이고, 프록시를 우회하는 3건은 모두 에이전트 밖에 있음  
에이전트 프레임워크 설정만 점검하면 보조 스크립트와 독립 서비스의 호출을 놓침

### 1 형태별 호출 방식과 프록시 경유 여부 사례 구성 조사 기준

1

#### 에이전트 루프

프레임워크가 대화와 도구 호출을 반복하며 모델 호출

호출 방식 루프 안에서 반복 호출

카·설정 위치 프레임워크 설정 파일

호출 횟수 미리 정하기 어려움

관리 담당 에이전트 운영자

#### ● LiteLLM 프록시 경유 2곳

업무 에이전트 스택 · 콘텐츠 공장 스택

#### 프록시 우회 0건

모든 호출이 프록시를 거침

2

#### 에이전트 밖 보조 스크립트

도구나 예약 작업이 실행하는 프로그램

호출 방식 작업 1건당 정해진 호출

카·설정 위치 스크립트 코드·환경변수

호출 횟수 작성자가 정한 만큼

관리 담당 스크립트 작성자

#### ● LiteLLM 프록시 경유 5곳

PDF 읽기 · 명함 인식 · 이력서 평가 · 콘텐츠 요약 · 영상 첫 댓글

#### 프록시 우회 2건

- AI 검색 인용 주간 점검 스크립트
- 위키 정리 스크립트

3

#### 독립 서비스

에이전트와 별개로 운영하는 검색·제작·관리 도구

호출 방식 임베딩 + 대화 호출

카·설정 위치 서비스 자체 설정

호출 횟수 서비스 요청량에 따라 변동

관리 담당 서비스 담당자

#### ● LiteLLM 프록시 경유 3곳

에이전트 콘솔 · 슬라이드 검색 서비스 · 영상 제작 도구 일부

#### 프록시 우회 1건

- 이미지 생성 백엔드

**점검 범위** 프레임워크 설정 파일만이 아니라 **스크립트 코드·환경변수**와 **서비스 자체 설정**까지 확인해야 우회 경로 3건이 드러남  
우회 3건 = 보조 스크립트 2 + 독립 서비스 1 + 에이전트 루프 0



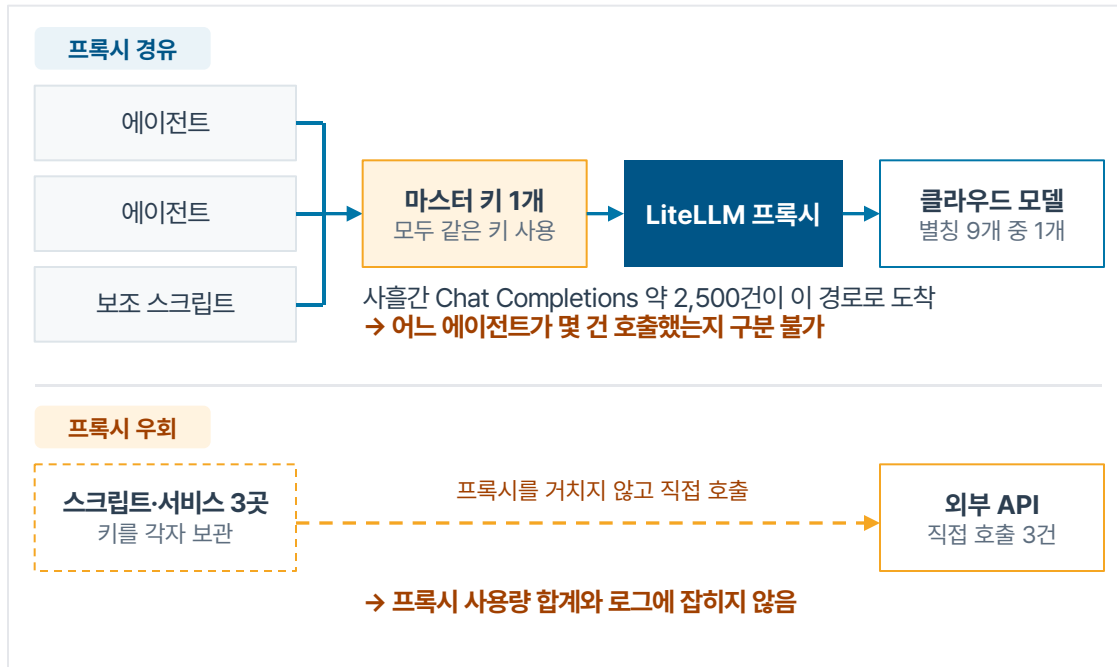
# 호출 경로가 흩어지면 키와 비용부터 관리가 안 됨

01 · 흩어진 LLM 호출 경로의 관리 문제

## KEY INSIGHT

클라이언트가 마스터 키 하나를 같이 쓰거나 프록시를 우회해 호출하므로, 누가 얼마를 호출했는지 알 수 없음  
호출자를 구분하지 못하니 키 회수, 비용 집계, 모델 교체를 한 곳에서 처리할 수 없음

### 1 지금 호출이 흐르는 모습 사례 조사, 사흘간 프록시 로그 기준



#### 조사로 드러난 점

- 외부로 나가는 경로 4곳 중 프록시 설정 파일로 확인되는 것은 1곳
- 컨테이너 로그에 요청별 모델 별칭이 남지 않아 사후 추적도 불가

### 2 그래서 관리할 수 없는 세 가지 증상과 그 상태에서 막히는 일



#### 해결 방향: 클라이언트마다 키를 나누고 모든 호출을 Gateway로 모음

- 키별로 사용 중지·교체 가능
- 키별 비용 집계와 예산 한도 설정 (LiteLLM Virtual Key 기능)
- 모델 별칭을 Gateway 설정 한 곳에서 변경



KEY INSIGHT

LLM을 호출하는 곳마다 다섯 항목을 조사하면 Gateway 도입 계획이 나옴  
— 항목마다 Virtual Key 단위, 모델 별칭, 이전 순서, 예산 한도 같은 결정 하나로 이어짐

1 조사 항목과 도입 때 정하는 것

| 조사 항목         | 적는 내용 · 작성 예시                                                          | 도입 때 정하는 것                   |
|---------------|------------------------------------------------------------------------|------------------------------|
| 클라이언트         | 역할 이름과 호출 형태 (에이전트 루프·보조 스크립트·독립 서비스)<br>예) 장애 분석 에이전트 · 에이전트 루프       | Virtual Key를 발급할 단위          |
| 사용 키          | 키 종류 (마스터·Virtual·프로바이더)와 다른 클라이언트와의 공유 여부<br>예) 프로바이더 키 · 스크립트 2개와 공유 | 분리할 키, 유출 시 교체 범위            |
| 대상 모델         | 모델 별칭, 대화·임베딩 구분, 용도 한 줄<br>예) gpt-4o · 대화 · 장애 내용 요약                  | Gateway에 등록할 별칭, 키별 모델 접근 범위 |
| Gateway 경유 여부 | 경유·비경유, 비경유면 직접 호출하는 프로바이더<br>예) 비경유 · OpenAI API 직접 호출                | Gateway 경유로 옮길 경로            |
| 호출 빈도         | 하루·일주일 단위 대략적 요청 수, 예약 실행 여부<br>예) 하루 약 500건 · 매시 예약 실행                | 이전 순서, 예산 한도 초깃값             |

2 작성 기준과 결과 읽는 법

|                                                                                                            |                                                                                                       |                                                                                                             |
|------------------------------------------------------------------------------------------------------------|-------------------------------------------------------------------------------------------------------|-------------------------------------------------------------------------------------------------------------|
| <p><b>조사 범위</b></p> <p><b>포함</b> 임베딩 호출 (대화 호출과 같은 프록시·키를 거침)</p> <p><b>제외</b> LLM을 쓰지 않는 예약 작업, 음성 인식</p> | <p><b>작성 요령</b></p> <p><b>키</b> 값은 적지 않고 종류와 보관 위치만 기재</p> <p><b>빈도</b> 예약 주기·세션 수처럼 이미 아는 값으로 기재</p> | <p><b>결과 읽는 법</b></p> <p><b>신호</b> 보조 스크립트·서비스 행이 에이전트 행보다 많음</p> <p><b>판단</b> 호출 경로가 이미 흠어진 상태, 통합이 먼저</p> |
|------------------------------------------------------------------------------------------------------------|-------------------------------------------------------------------------------------------------------|-------------------------------------------------------------------------------------------------------------|

## SECTION 02

# LLM 운영 과제와 벤더 AI 통합의 한계

호출 경로의 제약과 게이트웨이가 필요한 이유



## KEY INSIGHT

애플리케이션마다 따로 대응하면 같은 로직을 여러 번 만들게 됨  
— 그래서 추상화 API, 모니터링, 부하 분산 도구가 나옴



### 비용

토큰 수에 비례한 과금. 서비스 규모가 커지면 무시하기 어려운 수준

필요한 대응

**사용량 모니터링과 비용 상한**



### 토큰량 예측

토큰은 글자 수와 달라 호출 전에 정확한 양을 알기 어려움

필요한 대응

**실측 기록 기반 추정**



### 속도

짧은 글은 1초~수 초, 긴 글은 5~10초 이상. 동기 호출이면 대기 발생

필요한 대응

**비동기 처리와 재시도**



### Rate Limit

RPM(분당 요청 수)과 TPM(분당 토큰 수) 상한을 넘으면 요청 거부

필요한 대응

**여러 배포로 부하 분산**



### 가용성

프로바이더의 정지와 지연이 서비스 기능에 그대로 전달

필요한 대응

**다른 프로바이더로 폴백**

## 이 제약에 대응하는 도구 세 종류

### 추상화 API

여러 프로바이더를 한 가지 형식으로 호출  
예: LiteLLM, LangChain

### 모니터링

토큰과 비용, 지연을 요청 단위로 기록  
예: Langfuse, LangSmith

### 부하 분산과 장애 대책

한도 초과와 프로바이더 장애 때 다른 경로로 전환  
예: LiteLLM



## KEY INSIGHT

개인 계정 사용은 금지만으로 사라지지 않음

— 업무 효율 때문에 계속 쓰므로 대신 쓸 공식 서비스가 필요함

### 추적되지 않는 AI 사용 (Shadow IT)

- **전송 내역 추적 불가**  
무엇을 어느 서비스에 보냈는지 조직이 알 수 없음
- **전송 여부 관리 불가**  
사내 정보가 나갔는지조차 확인할 수단이 없음
- **사고 대응 불가**  
사고가 나도 영향 범위를 특정할 수 없음

### 데이터 이용 정책의 사각지대

- **서비스마다 다른 정책**  
입력 데이터의 학습 이용 여부가 회사마다 다름
- **약관 변경 추적이 어려움**  
정책이 바뀌어도 조직이 따라가기 어려움
- **개인에게 맡겨진 설정**  
학습 이용 거부 설정 여부가 사용자마다 다름

### 그런데도 개인 계정을 쓰는 이유

업무 효율이 확실히 오르기 때문. 규모가 작은 조직일수록 보안 담당자가 직접 활용하면서 동시에 안전한 사용을 책임져야 함

primeNumber 의 진단: 생성형 AI 를 업무에 쓰지 않는 조직이 오히려 드문 상황



## KEY INSIGHT

클라우드 벤더 AI로 모으면 데이터 학습 문제는 풀리지만, 누가 · 어떤 모델을 · 얼마나 쓰는지는 여전히 조직이 관리해야 함  
벤더가 약관으로 보장하는 것은 데이터 처리 방식이며, 사용자 단위의 비용 · 권한 · 모델 관리는 별도로 구성해야 함

### 1 통합 단계별로 해결되는 것과 남는 것 개인 계정 → 클라우드 벤더 AI → 남는 과제

#### 개인 계정으로 사용

도입 전

##### ✗ 추적할 수 없는 전송 (Shadow IT)

무엇을 어느 서비스에 보냈는지 알 수 없어 사고가 나도 영향 범위를 특정할 수 없음

##### ✗ 개인에게 맡겨진 데이터 정책

학습 이용 거부 설정이 사람마다 다름

#### 클라우드 벤더 AI로 통합

1차 통합

##### ✓ 입력 · 생성 데이터를 모델 학습에 쓰지 않음

Amazon Bedrock, Azure OpenAI Service, Google Vertex AI가 약관에 명시

##### ✓ 허용 서비스를 정하면 사용자도 그 서비스로 옮겨감

Shadow IT와 데이터 정책 문제는 상당 부분 해소

#### 조직이 직접 관리할 영역

남는 과제

① 비용 · 모니터링

② 권한 관리

③ 이용 모델 제한

④ 여러 프로바이더

↓ 아래에서 하나씩 설명

### 2 그래도 남는 과제 네 가지 벤더 AI에서 바로 해결되지 않는 이유와 필요한 기능

#### 1 비용 관리와 모니터링

사용자별 사용량을 따로 추적하기 어렵고, 악의 없는 과다 사용만으로도 요금이 급증함 (EDoS, 경제적 서비스 거부)

필요 ▶ 사용자별 사용량 집계와 예산 한도

#### 2 권한 관리

모델을 쓰려면 사용자 전원에게 클라우드 IAM 권한을 발급해야 하며, 비개발자가 다루기 어려움

필요 ▶ IAM 없이 쓰는 사용자별 키 발급

#### 3 이용 가능 모델 제한

모든 사용자에게 모든 모델을 열 필요는 없지만, 사람이나 팀 단위로 제한할 간단한 방법이 없음

필요 ▶ 사용자 · 팀별 허용 모델 목록

#### 4 여러 프로바이더 대응

프로바이더마다 API 형식이 달라 클라우드, SaaS, 로컬 모델을 나눠 쓰면 호출 코드가 늘어남

필요 ▶ 하나의 API 형식으로 통일

### 네 과제의 공통점: 클라우드 콘솔이 아니라 사용자와 모델 사이에서 관리해야 하는 문제

- 호출을 한 지점으로 모으면 사용자별 키 · 사용량 · 허용 모델 · API 형식을 한곳에서 다룰 수 있음 → LLM Gateway를 두는 이유



KEY INSIGHT

**LLM Gateway 는 토큰과 모델 단위로 제어함**  
— 요청 수만 세는 API Gateway 로는 비용 상한을 둘 수 없음

| 비교 항목  | API Gateway           | LLM Gateway              |
|--------|-----------------------|--------------------------|
| 제한 단위  | 요청 수 (RPS · RPM)      | 토큰 수 (TPM) 와 예산 금액       |
| 라우팅 기준 | URL 경로와 백엔드 서비스       | 모델 이름(별칭)과 프로바이더         |
| 장애 대응  | 재시도, 서킷 브레이커          | 다른 모델이나 프로바이더로 폴백        |
| 비용 집계  | 호출 건수 기준              | 모델별 토큰 단가로 요청마다 계산       |
| 본문 처리  | 헤더와 경로 중심, 본문은 그대로 전달 | 프롬프트와 응답을 검사(가드레일)하고 기록  |
| 응답 형태  | 짧은 요청과 응답             | 스트리밍 응답이 많아 TTFT 를 따로 측정 |

### 두 Gateway 의 공통 기능

- 인증과 접근 제어
- 요청 로깅과 지표 수집
- 여러 백엔드로 부하 분산

### 함께 쓰는 구성

- API Gateway 뒤에 LLM Gateway 를 따로 배치
- API Gateway 에 LLM 기능을 확장 (Kong · Higress 방식)
- LLM 전용 Gateway 만 단독으로 배치 (LiteLLM 방식)



**KEY INSIGHT** 실제 프로바이더 키는 게이트웨이 안에만 두고, 예산·한도를 건 Virtual Key 를 팀에 나눠 주면 사고가 커지기 전에 멈출 수 있음

| 항목        | 게이트웨이 없음 (직접 연동) | LiteLLM 게이트웨이 도입 <span>권장</span> |
|-----------|------------------|----------------------------------|
| API 키 관리  | 앱마다 분산 저장        | 게이트웨이에서 중앙 발급·회수                 |
| 팀별 사용량 집계 | 불가 · 수작업 취합      | Virtual Key 단위로 자동 집계            |
| 예산 상한     | 없음 (청구서로 사후 확인)  | 팀·카·모델별 사전 한도 설정                 |
| 비용 초과 대응  | 사후 역추적           | 한도 도달 시 실시간 차단·알림                |
| 감사 추적     | 로그가 앱마다 흩어짐      | 요청 단위 중앙 로깅                      |
| 접근 권한 회수  | 키를 쓰는 코드를 일일이 수정 | 해당 Virtual Key 하나만 비활성화          |

## SECTION 03

# LiteLLM 의 개념과 OpenAI 호환 API

프로바이더 통합 방식, SDK 와 Proxy, 최근 변화



# 100개 넘는 프로바이더를 OpenAI 호환 API 하나로 호출

03 · LiteLLM의 개념과 OpenAI 호환 API

## KEY INSIGHT

**LiteLLM은 프로바이더마다 다른 호출 방식을 OpenAI 형식 하나로 바꿔 주는 오픈소스 변환 계층**  
성능을 높이는 도구가 아니라 표준화 도구. 호출 코드는 그대로 두고 model 값만 바꾸면 다른 모델을 호출

## 1 요청이 하나의 형식으로 오가는 구조 앱 → LiteLLM → 프로바이더

100+ 프로바이더

### 앱 · 에이전트

OpenAI 형식으로 요청 1개만 작성

```
completion(
  model="anthropic/claude",
  messages=[...]
)
```

모델 교체 = model 값만 변경

### LiteLLM 변환 계층

인증 헤더 프로바이더별 키 방식

파라미터 이름 max\_tokens 등 이름 차이

엔드포인트 · 모델 ID 주소와 식별자 매핑

응답 스키마 OpenAI 형식으로 되돌림

요청은 프로바이더 형식으로, 응답은 OpenAI 형식으로

### 상용 API 모델

외부 SaaS 형태의 대형 모델

기존 차이

인증 헤더 · 파라미터 · 응답 스키마가 서로 다름

### 오픈 웨이트 모델

자체 배포 가능한 공개 모델

기존 차이

엔드포인트 형식 · 파라미터 이름이 다름

### 자체 호스팅 추론 엔진

사내 GPU 위 vLLM 등

기존 차이

접속 경로 · 모델 식별자 매핑이 다름

## 2 LiteLLM 개요 만든 곳, 쓰는 방식, 확인할 점

### 만든 곳

BerriAI 오픈소스 프로젝트

- GitHub BerriAI/litellm 공개 저장소
- 소스 공개로 요청 변환 과정을 코드에서 검증 가능

### 쓰는 방식

Python SDK 또는 Proxy Server

- SDK: 앱 코드 안에서 라이브러리로 호출
- Proxy: 별도 서버로 띄워 여러 클라이언트가 공유 (이 문서의 Gateway)

### 확인 필요

버전과 기능 범위

- 저장소 생성 2023년 7월 27일
- 지원 프로바이더와 릴리스 이력은 GitHub에서 확인

**Gateway로 쓰는 이유** 호출 형식이 하나로 모여야 키 발급, 비용 집계, 모델 교체를 한 곳에서 처리할 수 있음



## KEY INSIGHT

LiteLLM 은 요청 · 응답 · 예외를 OpenAI 형식 하나로 통일함  
— 모델 교체가 설정 변경으로 끝나고 재시도와 폴백을 공통으로 쓸 수 있음

### 통일 전: 세 층마다 프로바이더별 코드

#### 요청

파라미터 이름, 메시지 구조, 인증 방식

→ 프로바이더별 요청 조립 코드

#### 응답

생성 결과의 필드 위치, 토큰 사용량 필드

→ 응답 해석 코드와 사용량 집계 코드

#### 예외

Rate Limit, 인증 실패, 타임아웃, 컨텍스트 초과 오류

→ 재시도 · 중단 판단 코드

창업자 설명: Azure 와 Cohere 추가 과정의 경험  
호출마다 100줄 안팎의 if/else 분기가 여러 개 필요

### 통일 후: OpenAI 형식 하나로 호출



- **호출 코드 하나**  
요청 조립과 응답 해석이 프로바이더와 무관, 모델 선택은 설정 변경으로 처리
- **같은 종류의 예외**  
Rate Limit 이나 타임아웃이 어느 프로바이더에서 발생해도 같은 예외로 반환
- **통일 뒤에도 남는 일**  
프로바이더가 지원하지 않는 파라미터 처리와 작업별 모델 선택



## KEY INSIGHT

OpenAI 호환은 요청·응답 형식이 같다는 뜻임  
— base\_url 만 바꾸면 기존 SDK 코드로 호출됨

### OpenAI 형식의 요청

POST /v1/chat/completions  
Authorization: Bearer sk-...

```
{
  "model": "gpt-4o",
  "messages": [
    {"role": "user", "content": "안녕하세요"}
  ]
}
```

### OpenAI 호환의 의미

- 주소 경로와 요청 · 응답 JSON 형식이 같음
- OpenAI SDK 의 base\_url 만 바꿔 호출 가능
- 모델 품질과 지원 파라미터는 서버마다 다름

### 용어

### 뜻

#### base\_url

API 서버 주소. 이 값을 바꾸면 다른 서버를 호출함

#### 토큰 (token)

모델이 처리하는 텍스트 단위  
입력 토큰과 출력 토큰 수로 과금됨

#### RPM · TPM

분당 요청 수와 분당 토큰 수  
Rate Limit 의 기준 값

#### 모델 그룹

클라이언트가 부르는 이름(model\_name)  
이 자료에서는 별칭(alias)으로 표기

#### 배포 (deployment)

모델 그룹 뒤에 연결된 실제 모델과 주소

#### Virtual Key

Proxy 가 발급하는 클라이언트용 키  
마스터 키는 관리 작업에만 사용



## KEY INSIGHT

클라이언트가 둘 이상이고 키 · 예산 · 로그 담당자가 있으면 Proxy 서버가 필요함

— 이런 통제는 호출 경로 중간에서만 걸 수 있음

BerriAI 창업

## 2023년

Y Combinator Winter 2023 배치 참여

공개 저장소 생성

## 2023-07-27

Python SDK 와 FastAPI 서버로 출발

최신 릴리스

## v1.104.0

2026-10-03 배포

저장소 스타

## 60,173

포크 12,046개

### LiteLLM SDK: 애플리케이션에 넣는 라이브러리

- 해결하는 문제: 프로바이더별 호출 형식 차이
- 프로바이더 키가 애플리케이션 수만큼 흩어짐
- 사용 기록은 애플리케이션마다 따로 보관

### LiteLLM Proxy: 플랫폼 팀이 구축하는 게이트웨이 서버

- 해결하는 문제: 키 · 비용 · 접근 통제의 중앙 관리
- 프로바이더 접속 정보는 서버 설정 한 곳에만 보관
- Virtual Key 로 비용 추적, 모델 접근 통제, 예산 설정

### 단일 엔드포인트의 의미

- OpenAI 호환 클라이언트는 접속 주소와 키만 바꿔 연결
- 에이전트 프레임워크는 사용자 지정 엔드포인트로 설정
- 주소를 바꾸지 않은 스크립트는 계속 프로바이더를 직접 호출
- 통제는 실제로 엔드포인트를 거치는 호출에만 적용



## KEY INSIGHT

차이는 애플리케이션 안에 두느냐 밖에 두느냐임  
— 밖에 두면 여러 앱이 같은 정책을 공유함

### </> LangChain (라이브러리)

- 애플리케이션 코드 안에서 여러 API 를 호출
- JBS 발표의 평가: 기능이 가장 풍부한 라이브러리
- 시연 9번: with\_fallbacks 로 로컬 우선, 실패 시 OpenAI 호출
- 보충: 2025-10 에 1.0 출시



### OpenRouter (호스팅형 게이트웨이)

- 주요 클라우드 LLM 을 한 API 로 호출
- OpenAI SDK 그대로 사용
- 프로바이더 장애 시 자동 우회
- 운영을 서비스에 위임



### LiteLLM (자체 호스팅 게이트웨이)

- 100개 이상 LLM 을 OpenAI 형식으로 호출
- 모델 접근, 폴백, 비용 추적
- 사내에 직접 배포하므로 로컬 LLM 연결 가능

### 두 게이트웨이의 차이 (MSAP.ai 해석)

OpenRouter 는 인터넷의 서비스라 사내 로컬 LLM 을 뒤에 둘 수 없음. 혼합 구성의 프록시 자리에는 자체 호스팅 게이트웨이가 필요  
JBS 발표에는 두 제품의 차이에 대한 설명이 없음



## KEY INSIGHT

ADK 는 LiteLLM 을 라이브러리로 사용함  
— 프록시가 아닌 SDK 래퍼라는 점을 구분해야 함

### LiteLLM 으로 다른 모델 지정 (문서의 형식)

```
from google.adk.agents import Agent
from google.adk.models.lite_llm import LiteLlm

agent = Agent(
    name="assistant",
    model=LiteLlm(model="<프로바이더>/<모델 이름>"),
    instruction="...",
)
```

- 프로바이더별 API 키는 환경 변수로 설정
- 이 연결은 ADK Python 에서만 지원 (문서 기준)

### ADK 개요

- 오픈소스, 코드 우선 에이전트 프레임워크
- Gemini 지원, 그 밖의 LLM 은 LiteLLM 경유
- 다중 에이전트와 순차, 병렬, 반복 오케스트레이션
- 도구 호출은 MCP, 다중 에이전트는 A2A

### i 2026-10 기준 보충

- Python v2.11.0 (2026-10-02)
- 지원 언어: Python, TypeScript, Go, Java, Kotlin
- 문서에 LiteLLM 1.82.7, 1.82.8 관련 보안 경고 수록
- 자체 호스팅 서버 주소는 api\_base 인자로 지정



KEY INSIGHT

에이전트용 기능이 늘고 릴리스 체계가 정비됨  
— 발표 당시 한계 일부는 해소됐으나 Enterprise 전용 기능 범위는 그대로임

최신 Stable 버전

**v1.104.0**

2026-10-03 릴리스

Stable 릴리스 주기

**매주**

v1.84.0 부터 시맨틱 버저닝

GitHub 스타

**약 6만**

2026-10-06 조회

프로바이더

**138종**

가격표 파일 직접 집계

 새로 추가된 에이전트용 기능

- **MCP 게이트웨이**  
등록한 MCP 서버를 한 경로로 모으고 키 단위로 접근 제어
- **A2A 지원**  
에이전트 호출에도 Virtual Key 인증과 비용 집계
- **Responses API**  
OpenAI 규격 엔드포인트를 다른 프로바이더로 변환

발표 당시의 한계

2026년 10월 상태

Enterprise 전용 기능

그대로. SSO 는 5명까지 무료로 완화

잡은 버그

릴리스 체계 강화. 다만 보안 사고가 새로 발생

호스팅 부담

Helm 차트와 Terraform 모듈 제공

Prometheus 지표

오픈소스로 사용 가능

캐시 토큰 비용 계산은 도입 때 청구서와 대조해 확인 필요



**KEY INSIGHT** LiteLLM 은 모델 호출의 단일 진입점 — 키·별칭·폴백·비용을 한곳에서 관리함



## SECTION 04

# LiteLLM Proxy의 핵심 기능

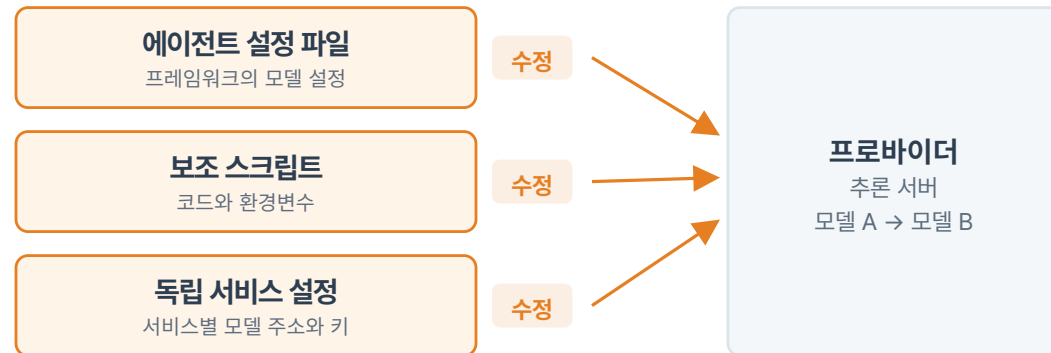
별칭, Virtual Key, 예산, 라우팅, 가드레일, 캐싱



## KEY INSIGHT

프로바이더가 모델을 바꿀 때도 같은 절차로 처리함  
— 고칠 곳이 설정 한 곳뿐이라 일부만 바뀐 상태가 생기지 않음

### LLM Gateway 없는 구성: 클라이언트마다 수정



수정 대상 세 곳과 빠진 곳을 확인하는 작업이 함께 발생

### LiteLLM Proxy 구성: 별칭 정의 한 곳만 수정



클라이언트가 호출하는 이름은 그대로, 뒤에 연결된 백엔드만 교체

### 별칭(alias)

클라이언트가 요청에 적는 모델 이름을 실제 백엔드에 연결하는 프록시 설정 항목

### 장애 대응의 단위

폴백은 어느 별칭이 실패하면 어느 별칭으로 전환할지를 목록으로 정의

### 효과가 없는 경로

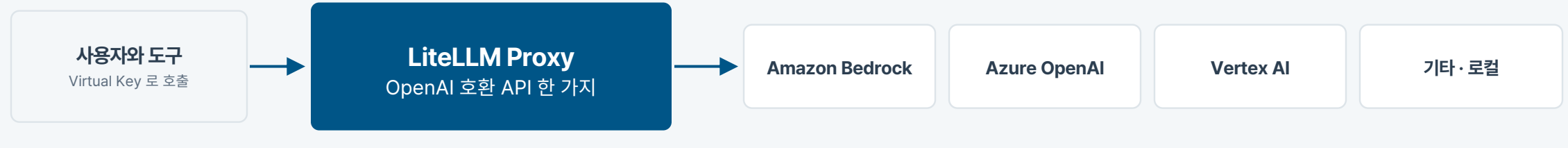
프록시 밖 경로 3건은 모델을 바꿀 때 경로마다 따로 수정



## KEY INSIGHT

프로바이더 앞에 Proxy 를 두고 Virtual Key 로 접근을 관리함  
— 경로가 하나가 되므로 비용과 권한 통제를 거기에 걸 수 있음

### LiteLLM 이 하는 일



1

#### 비용·사용량 모니터링과 제한

키, 사용자, 팀 단위 예산과 호출 한도

대응하는 과제

**비용 관리와 모니터링**



2

#### 권한 관리

Virtual Key 발급과 삭제, 모델 접근 제한

대응하는 과제

**권한 관리, 모델 제한**



3

#### 확장성

모니터링 도구와 가드레일 연동

대응하는 과제

**발표 시점 미도입**



4

#### 사실상 표준화

코딩 도구와 집계 도구가 먼저 지원

대응하는 과제

**여러 프로바이더 대응**



## KEY INSIGHT

프로바이더 키는 Proxy 만 보유함

— 유출되어도 Virtual Key 하나만 폐기하면 되고 피해는 그 키의 예산 안에 그침

### 예산과 한도가 걸리는 계층

#### 조직 (Organization)

팀의 상위 묶음. Enterprise 기능

#### 팀 (Team)

허용 모델, 팀 예산, RPM · TPM

#### 사용자 (User)

개인 예산과 한도

#### Virtual Key

sk- 로 시작하는 키. 유효 기간, 허용 모델, 예산

### 요청이 지나는 길

#### 사용자 · 도구

Virtual Key 만 보유

#### LiteLLM Proxy

프로바이더 자격증명 보유

#### 프로바이더

Bedrock, OpenAI 등

요청의 user 값으로 최종 고객(end-user)까지 비용을 나눠 집계



### Virtual Key 로 정하는 것

#### • 키 발급 때 정하는 값

허용 모델, 최대 예산과 초기화 주기, RPM · TPM 한도, 유효 기간, 소속 팀과 사용자, 메타데이터

#### • 전제 조건

PostgreSQL 과 마스터 키 (sk- 로 시작)

#### • 오픈소스 범위

Virtual Key, 예산, 지출 추적. 키 자동 교체와 감사 로그는 Enterprise



### primeNumber 의 적용 방식

#### • 권한 단순화

Proxy 실행 Role 에 bedrock:InvokeModel 만 부여. 개인별 IAM 발급 불필요

#### • 사고 대응과 단계 개방

문제가 생기면 관리자가 키 삭제. 최신 모델은 R&D 팀에 먼저 허용



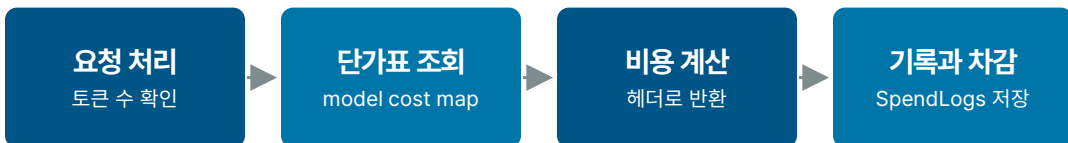
## KEY INSIGHT

비용은 내장 단가표로 계산됨  
— 청구서와 주기적으로 대조해야 함

### 한도를 거는 단위

| 단위          | 걸 수 있는 한도                    | 용도             |
|-------------|------------------------------|----------------|
| Virtual Key | 최대 예산(USD), 초기화 주기, RPM, TPM | 도구나 서비스 하나의 상한 |
| 사용자         | 개인 월 예산과 호출 한도               | 개인별 사용량 제한     |
| 팀           | 팀 예산, 허용 모델, RPM, TPM        | 부서별 비용 배분      |
| 모델          | 모델별 예산과 한도                   | 비싼 모델의 사용 억제   |

### 요청 한 건의 비용이 계산되는 순서



건별 비용은 응답 헤더 x-litellm-response-cost 로 반환, 배포별 단가 직접 지정 가능

### primeNumber 관리 화면의 수치 (2025-06-18 ~ 07-15)

총 요청

31,124

성공

8,899

실패

22,225

실패가 성공보다 많은 이유는 발표에 설명이 없음. 금액과 토큰 수는 가림 처리. 사용량 1위 모델은 Claude Sonnet 4

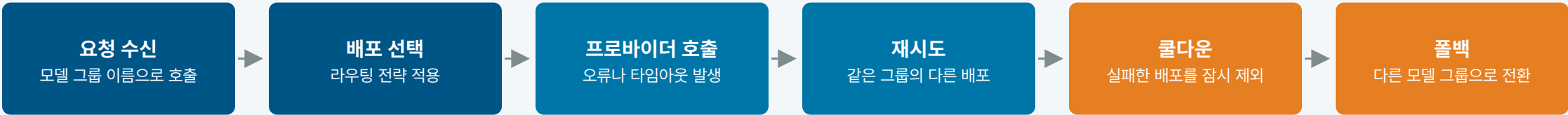
### 비용 수치를 그대로 믿기 전에 확인할 점

- **Anthropic 캐시 토큰 비용 오계산**  
캐시된 토큰이 제대로 반영되지 않아 실제보다 높게 집계 (primeNumber 보고)
- **로컬 모델과 신규 모델**  
단가표에 값이 없으면 비용 0 으로 기록. 배포에 단가 직접 입력
- **집계 축**  
키, 사용자, 팀, 최종 고객, 태그, 모델



KEY INSIGHT

폴백은 오류와 타임아웃에만 반응함  
— 응답이 느려진 상태는 별도로 모니터링해야 함



| 라우팅 전략                 | 배포를 고르는 기준                    |
|------------------------|-------------------------------|
| simple-shuffle         | RPM · TPM 가중 무작위. 기본값이자 운영 권장 |
| least-busy             | 진행 중인 요청이 가장 적은 배포            |
| latency-based-routing  | 응답 지연이 가장 짧은 배포               |
| usage-based-routing-v2 | 분당 토큰 사용이 가장 적은 배포. Redis 필요  |
| cost-based-routing     | 단가표 기준 가장 싼 배포                |

운영 문서 권고: 트래픽이 많은 배포에서는 usage 기반 라우팅 회피

## 폴백 세 종류

- 일반 폴백  
모델 그룹이 실패하면 지정한 다른 그룹으로 전환
- 컨텍스트 길이 초과 폴백  
더 큰 컨텍스트 모델로 전환
- 콘텐츠 정책 위반 폴백  
정책이 다른 프로바이더로 전환

## 기본값과 사각지대

- 클다운 기본값  
분당 실패 3회 초과 시 5초 제외
- 반응하지 않는 경우  
응답은 오지만 느려진 상태, 출력 품질이 떨어진 상태
- 세션 고정  
헤더로 같은 배포에 유지 (기본 1시간)



## KEY INSIGHT

재시도를 늘리면 대기가 길어지고 쿨다운을 늘리면 복귀가 늦어짐  
— 쿨다운은 추론 서버의 모델 로딩 시간에 맞춰 정해야 함



| 폴백 설정 키                         | 의미                         |
|---------------------------------|----------------------------|
| <b>fallbacks</b>                | 일반 오류로 실패했을 때 전환할 별칭 목록    |
| <b>context_window_fallbacks</b> | 입력이 컨텍스트 한도를 넘었을 때 전환할 별칭  |
| <b>content_policy_fallbacks</b> | 콘텐츠 정책 위반으로 거부됐을 때 전환할 별칭  |
| <b>default_fallbacks</b>        | 기본 폴백 대상, 적용 조건은 공식 문서로 확인 |

| 안정성 설정 키               | 의미                           |
|------------------------|------------------------------|
| <b>num_retries</b>     | 모델별 재시도 횟수                   |
| <b>allowed_fails</b>   | 1분 안에 이 횟수를 넘겨 실패하면 쿨다운으로 전환 |
| <b>cooldown_time</b>   | 쿨다운 상태를 유지하는 시간 (초)          |
| <b>request_timeout</b> | 요청 한 건의 응답을 기다리는 제한 시간       |

여덟 설정 모두 설정 파일에 적고 데이터베이스 없이 동작, SDK 와 Proxy 에서 의미가 같음



KEY INSIGHT

마스킹은 비스트리밍 응답에서만 가능함  
— 스트리밍 서비스는 차단만 되므로 개인정보 처리 방식을 따로 정해야 함

가드레일이 실행되는 네 시점

|              |                                |
|--------------|--------------------------------|
| pre_call     | 호출 전 입력 검사<br>개인정보 마스킹, 인젝션 차단 |
| during_call  | 호출과 병렬 검사<br>지연을 늘리지 않고 차단     |
| post_call    | 입력과 출력 검사<br>응답 속 민감 정보 확인     |
| logging_only | 기록할 때만 검사<br>응답은 그대로, 로그만 가림   |

- 연동 대상: Presidio(개인정보 마스킹), Amazon Bedrock Guardrails, Lakera, Azure Content Safety 등
- 본문 수정(마스킹)은 비스트리밍에서만 가능, 스트리밍은 차단만 가능
- Presidio 는 오픈소스, 키와 팀별 가드레일 지정은 Enterprise

한 경로로 모이는 호출 기록



- 본문 제외 옵션**  
turn\_off\_message\_logging 을 켜면 메시지와 응답을 빼고 지출 같은 메타데이터만 기록
- 부수 효과**  
모든 요청이 기록되어 클라이언트 도구의 시스템 프롬프트까지 열람 가능
- 지표 보호**  
v1.85.0 부터 /metrics 에도 API 키 인증이 기본



## KEY INSIGHT

같은 요청은 캐시에서 응답해 비용과 지연이 줄어듦  
— 시맨틱 캐시는 임계값에 따라 다른 질문에 같은 답이 나갈 수 있음

| 캐시 종류 (type)                      | 특징                                      |
|-----------------------------------|-----------------------------------------|
| local                             | Proxy 메모리에 저장. 인스턴스 하나 안에서만 유효          |
| redis                             | 여러 Proxy 인스턴스가 캐시를 공유                   |
| redis-semantic<br>qdrant-semantic | 프롬프트를 임베딩해 유사도로 판정<br>비슷한 질문에 같은 응답을 반환 |
| s3 · gcs · disk                   | 저장소나 디스크에 보관                            |

## config.yaml 설정

```
litellm_settings:
  cache: True
  cache_params:
    type: redis
    ttl: 600 # 초 단위 유효 시간
    # 시맨틱 캐시일 때
    # similarity_threshold: 0.8
```

## 요청별 캐시 제어

- no-cache: 캐시를 무시하고 모델을 호출
- no-store: 응답을 캐시에 저장하지 않음
- ttl · s-maxage: 요청마다 유효 시간 지정
- namespace: 팀이나 용도별로 캐시를 분리
- 응답 헤더 x-litellm-cache-key 로 캐시 항목 확인

## 적용 조건

- 같은 입력이 반복되는 단발 요청에 적합
- 시맨틱 캐시는 similarity\_threshold 로 일치 기준 지정
- 여러 턴이 이어지는 에이전트 대화에는 시맨틱 캐시가 적합하지 않음 (공식 문서의 안내)
- 인스턴스가 둘 이상이면 redis 로 공유

## SECTION 05

# SDK·Proxy 선택과 설치 구성

두 형태의 운영 차이, 최소 설치, 클라이언트 연결



KEY INSIGHT

SDK 는 장애 영향이 앱 하나에 그치고 Proxy 는 키와 설정을 한곳에 모음  
— Proxy 를 도입하면 서버 이중화와 헬스체크 계획도 함께 세워야 함

| 항목      | LiteLLM SDK             | LiteLLM Proxy                                       |
|---------|-------------------------|-----------------------------------------------------|
| 설치 위치   | 파이썬 애플리케이션의 프로세스 안      | 애플리케이션 밖의 공용 서버                                     |
| 사용 팀    | LLM 프로젝트를 만드는 개발자       | 조직의 LLM 접근을 관리하는 플랫폼 팀                              |
| 선택 시점   | 클라이언트가 하나일 때            | 클라이언트가 둘 이상이고 키를 한 곳에서 관리해야 할 때                     |
| 키 관리    | 프로바이더 키를 애플리케이션마다 보관    | 프로바이더 키는 서버에만 두고 클라이언트에게 Virtual Key 발급             |
| 장애 범위   | 해당 애플리케이션 하나            | Proxy 뒤의 모든 클라이언트 (단일 장애점, single point of failure) |
| 수반되는 부담 | 애플리케이션이 늘면 키와 설정이 함께 증가 | 서버 운영과 가용성 책임                                       |

Proxy 가 하지 않는 일

토큰 생성은 뒤의 추론 서버와 프로바이더가 담당 — Proxy 를 도입해도 모델의 생성 속도는 그대로

계층 분리의 이점

추론 서버를 교체하거나 늘릴 때 별칭 뒤의 배포 정의만 변경, 게이트웨이의 키와 폴백 정의는 유지



KEY INSIGHT

재시도 · 폴백 · 부하 분산은 SDK 에서도 동작함  
— 서버를 따로 두는 이유는 Virtual Key, 예산, 중앙 로그 세 가지임

| 기능             | 소속          | LiteLLM SDK | LiteLLM Proxy       |
|----------------|-------------|-------------|---------------------|
| 재시도            | 공통 (Router) | 사용 가능       | 사용 가능               |
| 폴백             | 공통 (Router) | 사용 가능       | 사용 가능               |
| 부하 분산          | 공통 (Router) | 사용 가능       | 사용 가능               |
| Virtual Key    | Proxy 전용    | 없음          | 사용 가능 (Postgres 필요) |
| 키 · 팀 · 사용자 예산 | Proxy 전용    | 없음          | 사용 가능 (Postgres 필요) |
| 관리 화면          | Proxy 전용    | 없음          | 사용 가능               |
| 중앙 로깅          | Proxy 에서 제공 | 애플리케이션별 콜백만 | 사용 가능               |

공통 기능의 차이는 설정 위치

- SDK: 폴백 순서를 애플리케이션마다 따로 관리
- Proxy: 설정 파일 한 곳의 순서를 모든 클라이언트가 따름
- 폴백과 재시도는 Proxy 의 고유 장점으로 보지 않음

SDK 로 충분한 경우

- 클라이언트가 하나
- 비용을 클라이언트별로 나눠 볼 필요가 없음
- 키를 애플리케이션 담당자가 직접 관리해도 되는 상황



💡 KEY INSIGHT

Virtual Key 와 예산은 오픈소스 버전 기능이라 추가 계약 없이 사용 가능  
— 대신 데이터베이스 백업과 복구 업무가 생기므로 도입 시점은 따로 판단해야 함

| 항목                        | YAML 단독 구성      | Postgres 연결 구성            |
|---------------------------|-----------------|---------------------------|
| 폴백, 재시도, 쿨다운, 타임아웃, 부하 분산 | 사용 가능           | 사용 가능                     |
| 인증                        | 마스터 키 하나        | 마스터 키와 클라이언트별 Virtual Key |
| 비용 추적                     | 없음              | 키 단위로 추적                  |
| 모델 접근 통제                  | 없음              | 키마다 허용 모델 지정              |
| 예산                        | 없음              | 키 · 사용자 · 팀 단위            |
| 운영 대상                     | 컨테이너 한 개와 설정 파일 | 컨테이너와 Postgres 데이터베이스     |
| 복구 방법                     | 컨테이너 재시작        | 컨테이너 재시작과 데이터베이스 복구       |

YAML 단독 구성이 적합한 조건

클라이언트가 적고 서로 신뢰하는 범위 안에 있으며 비용을 나눠 볼 필요가 없는 경우

Postgres 연결이 선행 조건인 경우

클라이언트별 사용량을 나눠 보거나 유출된 키 하나만 교체하려는 경우, 클라이언트별 모니터링을 계획하는 경우



## KEY INSIGHT

Proxy 컨테이너는 CPU 만으로 실행됨

— GPU 는 api\_base 가 가리키는 추론 서버에만 필요함

### docker-compose.yaml 최소 구성

```
services:
  litellm:
    image: ghcr.io/berriai/litellm:<버전 태그>
    ports:
      - "4000:4000"
    volumes:
      - ./config.yaml:/app/config.yaml
    command: ["--config", "/app/config.yaml",
              "--port", "4000"]
    environment:
      LITELLM_MASTER_KEY: "sk-..."
    restart: unless-stopped
```

### 실행과 확인

- docker compose up -d 로 실행
- 4000 포트에서 OpenAI 호환 요청을 수신
- /health/liveliness 로 실행 여부 확인

### 최소 설치 요건 세 가지

- 컨테이너 런타임 (Docker)
- 공식 Proxy 이미지
- 모델 등록 규칙을 적은 config.yaml

### 이 구성에 없는 것

- PostgreSQL: Virtual Key 와 예산을 쓸 때 추가
- Redis: Proxy 를 둘 이상 실행할 때 추가

### 주의할 점

- 이미지 태그는 latest 대신 버전을 고정
- 마스터 키는 sk- 로 시작하는 값으로 지정
- GPU 는 Proxy 가 아닌 추론 서버에 필요



## KEY INSIGHT

같은 `model_name` 으로 배포를 여러 개 등록하면 부하가 분산됨  
— 클라이언트 코드는 바뀌지 않음

### config.yaml 의 네 구역

```
model_list:
  - model_name: chat-default # 클라이언트가 부르는 별칭
    litellm_params:
      model: hosted_vllm/<모델 이름>
      api_base: http://<추론 서버 1>
      rpm: 120
  - model_name: chat-default # 같은 별칭의 두 번째 배포
    litellm_params:
      model: hosted_vllm/<모델 이름>
      api_base: http://<추론 서버 2>
router_settings:
  routing_strategy: simple-shuffle
litellm_settings:
  fallbacks: [{"chat-default": ["chat-backup"]}]]
general_settings:
  master_key: os.environ/LITELLM_MASTER_KEY
```

### 구역

### 정하는 것

#### model\_list

별칭과 실제 모델 · 주소의 연결  
배포별 RPM · TPM

#### router\_settings

배포를 고르는 라우팅 전략  
Redis 연결 정보

#### litellm\_settings

폴백, 콜백, 캐시, 타임아웃

#### general\_settings

마스터 키, 데이터베이스 연결

### 설정 작성 시 확인할 점

- `os.environ/` 접두어로 환경변수 값을 읽음
- 키와 비밀번호는 파일에 직접 적지 않음
- 별칭이 같으면 한 모델 그룹으로 묶임



## KEY INSIGHT

기존 OpenAI SDK 코드는 base\_url 과 키 두 값만 바꾸면 됨  
— 모델 이름에는 별칭을 넣음

### Python (OpenAI SDK)

```
from openai import OpenAI

client = OpenAI(
    base_url="http://<Proxy 주소>:4000",
    api_key="sk-<Virtual Key>",
)
resp = client.chat.completions.create(
    model="chat-default", # 별칭
    messages=[{"role": "user", "content": "..."}],
)
```

### curl

```
curl http://<Proxy 주소>:4000/v1/chat/completions \
-H "Authorization: Bearer sk-<Virtual Key>" \
-H "Content-Type: application/json" \
-d '{
  "model": "chat-default",
  "messages": [{"role": "user", "content": "..."}]
}'

# 사용할 수 있는 별칭 조회
curl http://<Proxy 주소>:4000/v1/models ...
```

### 1. 바꾸는 값은 두 가지

- base\_url: Proxy 주소
- api\_key: Virtual Key
- 나머지 코드는 그대로 사용

### 2. 모델 이름에는 별칭

- config.yaml 의 model\_name 값
- /v1/models 로 목록 조회
- 실제 모델은 Proxy 가 결정

### 3. 응답 헤더로 확인

- x-litellm-response-cost: 건별 비용
- x-litellm-call-id: 요청 추적 ID
- 스트리밍도 같은 방식으로 호출



## KEY INSIGHT

**Claude Code 와 Codex 는 설정만으로 연결됨**  
— 정액 구독과 Cursor 사설망 구성은 Proxy 를 거치지 않음

### > Claude Code 설정 예 (~/.claude/settings.json)

```
{
  "env": {
    "ANTHROPIC_BASE_URL": "https://<litellm-url>",
    "ANTHROPIC_AUTH_TOKEN": "<virtual-key>",
    "ANTHROPIC_MODEL": "bedrock-claude-4-0-sonnet",
    "ANTHROPIC_SMALL_FAST_MODEL": "bedrock-amazon-nova-pro"
  }
}
```

- **인증 토큰 자리에 Virtual Key**  
마스터 키 대신 모델을 제한한 Virtual Key 사용
- **가벼운 작업용 모델도 함께 지정**  
Claude Code 가 내부에서 쓰는 소형 모델이 Proxy 에 없으면 오류 발생
- **/v1/messages 통합 엔드포인트**  
Anthropic 형식 요청을 다른 프로바이더 형식으로 변환, 비용 추적과 폴백 적용

| 도구          | 연결 방법                            | 주의점                                      |
|-------------|----------------------------------|------------------------------------------|
| Claude Code | 환경변수 두 개로 주소와 키 지정               | 소형 모델 지정 필요, 정액 구독 사용분은 미경유              |
| Codex CLI   | config.toml 에 사용자 프로바이더로 등록      | wire_api 값은 responses 필수                 |
| Cursor      | OpenAI Base URL 재정의에 Proxy 주소 입력 | Proxy 가 공개 주소여야 동작, Agent 모드는 v1.97.0 이상 |
| Dify · n8n  | OpenAI 호환 프로바이더로 등록              | 벤고시닷컴이 사용 중인 구성                          |

### 게이트웨이 통제 밖에 남는 사용

- **정액 구독 (Claude Max 등)**  
Proxy 를 거치지 않아 사용량과 본문이 기록되지 않음
- **Cursor 의 사설망 구성**  
요청을 Cursor 서버가 보내므로 사내망 안의 Proxy 에는 연결 불가
- **GitHub Copilot · Devin**  
벤고시닷컴 구성도에서 Proxy 미경유로 표시

## SECTION 06

# LLM Gateway 요건 평가와 대안 비교

요건 8개, Enterprise 버전 경계, 오픈소스·국내 대안



# LLM Gateway 가 갖춰야 할 요건

06 · LLM Gateway 요건 평가와 대안 비교

## KEY INSIGHT

기본 요건이 빠지면 클라이언트가 Gateway 를 우회해 직접 호출함  
— 상세 요건은 경로를 하나로 모으면서 생긴 보안·가용성 위험에 대응함



기본

### OpenAI 호환 엔드포인트와 모델 별칭

클라이언트 코드 수정 없이 경로 이전, 실제 백엔드를 클라이언트에게 숨김



기본

### 키 관리

프로바이더 키는 게이트웨이가 보관,  
클라이언트에게는 전용 키 발급



기본

### 폴백과 안정성 설정

배포 하나가 실패하면 클라이언트가 오류를  
받기 전에 다른 배포로 전환



기본

### 요청 · 응답 로깅

요청별로 별칭과 클라이언트가 함께 남아야  
충족



상세

### 클라이언트별 Virtual Key 와 예산

비용 추적, 모델 접근 통제, 키 · 팀 · 사용자  
단위 예산



상세

### 지연 · TTFT 모니터링

전체 지연, TTFT(Time To  
First Token), 폴백 발생 수집



상세

### 공급망 보안

이미지 버전과 다이제스트 고정, 릴리스  
확인 절차



상세

### LLM Gateway 가용성

배포 그룹과 폴백, 인스턴스 이중화와  
헬스체크

판정 기준은 기능의 존재보다 운영 중인 구성에서 활성화했는지 여부, 판정 값은 충족 · 부분 충족 · 미충족 세 가지

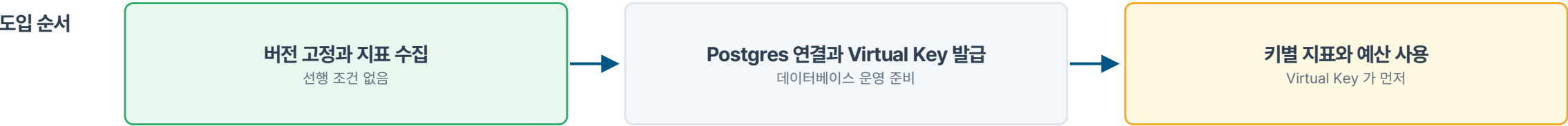


# YAML 설정만으로 Gateway 요건 8개 중 5개 충족

KEY INSIGHT

클라이언트를 구분하는 요건만 Postgres 가 필요함  
— 그래서 Postgres 연결 시점이 도입 순서를 정하는 기준이 됨

| 요건 항목                   | YAML 단독 구성 | Postgres 연결 구성 | 판정이 다른 이유                         |
|-------------------------|------------|----------------|-----------------------------------|
| OpenAI 호환 엔드포인트와 모델 별칭  | 충족 가능      | 충족 가능          | 설정 파일만으로 동작                       |
| 키 관리                    | 부분 충족에 그침  | 충족 가능          | 마스터 키만으로는 클라이언트를 구분하지 못함          |
| 폴백과 안정성 설정              | 충족 가능      | 충족 가능          | 설정 파일만으로 동작                       |
| 요청 · 응답 로깅              | 부분 충족에 그침  | 충족 가능          | 클라이언트를 기록하려면 Virtual Key 가 먼저 필요  |
| 클라이언트별 Virtual Key 와 예산 | 미충족        | 충족 가능          | Virtual Key 는 Postgres 가 있어야 발급   |
| 지연 · TTFT 모니터링          | 충족 가능      | 충족 가능          | 콜백 한 줄로 수집, 키별 구분만 Virtual Key 필요 |
| 공급망 보안                  | 충족 가능      | 충족 가능          | 배포 방식으로 해결                        |
| LLM Gateway 가용성         | 충족 가능      | 충족 가능          | 배포 방식으로 해결                        |





💡 KEY INSIGHT

요건 8개 중 제품 기능 6개는 오픈소스 버전에 포함됨  
— 평가표의 미충족 항목은 유료 기능이 없어서가 아니라 구성 선택 때문임

| 영역     | 오픈소스 버전              | Enterprise 버전 전용                                            |
|--------|----------------------|-------------------------------------------------------------|
| 인증     | 마스터 키, Virtual Key   | 관리 화면 SSO(Single Sign-On, 통합 로그인), JWT 인증, IP 접근 제어 목록      |
| 감사     | 요청 · 응답 로깅           | 감사 로그와 보관 정책, 팀별 로깅과 로그 내보내기                                |
| 시크릿 보관 | 해당 없음                | 외부 시크릿 저장소 연동 (AWS KMS, Azure Key Vault, HashiCorp Vault 등) |
| 예산     | 키 · 사용자 · 팀 단위 예산    | 태그 기반 예산과 프로젝트 관리                                           |
| 가드레일   | 오픈소스 버전 기능 목록에 기재 없음 | 키 · 팀 단위 가드레일                                               |
| 지표     | Prometheus 지표        | Managed Batch 지표 묶음                                         |

Enterprise 버전 검토가 필요한 조건

- 관리 화면 계정을 사내 계정 체계와 통합해야 하는 규정
- 키 생성과 폐기 이력을 감사 로그로 보관해야 하는 규정
- 프로바이더 키를 외부 시크릿 저장소에 두도록 한 규정

라이선스와 설치 방식

- 저장소는 MIT, enterprise/ 디렉터리는 별도 라이선스
- Enterprise 버전도 조직의 자체 인프라에 설치하는 방식
- 버전별 기능 범위는 도입 시점의 공식 문서로 재확인



## KEY INSIGHT

GitHub 스타 수는 그대로 비교할 수 없음

— Kong 의 수치에는 API 게이트웨이 용도가 포함돼 LLM 도입 규모로 보면 안 됨

| 프로젝트         | 스타     | 라이선스       | 저장소 생성    | 프로젝트 성격      |
|--------------|--------|------------|-----------|--------------|
| LiteLLM      | 60,173 | MIT        | 2023년 7월  | LLM 전용 게이트웨이 |
| Kong         | 44,242 | Apache-2.0 | 2014년 시작  | API 게이트웨이 확장 |
| Higress      | 9,492  | Apache-2.0 | 2022년 10월 | API 게이트웨이 확장 |
| Bifrost      | 8,565  | Apache-2.0 | 2025년 3월  | LLM 전용 게이트웨이 |
| Agent Router | 2,188  | Apache-2.0 | 2024년 10월 | 에이전트 라우터     |

LiteLLM 의 포크는 12,046개, enterprise/ 디렉터리는 별도 라이선스. 성격 구분은 저장소 설명 기준

## Agent Router (옛 Envoy AI Gateway)

- 2026년 6월 1.0 출시, 2026-09-09 Agentic AI Foundation 합류 공지
- 모델 쪽 OpenAI 호환 인터페이스와 도구 쪽 MCP(Model Context Protocol) 라우터를 함께 제공
- 관리자 9석을 다섯 회사가 분담, 과반을 가진 회사 없음

## 스타 수 해석 시 유의점

- 누적 관심도이며 설치 수나 사용량이 아님
- Kong 과 Higress 수치에는 API 게이트웨이 용도 포함
- 생성 시기가 다르면 같은 기간의 값이 아님
- 제작사가 발표한 성능 수치는 비교에서 제외

## 후보 선정 기준

- 공개 저장소
- 오픈소스 라이선스
- LLM 호출 중계 기능
- 도입 규모와 프로젝트 이력



## KEY INSIGHT

먼저 검토할 근거는 도입 사례 수와 오픈소스 버전의 기능 범위 두 가지임  
— 지연과 안정성은 공개 수치로 비교할 수 없어 직접 테스트해야 함

### 공개 수치와 문서로 확인되는 LiteLLM 현황

- **스타 60,173개, 포크 12,046개**  
스타 수는 다섯 후보 가운데 가장 많음
- **최신 릴리스 v1.104.0**  
2026-10-03 배포
- **MIT 라이선스**  
enterprise/ 디렉터리는 별도 라이선스
- **100개 이상 프로바이더**  
OpenAI 형식 하나로 호출
- **기본 요건이 오픈소스 버전에 포함**  
Virtual Key, 비용 추적, 예산, 폴백, 로깅
- **관리 담당**  
BerriAI 한 회사가 저장소를 관리하고 Enterprise 버전을 함께 제공

### 조직 조건

### 먼저 검토할 후보 유형

### 프로젝트

별도 API 게이트웨이 기반 없이 여러 에이전트와 서비스의 호출을 한 엔드포인트로 모으려는 조직

**LLM 전용 게이트웨이**

LiteLLM, Bifrost

같은 API 게이트웨이를 이미 운영하는 조직

**API 게이트웨이 확장형**

Kong, Higress

재단 소속과 중립 거버넌스가 도입 조건인 조직

**재단 거버넌스 프로젝트**

Agent Router

모델 호출과 MCP 도구 호출을 같은 프로젝트에서 중계하려는 조직

**에이전트 라우터**

Agent Router

### 비교 항목에서 제외한 것

- 토큰 생성 속도: 뒤에 연결한 추론 서버와 프로바이더가 결정
- 성능과 안정성: 공개 메타데이터로 비교할 수 없어 도입 조직이 직접 측정
- 표의 분류는 프로젝트 성격 기준이며 기능의 우열을 뜻하지 않음



**KEY INSIGHT** 제품이 유지되는지부터 확인해야 함 — 2026년에 인수·유지보수 전환·종료가 이어짐

## 자체 호스팅 · 유지 중

|                                                                           |                   |
|---------------------------------------------------------------------------|-------------------|
| <b>LiteLLM</b><br>프로바이더 폭이 넓고 Python 생태계와 결합<br><b>활발. 2026년 보안 사고 집중</b> | Python · MIT      |
| <b>Bifrost</b><br>낮은 오버헤드를 강조 (자체 측정)<br>릴리스가 잦음                          | Go · Apache-2.0   |
| <b>Kong AI Gateway</b><br>범용 API 게이트웨이 위의 AI 플러그인<br>기존 Kong 운영 조직에 유리    | Lua · 코어 가능       |
| <b>agentgateway</b><br>MCP 와 A2A 데이터 플레인 이 출발점<br>최신 v1.6.0 (2026-10-02)  | Rust · Apache-2.0 |

## 이름 변경 · 인수

|                                                                                                             |
|-------------------------------------------------------------------------------------------------------------|
| <b>Agent Router</b><br>Go · 가능 (Apache-2.0)<br>쿠버네티스 Gateway<br>API 중심<br>Envoy AI Gateway 에서<br>이름과 저장소 변경 |
| <b>Portkey Gateway</b><br>TypeScript · 가능 (MIT)<br>가드레일 내장<br>Palo Alto Networks 인수<br>(2차 자료)              |

## 유지보수 · 보관

|                                                                                                                 |
|-----------------------------------------------------------------------------------------------------------------|
| <b>Helicone · TensorZero</b><br>TypeScript · Rust<br>모니터링 중심,<br>최적화 통합<br><br>유지보수 모드<br>(2차 자료),<br>저장소 보관 처리 |
|-----------------------------------------------------------------------------------------------------------------|

## 호스팅 · 관리형

|                                                                                                |
|------------------------------------------------------------------------------------------------|
| <b>OpenRouter</b><br>호스팅 서비스 · 자체 호스팅 불가<br>키 하나로 여러 프로바이더,<br>결제 통합<br>운영 부담 없음.<br>통제 지점은 외부 |
| <b>Cloudflare AI Gateway</b><br>관리형 · 자체 호스팅 불가<br>엣지에서 캐싱과 한도,<br>폴백<br>모든 요금제에서 사용 가능        |

## LiteLLM 의 위치

강점은 Python 생태계와 프로바이더 폭. 약점은 Go · Rust 계열에  
뒤지는 성능과 2026년에 집중된 보안 사고

## 선택 순서

자체 호스팅 필요 여부 → 기존 인프라와의 결합(쿠버네티스, Kong)  
→ 프로젝트 지속성 확인

Source: 각 프로젝트 GitHub 저장소와 공식 문서 (2026-10-06 조회). 인수와 유지보수 전환은 2차 자료 기준



KEY INSIGHT

유형 분류로는 중심 기능만 알 수 있음  
— Virtual Key, 폴백 같은 기본 요건은 제안서 기능 목록을 평가표와 대조해야 확인됨

| 유형                   | 중심 기능                         | 적합 조건                                      |
|----------------------|-------------------------------|--------------------------------------------|
| SaaS형 통합 과금 게이트웨이    | 여러 프로바이더 사용료의 통합 정산           | 게이트웨이를 직접 운영하지 않으려는 조직, 프로바이더별 계약을 줄이려는 조직 |
| GPU 플랫폼에 결합된 모델 라우터  | 플랫폼에 올린 모델 사이의 요청 분배          | 온프레미스 GPU 플랫폼을 쓰고 있거나 함께 도입할 조직            |
| 보안 필터링 프록시           | 요청 · 응답 검사와 민감 정보 통제          | 외부 생성형 AI 사용을 통제해야 하는 보안 조직                |
| 글로벌 API 게이트웨이의 국내 공급 | 기존 API 게이트웨이 정책 체계에 LLM 중계 추가 | 같은 API 게이트웨이를 이미 운영하는 조직                   |
| 네트워크 어플라이언스형         | 네트워크 장비 운영 체계에 LLM 중계 추가      | 네트워크 조직이 LLM 호출 경로를 관리하는 조직                |

유형별로 먼저 확인할 것

- SaaS형: 호출 내용의 경유 경로와 로그 보관 방식
- GPU 플랫폼 결합형: 플랫폼 밖 클라우드 모델의 지원 범위
- 어플라이언스형: 설정 변경 절차가 별칭과 예산 변경 속도에 맞는지

제안서를 평가표와 대조하는 방법

- 유형을 먼저 구분한 뒤 요건 항목별 충족 여부를 따로 확인
- 점검 항목: OpenAI 호환 게이트웨이, Virtual Key, 비용 추적, 예산, 폴백, 요청 · 응답 로깅

## SECTION 07

# 로컬 LLM 과 클라우드의 혼합 구성

기밀 데이터 대응을 위한 로컬 모델 연결 방식



## KEY INSIGHT

클라우드를 못 쓰는 업무에만 로컬 LLM 을 적용함  
— 나머지 업무는 클라우드를 그대로 쓰는 편이 현실적

### 🔒 기밀 데이터

- 외부 전송이 허용되지 않는 데이터를 취급
- 클라우드 API 호출 자체가 불가

### \$ 종량 과금

- 비용 상한을 알 수 없어 자유로운 사용을 허용하기 어려움
- 예산 통제 수단이 없으면 이용 제한으로 이어짐

### 🏠 로컬 LLM 이 해법이 되는 이유

- 데이터가 사내를 벗어나지 않음
- 사용량이 늘어도 추가 과금 없음
- 엔지니어 개인이 제한 없이 사용 가능

### 👥 JBS 발표의 전제

- 생성 AI 활용은 사람과 조직에 따라 차이가 큼
- 주류는 클라우드형 LLM
- 대상 청중: 기업 정보시스템 부서, Microsoft 기술 관심층



## KEY INSIGHT

대부분 설정과 모델 크기로 해결되는 문제임

— 세 도구 모두 OpenAI 호환 API 를 제공해 주소만 바꾸면 전환됨

| 질문                                                                                                    | 보충 답변                                   | 근거                         |
|-------------------------------------------------------------------------------------------------------|-----------------------------------------|----------------------------|
|  애플리케이션을 다시 만들어야 하는가 | 세 도구 모두 OpenAI 호환 API 제공, 주소 변경으로 전환    | Ollama, vLLM, LM Studio 문서 |
|  로컬 LLM 은 성능이 낮은가    | 모델 크기에 좌우, 품질 비교 수치는 이번 조사 범위 밖         | 비교 수치 없음                   |
|  매우 비싼 GPU 가 필요한가    | 4bit 기준 10억 파라미터당 약 0.6~0.7GB 로 추정      | Ollama 라이브러리의 파일 크기        |
|  여러 사람이 동시에 쓸 수 있는가 | Ollama 는 기본 1 이라 설정 필요, vLLM 은 연속 배칭 제공 | Ollama FAQ, vLLM 문서        |
|  쉽게 실행할 수 있는가      | 데모 코드의 명령이 한두 줄                         | JBS 샘플 저장소                 |

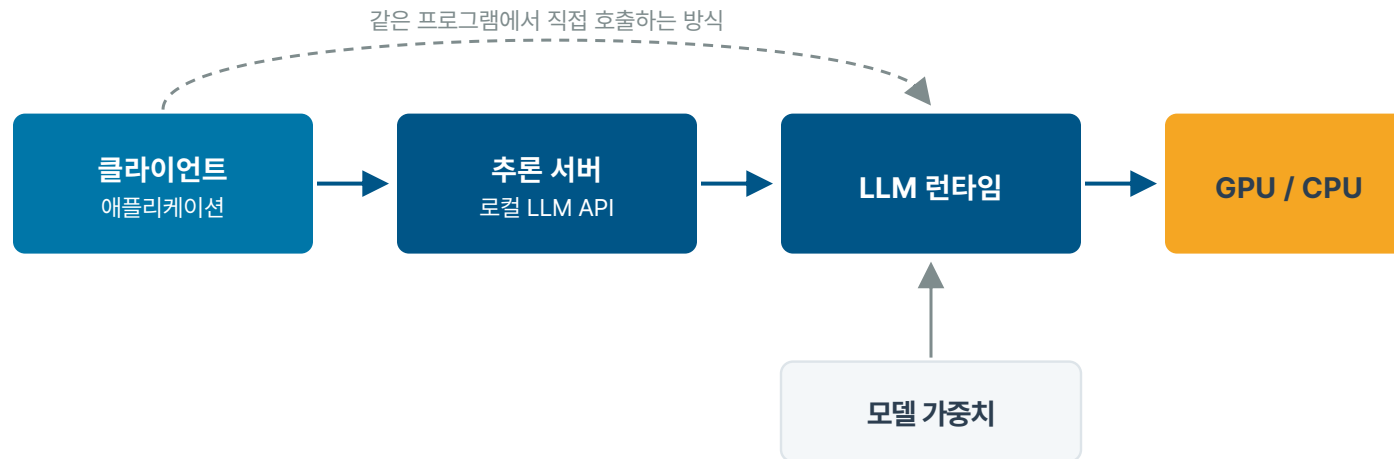


## KEY INSIGHT

### 서버 방식이 공유에 유리함

— 여러 애플리케이션이 같은 모델을 HTTP 로 호출할 수 있기 때문

## 로컬 LLM 의 실행 구조



### • 서버 공유

여러 애플리케이션이 HTTP 로 같은 모델을 공유

### • 직접 호출

프로그램 안에서 런타임을 실행, 데모 코드 1번이 이 방식

## Hugging Face

- AI 모델을 내려받는 공개 저장소
- JBS 발표의 설명: 모델의 GitHub 또는 Docker Hub 에 해당
- 가중치를 받아 런타임이 로드



KEY INSIGHT

개인 실험용과 다중 사용자 서빙용은 요구 사항이 다름  
— 쉽게 시작할지, 처리량이 중요한지로 고름

| 도구               | 조작 방식 | JBS 발표의 소개       | 2026-10 기준 보충                                          |
|------------------|-------|------------------|--------------------------------------------------------|
| Ollama           | CUI   | 간단한 명령으로 조작      | v0.35.1, MIT, OpenAI 호환 API 와 도구 호출 지원                 |
| LM Studio        | GUI   | 가장 쉬운 선택지        | 2025-07-08 부터 업무 사용도 무료, 제품 표기가 LM Studio Bionic 으로 변경 |
| vLLM             | 서버    | 성능 좋은 LLM 서버 구축용 | v0.31.0, Apache-2.0, 연속 배치와 프리픽스 캐싱                    |
| llama.cpp server | 서버    | 발표에 없음           | Ollama 와 LM Studio 가 내부에서 쓰는 엔진의 원본, 병렬 슬롯 지원          |

개인 실험

## Ollama · LM Studio

명령 한두 줄 또는 GUI

여러 사용자 서빙

## vLLM

연속 배치로 처리량 확보

클러스터 규모

## Production Stack

라우터, 모니터링, 캐시 포함

JBS 발표 시점 (2025-05) 에는 LM Studio 의 업무 사용에 별도 라이선스가 필요



💡 KEY INSIGHT

OpenAI 호환 주소만 있으면 연결됨 — Tool Calling 지원과 컨테이너 네트워크는 따로 확인해야 함

### config.yaml 예 (이름과 주소는 예시)

```
model_list:
- model_name: local-chat
  litellm_params:
    model: ollama_chat/<모델 이름>
    api_base: http://host.docker.internal:11434
  model_info:
    supports_function_calling: true
- model_name: local-serving
  litellm_params:
    model: hosted_vllm/<모델 이름>
    api_base: http://<vLLM 서버 주소>
- model_name: cloud-chat
  litellm_params:
    model: <클라우드 모델>
```

시연 저장소에는 LiteLLM 설정 파일이 없어  
공식 문서 형식으로 작성

| 대상        | 모델 접두어            | 💡 KEY INSIGHT                                    |
|-----------|-------------------|--------------------------------------------------|
| Ollama    | ollama_chat/ (권장) | 기본 주소 localhost:11434, keep_alive 로 메모리 유지 시간 조절 |
| vLLM      | hosted_vllm/ (권장) | OpenAI 호환 서버 주소를 api_base 에 지정                   |
| LM Studio | lm_studio/        | LM_STUDIO_API_BASE 필수                            |

vLLM 의 api\_base 는 공식 문서 예시에서 /v1 없이 서버 주소만 지정

### 주의점

Tool Calling 미지원 모델은  
JSON 모드로 대체되므로  
지원 여부를 모델별로 명시

Docker 안의 localhost 는  
컨테이너 자신, 호스트는  
**host.docker.internal**

로컬 모델은 가격표에  
값이 없어 비용 추적에는  
토큰 단가 직접 입력

Source: docs.litellm.ai (Ollama, vLLM, LM Studio 프로바이더 문서)



## KEY INSIGHT

앱은 주소 하나만 알면 됨 — 어느 모델로 보낼지는 Proxy 설정으로 정함

## 앱이 보는 주소는 하나, 뒤에서 분기되는 호출

JBS 발표의 설명: 가능한 구성의 예시이며 권장 구성으로 제시한 것은 아님



### 로컬이 필요한 경우

외부 전송이 금지된 기밀 데이터, 상한을 알 수 없는 종량 과금

### config.yaml 에 로컬 모델을 등록하는 예 (발췌)

```
model_list:
- model_name: local-llama
  model: ollama_chat/llama3.1
  api_base: http://host.docker.internal:11434
  model_info: supports_function_calling: true
- model_name: local-vllm
  model: hosted_vllm/<모델 이름>
  api_base: http://<vllm 서버 주소>
```

### 연결할 때 주의점

- 1 Docker 로 실행한 Proxy 에서 localhost 는 컨테이너 자신  
호스트의 Ollama 는 host.docker.internal 로 지정
- 2 Tool Calling 을 지원하지 않는 모델은 JSON 모드로 대체하므로  
지원 여부를 모델별로 명시
- 3 로컬 모델은 가격표에 값이 없어 비용을 직접 입력

## SECTION 08

# LiteLLM 도입 사례 네 건

벤티고시닷컴, RAKUS, IVRy, primeNumber 의 구성과 한계

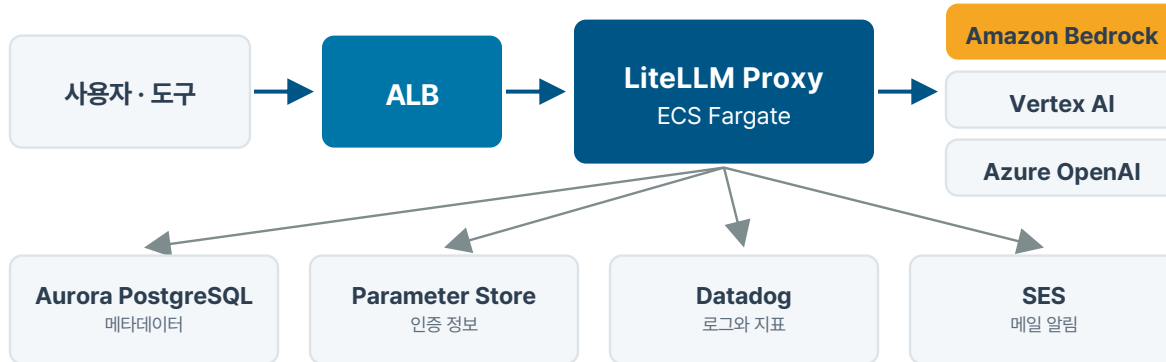


## KEY INSIGHT

도입 목적은 사용자별 비용 파악임

— 클라우드 세 곳에 따로 하던 신청과 사용자 관리를 한 곳으로 통합함

### AWS 위의 구성 (SRE Magazine 글 기준)



- 사용자 생성은 Slack 워크플로로 자동화
- 접근 로그를 Datadog 으로 모아 사용자, 모델, 토큰 단위로 비용 집계
- 활성 사용자는 수십 명 규모 (글 작성 시점)

### 도입 전의 문제

- Bedrock 에서도 모델 단위 사용량만 확인 가능
- AWS, Google Cloud, Azure 에 따로 신청하고 따로 관리

07-04  
프로젝트 시작

07-17  
SRE 실 내부  
파일럿

07-25  
지원자 파일럿

08-04  
전사 공개

### 공개 글에 없는 것

Virtual Key 발급 정책, 팀과 개인의 예산 한도, SSO 방식, 비용 절감액은 공개 글 세 편에서 확인되지 않음

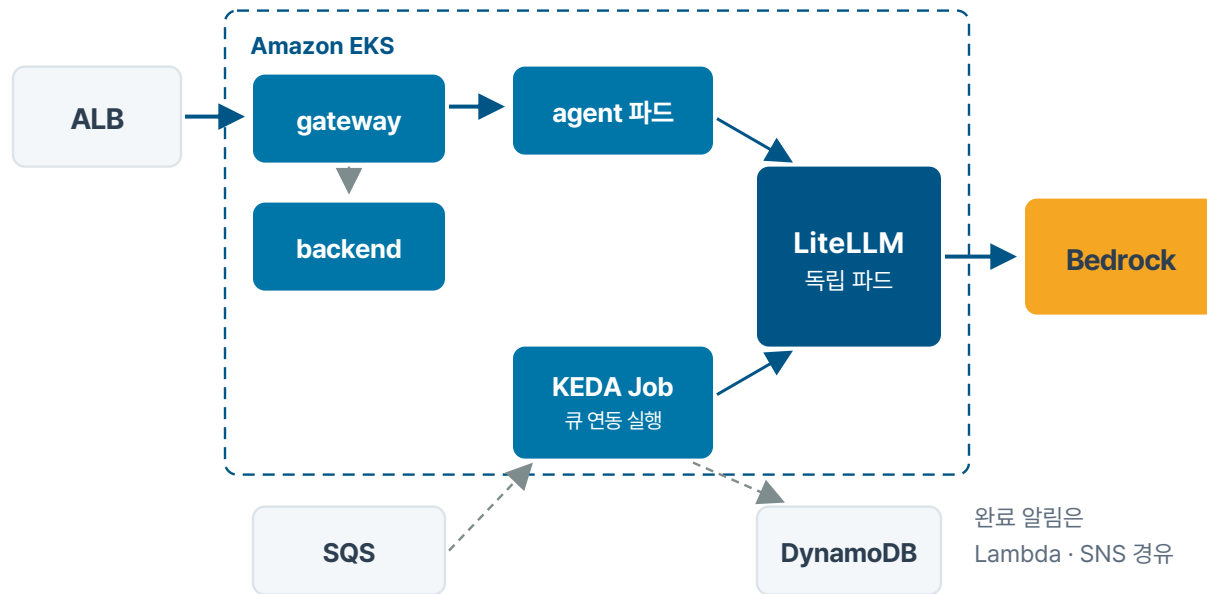


## KEY INSIGHT

### LLM 호출 계층을 핵심 로직과 분리함

— LiteLLM 스팸만 100% 저장하는 샘플링으로 토큰 추적과 모니터링 비용 절감을 함께 달성함

## 전표 작성 AI 에이전트의 실행 기반



- agent 파드와 Job 모두 LiteLLM 파드를 거쳐 Bedrock 호출
- LiteLLM 은 핵심 기능과 무관한 서비스로 분류해 분리
- Lambda, ECS, AgentCore 와 비교해 팀에 익숙한 EKS 선택

## tail sampling 기준

| 트레이스 대상                 | 보존 비율 | 이유                |
|-------------------------|-------|-------------------|
| LiteLLM 이면서 GenAI 속성 있음 | 100%  | 토큰과 모델 사용 현황 파악   |
| 상태가 Error 또는 UNSET      | 100%  | 장애 조사에 필요         |
| 그 밖의 일반 트레이스            | 5%    | 경향만 파악 (초기에는 70%) |

## 모니터링 경로



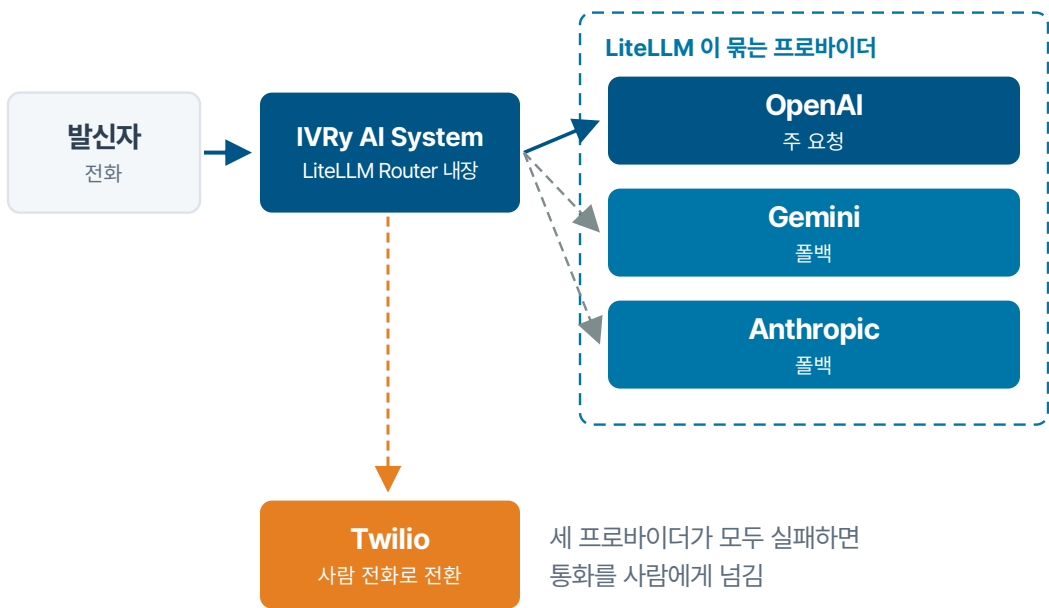
LiteLLM 이 생성한 스팸의 GenAI 속성이 샘플링 조건이 되고, CloudWatch GenAI Dashboard 에서 토큰 수와 모델 사용 현황으로 집계. 비용 확인 화면은 CloudWatch



## KEY INSIGHT

통화가 끊기는 서비스라 프로바이더 이중화가 필수임  
— 설정 몇 줄로 폴백이 적용됨

### 프로바이더 이중화와 마지막 대피 경로



통화 서비스 규모: 누적 계정 3만 개 이상, 누적 수신 통화 4,000만 건 이상

LLM 이 멈추면 통화 자체가 불가능해 안정성이 사할 문제

### Router 폴백 설정 (발표 장표의 코드, 일부 생략)

```
model_list = [
    {"model_name": "chatgpt", "litellm_params": {"model": "azure/gpt-4o"}},
    {"model_name": "gemini", "litellm_params": {"model": "gemini-1.5-flash-001"}},
]
fallbacks = [{"chatgpt": ["gemini"]}]]
router = litellm.router.Router(model_list=model_list, fallbacks=fallbacks,
    num_retries=0, # 타임아웃 외에는 즉시 폴백
    retry_policy=RetryPolicy(TimeoutErrorRetries=1))
```

### 🕒 도입 경과

- **2024-07-13**  
Azure OpenAI 가 여러 리전에서 정지, OpenAI 도 동시에 정지. 복구까지 약 하루
- **대응**  
감시 항목 확대와 폴백 도입. 폴백 라이브러리로 LiteLLM 채택
- **사용 형태**  
Python 라이브러리 Router 를 앱에 내장 (Proxy 서버 미사용)



## KEY INSIGHT

실제 장애는 오류 없는 지연과 출력 변화가 많음

— 업데이트 뒤 CPU 급증으로 IVRy 는 다른 라이브러리로 이전을 결정함

### 오류 없이 나빠지는 장애

#### • 지연 악화

평소 약 1초이던 지연이 p99 20초 근처까지 상승 (2025-09-19).

특정 입력 모달리티에서만 발생

#### • 출력 변화

같은 이름의 모델이 업데이트 뒤 같은 입력을 다르게 분류

명시적 오류가 없어 폴백 미발동



### LiteLLM 자체의 문제

#### • 업데이트 뒤 CPU 급증

새 모델을 쓰려고 올리자 20% 안팎이던 CPU 가 40~60%로 상승, 트래픽이 줄어도 유지

#### • 결정

버전을 바꿔 두 번 시도했으나 실패해 다른 라이브러리로 이전 결정

### 폴백 바깥을 메우는 세 가지 대책

1

#### 감시를 잘게 나누기

지연은 모델 종류, 입력 모달리티, 출력 토큰 길이에 좌우. 이 값이 크게 다른 작업은 따로 감시, 경보도 작업 유형별 발송

2

#### 장애 유형별 플레이백과 훈련

오류율 악화, 지연 악화, 정확도 저하 각각에 대해 사용자 영향, 탐지 방법, 조치 조건을 미리 작성. 훈련용 경보로 실제 대응 연습

3

#### 출력 일관성 검사

테스트 입력과 기대 출력을 정기 실행해 결과를 기록. 코드 병합 때마다 전화 E2E 자동 테스트 실행

### 검토한 대안 라이브러리

Pydantic AI

any-llm

AI SDK (TypeScript)

OpenRouter

최종 선택은 발표에 없음



KEY INSIGHT

한계는 기능보다 운영 쪽에 많음  
— 버그 추적과 호스팅 담당자가 없으면 유지하기 어려움

| 한계               | primeNumber 의 설명                     | 구체 내용                                       |
|------------------|--------------------------------------|---------------------------------------------|
| Enterprise 전용 기능 | 필요한 기능이 Enterprise 버전에 있음            | Organization(팀의 상위 개념), SSO/SCIM, Audit Log |
| 잦은 버그            | 수정은 빠르나 이슈를 계속 확인해야 함                | 캐시 토큰 비용 오계산, Bedrock Passthrough 인증 오류     |
| 호스팅 부담           | 테스트는 쉬우나 직접 운영하면 부담이 적지 않음           | ECS, RDS, 로드밸런서, 설정 저장소 운영                  |
| Terraform 관리     | 사용자와 팀을 코드로 관리하려면 커뮤니티 Provider 에 의존 | 공식 Provider 부재                              |
| 정액 구독            | Claude Max 는 Proxy 를 거치지 않음          | 게이트웨이의 통제·기록 대상 아님                          |

**primeNumber 의 정리**

만능 해결책은 아님. 본격 운영에는 상당한 구축·운영 부담이 따르고 버그와 기능 부족 때문에 이슈를 계속 확인해야 함

## SECTION 09

# 사례 구성 진단: Docker 기반 Proxy

컨테이너 한 대 구성의 별칭·폴백·키 관리 현황



KEY INSIGHT

Stateless 라 재배포는 쉽지만 클라이언트를 구분할 방법이 없음  
— 커뮤니티 이미지를 latest 태그로 써서 실행 버전도 운영자가 정하지 못함

9개

모델 별칭

3건

별칭 사이의 폴백

6종

프록시를 호출하는 클라이언트

1개

인증에 쓰는 마스터 키

| 항목          | 사례 구성                                   | 그에 따른 결과                                 |
|-------------|-----------------------------------------|------------------------------------------|
| 배포 형태       | 도커 컴포즈, 컨테이너 한 개                        | 운영 대상이 적고 재배포가 단순                        |
| 이미지         | LiteLLM 1.81 계열을 담은 커뮤니티 이미지, latest 태그 | 버전과 다이제스트를 고정하지 않아 공급망 관리 사각지대           |
| 자원과 재시작     | 메모리 2GB, CPU 1.5 코어, 자동 재시작             | 프록시는 추론을 하지 않아 GPU 불필요                   |
| 데이터베이스 · 캐시 | Postgres, Redis 모두 없음                   | Virtual Key, 예산, Rate Limit, 사용 기록 사용 불가 |
| 설정과 인증      | 읽기 전용 YAML 파일 한 개, 마스터 키 하나             | 변경 이력이 파일 하나로 모이지만 클라이언트 구분 불가           |
| 타임아웃        | 요청과 스트림 모두 30분                          | 약 26만 토큰 컨텍스트와 도구 호출 반복을 고려한 값           |
| 미지원 파라미터    | drop_params 로 요청에서 제거                   | 백엔드마다 다른 파라미터 차이를 처리                     |



## KEY INSIGHT

별칭 9개 중 8개가 자체 운영 백엔드에 연결됨  
— 별칭은 백엔드를 추상화할 뿐 GPU 서버와 모델 운영 부담은 그대로임

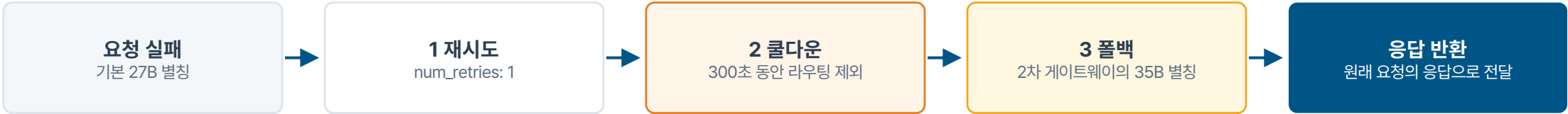




KEY INSIGHT

폴백 대상이 규모가 다른 모델이라 응답 품질과 속도가 달라짐  
— 폴백 발생을 모니터링하지 않으면 어느 모델이 응답했는지 알 수 없음

요청이 실패했을 때 프록시의 처리 순서



실패한 별칭

폴백 대상 별칭

기본 27B 별칭

35B 별칭 (Reasoning off)

27B Reasoning 별칭

35B Reasoning 별칭

2차 게이트웨이 경우 27B 우회 별칭

35B 별칭 (Reasoning off)

폴백 3건의 대상은 모두 쿠버네티스 2차 게이트웨이 뒤의 Qwen 계열 35B급 모델. 같은 모델의 복제본이 아니고 규모가 다른 모델

최근 사흘 로그에 폴백 · 쿨다운 기록 없음

그 기간에 vLLM 이 요청을 계속 처리했다는 사실일 뿐, 장애 때 기대대로 동작한다는 근거는 되지 못함

쿨다운 300초 동안 일어나는 일

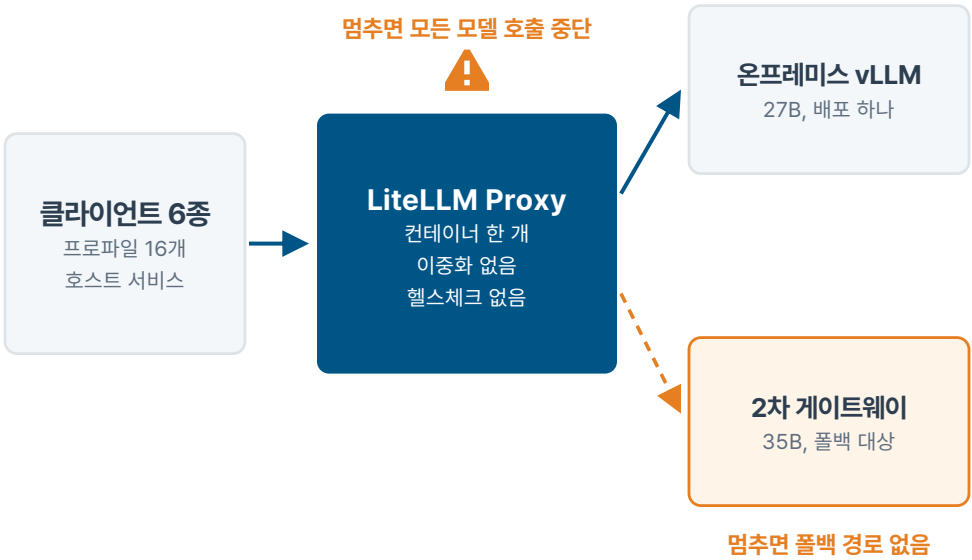
- 에이전트 전체의 트래픽이 35B 모델 하나로 집중  
기본 별칭이 프로파일 16개의 기본값
- 대화 한 번 안에서 응답 모델이 섞임  
일부 요청은 27B, 나머지는 폴백된 35B 가 처리
- 2차 게이트웨이가 멈추면 폴백 경로 전체가 사라짐  
타임아웃과 재시도, 키도 두 곳에서 따로 관리
- 쿨다운 시간의 기준  
추론 서버가 재시작해 모델을 다시 로딩하는 시간



KEY INSIGHT

폴백은 모델 서버 장애까지만 대응함  
— 호출 경로를 하나로 모았다면 프록시 자체의 이중화와 헬스체크가 필수임

장애 지점과 영향 범위



자동 재시작은 프로세스 종료만 복구. 응답하지 않는 프록시를 감지해 교체하는 기능은 없음

| 구성 요소     | 현재 상태                  | 장애 시 결과                        |
|-----------|------------------------|--------------------------------|
| 별칭당 배포    | 하나                     | 부하 분산 없음, 규모가 다른 모델로 폴백        |
| 폴백 대상     | 2차 게이트웨이의 35B 별칭       | 2차 게이트웨이가 멈추면 폴백 경로 없음         |
| 프록시 컨테이너  | 한 개, 자동 재시작            | 추론 서버가 정상이어도 모든 에이전트의 모델 호출 중단 |
| 폴백 미지정 별칭 | 경량, 지사, 임베딩, 클라우드, 35B | 백엔드 장애가 요청 실패로 이어짐             |

프레임워크 수준 폴백의 범위

콘텐츠 공장 스택은 기본 모델 호출이 실패하면 35B 별칭으로 다시 요청. 이 폴백도 같은 프록시를 거치므로 프록시 정지에는 효과 없음



# 모든 클라이언트가 마스터 키 하나로 Proxy 에 접속

## KEY INSIGHT

사흘간 요청 약 2,500건을 클라이언트별로 구분할 수 없음  
— 이 클라이언트 목록이 곧 발급할 Virtual Key 목록임



## 최근 사흘간 프록시가 받은 요청

|                    |          |
|--------------------|----------|
| Chat Completions   | 약 2,500건 |
| 임베딩                | 약 40건    |
| 모델 목록 조회           | 약 280건   |
| 잘못된 모델 이름 (400 오류) | 7건       |
| 키 없이 들어온 요청        | 11건      |

## 프록시를 거치지 않는 경로 3건

### AI 검색 인용 주간 점검 스크립트

직접 호출 → 클라우드 프로바이더 세 곳의 API

### 위키 정리 스크립트

직접 호출 → 클라우드 프로바이더 API

### 이미지 생성 백엔드 한 개

직접 호출 → 모델 중개 서비스



KEY INSIGHT

요건 8개 중 충족 1 · 부분 충족 4 · 미충족 3

— 미충족 3건은 오픈소스 버전 기능과 배포 방식으로 해결할 수 있어 제품 교체가 필요 없음

| 구분 | 요건 항목                   | 판정    | 사례 구성의 상태                                |
|----|-------------------------|-------|------------------------------------------|
| 기본 | OpenAI 호환 엔드포인트와 모델 별칭  | 부분 충족 | 별칭 9개가 한 엔드포인트로 통합됐으나 프록시 밖 경로 3건이 남음    |
| 기본 | 키 관리                    | 부분 충족 | 프로바이더 키는 프록시가 보관, 모든 클라이언트가 마스터 키 하나를 공유 |
| 기본 | 폴백과 안정성 설정              | 충족    | 재시도 1회, 쿨다운 300초, 폴백 3건                  |
| 기본 | 요청 · 응답 로깅              | 부분 충족 | 표준 출력 로그로 요청 · 오류 건수만 확인, 모델 별칭은 남지 않음   |
| 상세 | 클라이언트별 Virtual Key 와 예산 | 미충족   | 데이터베이스가 없어 사용 불가                         |
| 상세 | 지연 · TTFT 모니터링          | 미충족   | 지표 엔드포인트와 콜백이 비활성 상태                     |
| 상세 | 공급망 보안                  | 미충족   | 커뮤니티 이미지를 latest 태그로 사용                  |
| 상세 | LLM Gateway 가용성         | 부분 충족 | 자동 재시작은 있으나 인스턴스가 하나                     |

1

충족

설정 파일만으로 동작하는 폴백

4

부분 충족

경로와 키를 모았지만 구분 수단이 없음

3

미충족

데이터베이스, 콜백 설정, 이미지 고정이 필요

## SECTION 10

# 사례 구성 진단: 에이전트와 요청 흐름

에이전트 배치, 사용량 분포, 메시지 한 건의 요청 경로



# 컨테이너 22개 배치: 에이전트와 프록시의 네트워크 분리

10 · 사례 구성 진단: 에이전트와 요청 흐름

## KEY INSIGHT

프록시는 전용 네트워크에서 호스트 포트 하나로만 노출됨  
— 클라이언트가 늘어도 접근 통제와 모니터링 지점은 한 곳임





## KEY INSIGHT

에이전트를 늘려도 프록시 설정은 그대로임

— 다만 키가 컨테이너 환경변수에 있어 구조상 마스터 키를 공유하게 됨

### 프로파일별 전용 자원

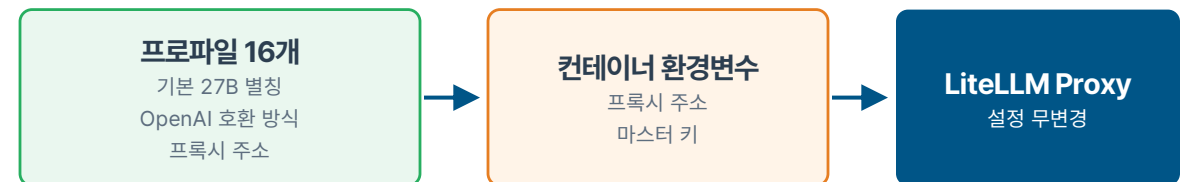
| 자원                 | 결정되는 내용             |
|--------------------|---------------------|
| 역할 문서              | 에이전트의 성격, 임무, 답변 방식 |
| 설정 파일              | 연결할 MCP 서버와 도구 구성   |
| 메신저 봇 토큰           | 사용자가 대화하는 채널        |
| 예약 작업 목록           | 정해진 시각에 실행할 업무      |
| 세션 DB(SQLite), 메모리 | 대화 기록과 장기 메모리       |

역할을 구분하는 요소는 역할 문서, 연결된 MCP 서버, 예약 작업 세 가지. 재무와 마케팅 에이전트는 같은 27B 모델을 같은 별칭으로 호출

### 모든 프로파일이 공유하는 자원

- 스킬 디렉터리와 보조 스크립트  
문서 읽기, 요약 같은 공용 기능
- 컨테이너 환경변수  
LiteLLM Proxy 주소와 키 — 같은 컨테이너의 프로파일은 모두 같은 키로 호출

### 16개 프로파일의 모델 호출 경로



### 프로파일마다 다른 MCP(Model Context Protocol) 연결의 예

인프라 점검  
APM, 위키

영업  
CRM, 위키

공공 데모  
법령, 공문서



# 업무 에이전트 사용량은 영업 에이전트에 집중됨

10 · 사례 구성 진단: 에이전트와 요청 흐름

## KEY INSIGHT

LLM 을 쓰는 예약 작업 48건의 호출량은 프록시에서 제한되지 않음  
— 사람이 지켜보지 않는 호출이라 예산과 Rate Limit 이 필요함

### 최근 14일 세션 수

단위: 건, 근사치

#### 영업 에이전트

대부분 예약 작업

약 340

#### 비서 에이전트

약 70

#### 주식 리포트 에이전트 A

약 45

#### 주식 리포트 에이전트 B

약 45

#### 마케팅 에이전트

약 30

#### 엔지니어링 에이전트

약 25

영업 에이전트 하나의 세션이 나머지 다섯의 합 (약 215)보다 많음

### 등록된 예약 작업 94건의 구성

48건

30건

16건

#### ● LLM 을 쓰는 작업

실행할 때마다 프록시 호출

#### ● 스크립트만 실행

프록시 호출 없음

#### ● 비활성 작업

프록시 호출 없음

부하를 추정할 때는 LLM 을 쓰는 48건만 계산

### LLM 사용 예약 작업의 에이전트별 건수

마케팅

13건

영업

9건

엔지니어링

6건

비서

5건

주식 리포트 B

5건

주식 리포트 A

4건

인프라 점검

3건

재무

2건

인사

1건

에이전트별 구분은 클라이언트별 Virtual Key 가 있어야 프록시에서 가능



KEY INSIGHT

발행 자격증명을 프로파일 하나에만 둔 방식을 프록시 키에도 적용할 수 있음  
— Virtual Key 를 나누면 유출 시 교체 범위가 에이전트 하나로 줄어듦

## 콘텐츠 공장 스택

생성과 발행의 분리

### 문서 에이전트

백서 생성, 칸반으로 작업 수신

자격증명 없음

### 블로그 에이전트

블로그 원고 생성

자격증명 없음

### 발행 에이전트

워드프레스 3곳과 위키에 발행

자격증명 있음

생성 단계에서 잘못된 내용이 만들어져도 외부로 직접 발행되지 않음. 작업은 칸반으로 전달하고 작업마다 독립 프로세스가 처리

## 에이전트 콘솔

읽기 전용 볼륨과 대시보드 API

- 두 스택의 데이터 볼륨을 읽기 전용으로 마운트
- 대화와 변경은 에이전트의 대시보드 API 를 거쳐 반영
- 콘솔이 프록시에 보내는 요청은 모델 목록 조회와 1토큰 점검 호출 두 가지
- 콘솔로만 대화하는 프로파일의 호출도 같은 경로와 같은 별칭으로 도착

## 스킬 마켓플레이스

보관과 설치의 분리

- 스킬과 프로파일을 보관하는 컨테이너 다섯 개
- 설치하는 호스트 설치 프로세스가 30초마다 요청을 확인해 에이전트 CLI 로 수행
- 새 프로파일은 컨테이너 환경변수의 프록시 주소와 키, 기본 27B 별칭을 그대로 사용
- 프록시 설정 파일과 추론 서버 구성은 무변경

세 구성 요소 모두 모델 호출은 에이전트 컨테이너에서 LiteLLM Proxy 로 가는 한 경로를 사용



## KEY INSIGHT

### 프록시는 요청 단위로 기록함

— 대화 한 번의 비용과 지연은 여러 요청의 합이라 트레이스가 없으면 묶어 볼 수 없음

#### 1 메시지 수신

##### 메신저 봇

음성이면 호스트 음성 인식 서버가  
먼저 텍스트로 변환

#### 2 프롬프트 구성

##### 프로파일의 게이트웨이 프로세스

역할 문서 · 스킬 · 메모리로  
프롬프트를 만들어 기본 별칭으로  
요청

#### 3 별칭 해석

##### LiteLLM Proxy

별칭을 온프레미스 vLLM 으로  
해석, 실패하면 재시도 · 쿨다운 ·  
폴백

#### 4 도구 실행

##### 에이전트 컨테이너

MCP 브리지 · 검색 · 웹 추출을  
실행하고 결과마다 프록시를 다시  
호출

#### 5 스트리밍 응답

##### 게이트웨이 프로세스

최종 응답을 메신저로 스트리밍 전달

도구 결과 하나마다 프록시 왕복 한 번 추가

#### 같은 경로로 모이는 시작점과 요청 수

- 예약 작업: 프로파일의 스케줄러가 실행, 2단계부터 같은 경로
- 콘솔 대화: 대시보드 API 를 거쳐 에이전트에 전달, 2단계부터 같은 경로
- 도구를 두 번 호출한 대화는 프록시에 요청 세 건으로 도착
- 스크립트 전용 예약 작업은 프록시에 기록을 남기지 않음

#### LLM Gateway 의 통제 범위는 모델 접근까지

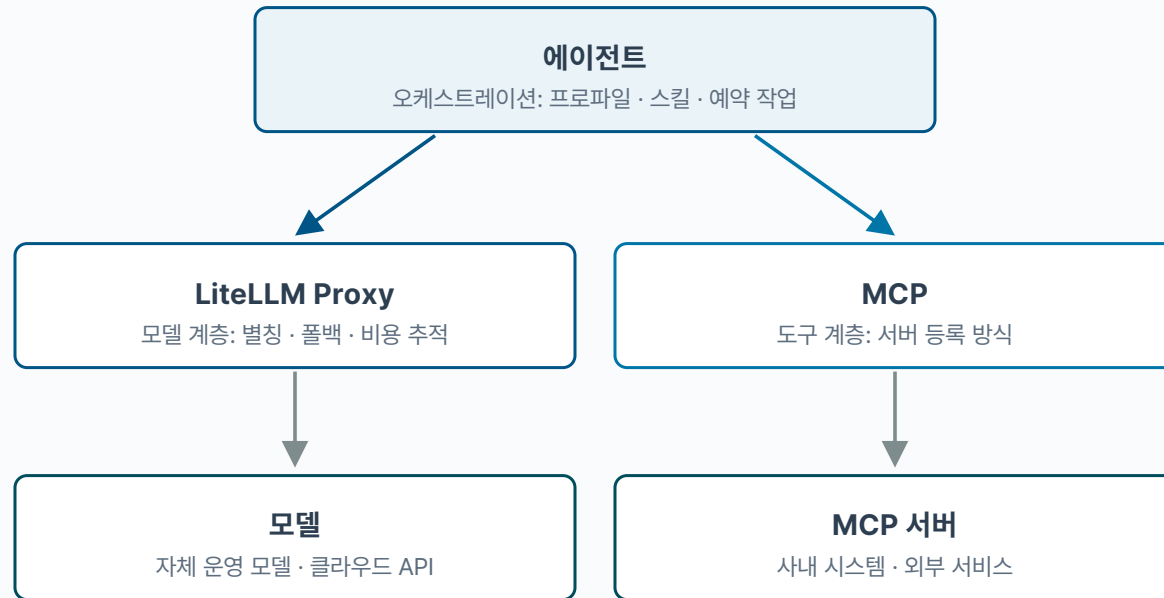
- 도구 호출 자체는 프록시를 거치지 않음
- 프록시는 요청 수와 토큰은 집계하지만 어떤 도구가 무엇을 요청했는지는 알지 못함
- 도구 쪽 권한은 프로파일마다 연결한 MCP 서버 구성에서 결정
- 검색과 웹 추출의 사용 기록은 각 도구 쪽에서 확인



## KEY INSIGHT

모델과 도구를 표준 인터페이스로 분리함  
— 한쪽을 교체해도 에이전트 코드는 수정하지 않음

에이전트 → LiteLLM(모델) / MCP(도구) 구조



에이전트는 두 표준의 클라이언트 역할만 수행  
모델 프로바이더와 도구에 직접 의존하지 않음

## 모델 교체

- LiteLLM 설정에서 별칭이 가리키는 모델만 수정
- 에이전트 코드와 프로파일은 그대로 유지

## 도구 추가

- MCP 서버를 새로 등록
- 한 번 등록한 서버를 여러 에이전트가 재사용

## 프로파일 단위 분리

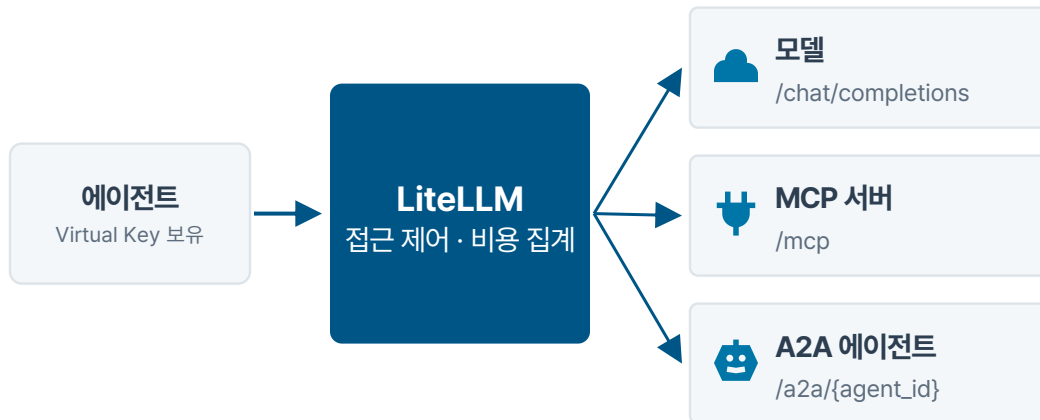
- 도구 접근 권한을 프로파일 선언 파일로 관리
- 선언 파일을 버전 관리해 변경 이력을 추적



## KEY INSIGHT

MCP 서버와 A2A 에이전트도 Virtual Key 로 접근을 제어함  
— 2026년 취약점이 이 엔드포인트에서 발생함

### Virtual Key 하나로 통제되는 세 종류의 호출



- MCP 서버는 Streamable HTTP, SSE, stdio 세 방식으로 등록, 키 · 팀 · 조직 단위 접근 제어
- A2A 는 Virtual Key 인증과 에이전트별 권한 검사, 헤더로 비용 집계
- Google ADK 는 Gemini 외 모델을 LiteLLM 으로 지원
- kagent 는 ModelConfig 의 baseUrl 교체만으로 프로바이더 변경 (게이트웨이 없이 모델을 바꾸는 예)



### MCP (Model Context Protocol)

에이전트와 도구를 잇는 규격. Anthropic 제안

- tools, resources, prompts, sampling 네 요소
- 연결 코드가 서버 설정으로 바뀌어 도구 추가가 쉬워짐
- Google 발표의 지적: 규격에 보안과 관측이 빠져 있음



### A2A (Agent2Agent)

에이전트끼리 잇는 규격. 2025년 4월 Google 발표, Linux Foundation 이관

- 에이전트 카드로 능력과 접속 방법 공개
- 작업, 산출물, 메시지를 JSON-RPC 2.0 으로 교환
- MCP 와 함께 쓰는 보완 관계

### 통제 지점이 곧 공격 지점

2026년 명령 주입 취약점(CVE-2026-42271)이 MCP 시험 엔드포인트에서 발생

## SECTION 11

# LiteLLM Proxy 모니터링

Prometheus 지표, TTFT, 로깅 연동, 사례 구성의 사각지대



**KEY INSIGHT** 폴백은 오류 없이 일어남 — 지표가 없으면 지금 어느 모델이 응답하는지 알 수 없음

|                 |                                                                                                                                        |                                |
|-----------------|----------------------------------------------------------------------------------------------------------------------------------------|--------------------------------|
| <b>비용 · 토큰</b>  | 누가 얼마나 쓰는가<br>spend_metric · total_tokens_metric<br>input_tokens_metric · output_tokens_metric                                         | 온프레미스 경로는 비용보다<br>토큰이 용량 판단 단위 |
| <b>요청 수</b>     | 얼마나 도착하고 얼마나 실패하는가<br>proxy_total_requests_metric · proxy_failed_requests_metric<br>in_flight_requests                                 | 시간대별 추세로 예약 작업이<br>집중되는 시각 구분  |
| <b>지연</b>       | 느려진 원인이 프록시인가 추론 서버인가<br>request_total_latency_metric · llm_api_latency_metric<br>overhead_latency_metric · request_queue_time_seconds | 어느 값이 커졌는지에 따라<br>조치 대상 결정     |
| <b>배포 상태</b>    | 백엔드가 정상으로 응답하는가<br>deployment_state · deployment_success_responses<br>deployment_failure_responses                                     | 응답 내용이 맞는지는<br>확인되지 않음         |
| <b>폴백 · 클다운</b> | 다른 모델이 대신 응답하고 있는가<br>deployment_successful_fallbacks<br>deployment_failed_fallbacks · deployment_cooled_down                          | 폴백 성공 증가는<br>기본 백엔드 점검         |

**먼저 알림을 설정할 대상**

**폴백 실패**  
대체 백엔드도 응답하지 못해  
클라이언트에 오류 전달

**클다운 발생과 TTFT 추세**  
오류 발생 전의 징후

**요청 단위와 대화 단위**

요청 수는 대화 수보다  
항상 크거나 같음

대화 단위 시간은  
에이전트 쪽에서 측정

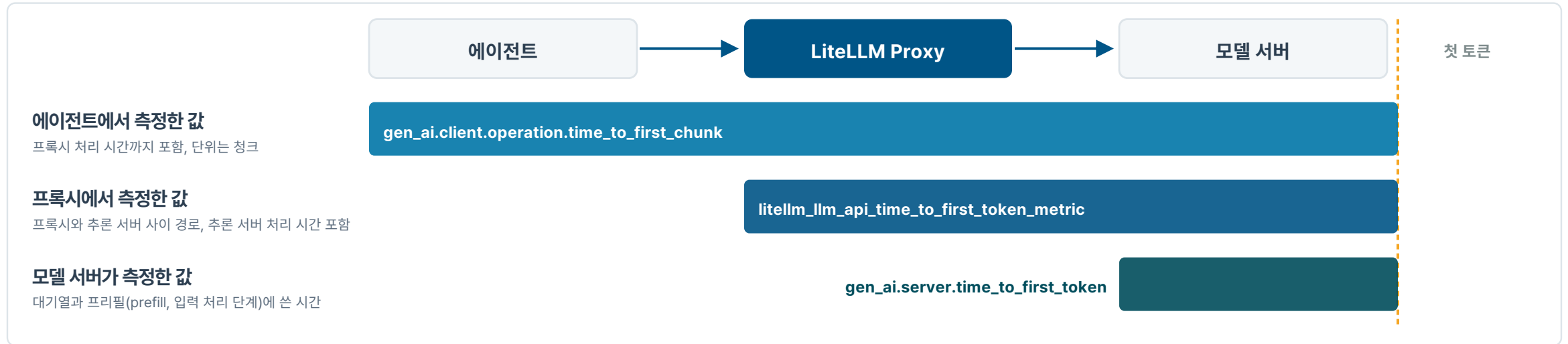
카드의 지표 이름은 앞의 litellm\_ 을 생략한 표기



# TTFT 는 측정 위치에 따라 값이 달라짐

## KEY INSIGHT

세 값의 차이를 보면 지연이 생긴 위치를 좁힐 수 있음  
— 한 그래프로 평균 내면 프록시와 추론 서버 중 어디를 조치할지 잘못 판단함



## 세 값을 함께 해석하는 방법

- **모델 서버 값은 그대로, 프록시 값만 증가**  
프록시와 추론 서버 사이의 경로 점검
- **프록시 값은 그대로, 에이전트 값만 증가**  
프록시 자체 지연과 대기열 시간 점검

## 대시보드 구성 시 유의점

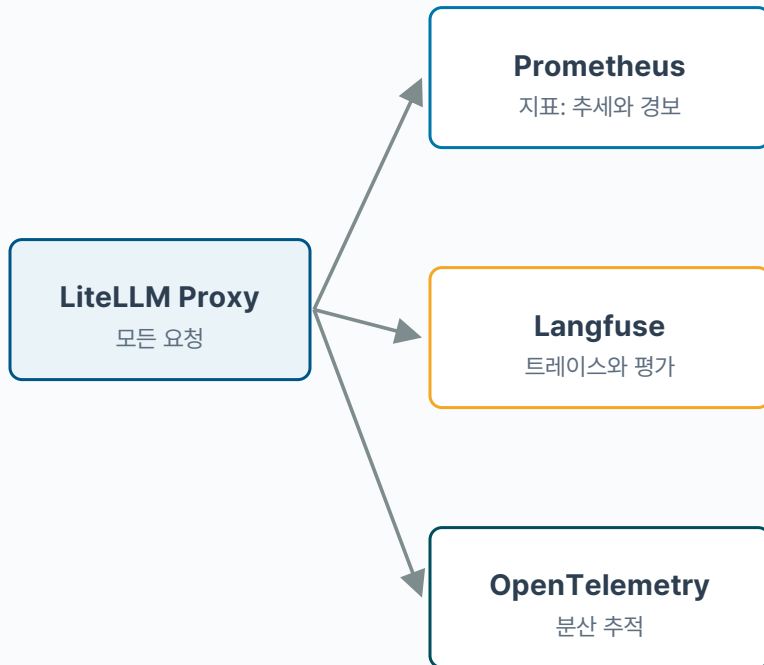
- OpenTelemetry GenAI 규격은 Development 단계라 지표 이름 변경 가능
- 에이전트와 프록시 양쪽의 토큰을 더하면 중복 집계, 비용 집계 기준으로 쓸 쪽을 먼저 결정
- 30분 타임아웃 구성에서는 수 분의 지연이 오류로 드러나지 않음



## KEY INSIGHT

지표는 추세를, 트레이스는 요청 한 건의 내용을 보여 줌  
— 콜백 설정만으로 Langfuse 와 OpenTelemetry 로 내보냄

Proxy 가 내보내는 세 가지 기록



## config.yaml 설정

litellm\_settings:

success\_callback: ["langfuse"]

callbacks: ["otel", "prometheus"]

turn\_off\_message\_logging: True # 본문 제외, 메타데이터만 기록

# 환경변수

LANGFUSE\_PUBLIC\_KEY · LANGFUSE\_SECRET\_KEY · LANGFUSE\_HOST

OTEL\_EXPORTER (otlp\_http · otlp\_grpc) · OTEL\_ENDPOINT · OTEL\_HEADERS

## 요청마다 기록되는 값

- 입력 · 출력 토큰 수와 비용
- 키 별칭, 사용자, 팀
- 캐시 적중 여부
- x-litellm-call-id 로 요청 추적

## 본문 기록 정책

- 기본값은 프롬프트와 응답 기록
- 개인정보가 있으면 본문을 제외
- OTEL 은 message\_logging 으로 따로 설정



## KEY INSIGHT

지연과 폴백 지표는 데이터베이스 없이 콜백 설정만으로 수집 가능  
— 클라이언트별 비용 지표만 Virtual Key 가 필요하므로 지표 수집을 미룰 이유가 없음

| 모니터링 항목      | 사례 구성     | 이유                     |
|--------------|-----------|------------------------|
| 요청 종류별 건수    | 로그에서 집계   | 표준 출력 로그에 요청이 남음       |
| 상태 코드별 오류 건수 | 로그에서 집계   | 표준 출력 로그에 오류가 남음       |
| 폴백 · 쿨다운 발생  | 로그 검색만 가능 | 지표 엔드포인트 비활성           |
| 모델 별칭별 요청 수  | 확인 불가     | 로그에 요청별 별칭이 없음         |
| 지연과 TTFT     | 확인 불가     | 지표 엔드포인트 비활성           |
| 토큰 사용량       | 확인 불가     | 지표 엔드포인트 비활성, 성공 콜백 없음 |
| 클라이언트별 사용량   | 확인 불가     | 마스터 키 하나를 공유           |

프록시를 거치지 않는 경로 3건은 지표를 활성화한 뒤에도 프록시 지표에 나타나지 않음. 모니터링 범위는 프록시를 경유하는 호출로 한정

## 콜백 한 줄로 수집하는 지표 (데이터베이스 불필요)

callbacks: ["prometheus"] → /metrics 경로로 노출

- 지연: 요청 전체, 모델 API 호출, 프록시 자체, 대기열 시간
- TTFT, 폴백 성공 · 실패, 쿨다운, 배포 상태
- v1.85.0 부터 /metrics 는 Bearer 토큰 인증 요구, 버전 고정과 함께 계획

## Postgres 와 Virtual Key 가 먼저 필요한 지표

Postgres 연결

클라이언트별 Virtual Key 발급

키 · 팀 · 사용자별 비용과 남은 예산

- 마스터 키를 공유한 채 콜백만 켜면 전체 합계만 표시
- 지표를 받을 Prometheus 서버가 운영 대상에 하나 추가



**KEY INSIGHT**

모니터링은 설정 변경으로 바로 시작할 수 있음

— 키 분리, 경로 이전, 이중화는 구조를 바꾸는 작업이라 순서를 정해야 함

| 한계                     | 결과                                       | 다음 단계                                   |
|------------------------|------------------------------------------|-----------------------------------------|
| 마스터 키 하나를 모든 클라이언트가 공유 | 클라이언트별 사용량을 나눌 수 없고 키가 유출되면 전부 교체        | 에이전트와 서비스별 Virtual Key 발급               |
| 데이터베이스 없음              | Virtual Key, 예산, Rate Limit, 사용 기록 사용 불가 | Postgres 연결                             |
| 지표 · 트레이스 수집 없음        | 지연, TTFT, 폴백 발생, 토큰 사용량 확인 불가            | Prometheus 지표 수집, OpenTelemetry 트레이스 연결 |
| 프록시를 거치지 않는 스크립트       | 해당 호출은 통제와 집계 대상에서 제외됨                   | 프록시 경유로 이전, 클라우드 키를 프록시로 집중             |
| 프록시가 단일 컨테이너           | 프록시가 멈추면 모든 에이전트 정지                      | 이중화와 헬스체크, 쿠버네티스 이전 검토                  |
| 별칭당 배포 하나              | 부하 분산 없음                                 | 같은 모델을 여러 배포로 구성                        |
| latest 이미지 태그          | 실행 버전을 운영자가 정하지 않음                       | 이미지 버전과 다이제스트 고정                        |

**설정 변경으로 바로 시작**

지표 수집, 이미지 버전과 다이제스트 고정

**구조를 바꾸는 작업**

Postgres 연결과 Virtual Key, 프록시 밖 경로 이전, 이중화와 여러 배포

## SECTION 12

# LLM Gateway 보안과 공급망 사고

2026년 침해 사고, 취약점, 점검 항목, 에이전트 위험



## KEY INSIGHT

게이트웨이에는 프로바이더 키가 모두 모임

— latest 태그는 실행 버전을 운영자가 정할 수 없으므로 바로 버전 고정으로 바뀌어야 함

1

### 침해 시작

CI/CD 파이프라인의  
Trivy 의존성 침해

2

### 악성 패키지 등록

1.82.7 과 1.82.8 을  
PyPI 에 직접 등록

3

### 약 40분 노출

2026-03-24 10:39(UTC)부터  
다운로드 가능한 상태

4

### PyPI 격리

두 버전 격리,  
공식 사고 보고 공개

## 영향 있음: 노출 시간에 PyPI 에서 설치한 환경

- 환경변수, SSH 키, AWS · GCP · Azure 자격증명, 쿠버네티스 토큰, 데이터베이스 비밀번호를 수집해 외부로 전송
- litellm\_init.pth 파일의 존재 여부로 감염 점검

## 영향 없음: 공식 도커 이미지로 운영한 환경

- 이미지가 의존성 버전을 고정해 침해된 패키지를 가져오지 않음
- 이 설명은 LiteLLM 프로젝트의 공식 이미지에 한정, 제3자가 다시 빌드한 이미지에는 그대로 적용 불가

| 항목           | latest 태그       | 버전 · 다이제스트 고정 |
|--------------|-----------------|---------------|
| 실행 버전을 정하는 쪽 | 이미지를 pull 한 시점  | 설정 파일에 적은 값   |
| 새 릴리스 반영     | 검토 없이 반영        | 릴리스 확인 뒤 반영   |
| 문제 발생 시 롤백   | 이전 버전을 따로 찾아야 함 | 직전 설정으로 복구    |

## 공식 권고와 사례 구성

- 권고: 시크릿 전부 교체, 파일 점검, 안전한 버전으로 고정
- 사례 구성: 커뮤니티 이미지와 latest 태그



KEY INSIGHT

Proxy 를 인터넷에 공개한 구성은 2026년 기준으로 위험함  
— 인증 전(pre-auth) 취약점이 공개 36시간 만에 공격받음

| 시점         | 사건                        | 내용                                          | 영향과 조치                                         |
|------------|---------------------------|---------------------------------------------|------------------------------------------------|
| 2026-03-24 | PyPI 공급망 침해               | CI 에서 유출된 배포 토큰으로 악성 버전 1.82.7, 1.82.8 게시   | 버전 고정 없이 pip 로 받은 환경만 영향. 공식 Docker 이미지는 영향 없음 |
| 2026-04    | CVE-2026-42208 (CVSS 9.3) | 인증 전 SQL 인젝션. Virtual Key 와 자격증명 테이블이 공격 대상 | 1.83.7 에서 수정. 공개 약 36시간 뒤 실제 공격 확인             |
| 2026-06    | CVE-2026-42271 (CVSS 8.7) | MCP 테스트 엔드포인트 명령 주입                         | 1.83.7 에서 수정. CISA 의 악용 확인 목록 등재               |



primeNumber 구성에 적용하면

- 공개 ALB 뒤의 Proxy 는 인증 전 취약점에 그대로 노출
- RDS 에 Virtual Key 와 프로바이더 설정이 모여 있어 침해 시 피해가 큼
- ECS 이미지를 latest 로 받으면 악성 버전 유입 가능



사고 후 BerriAI 의 조치

- 검증과 릴리스 저장소 분리, PyPI Trusted Publishing
- Docker 릴리스 태그 불변화, cosign 이미지 서명
- v1.85.0 부터 /metrics 에도 API 키 인증 기본



**KEY INSIGHT** 공식 Docker 이미지를 버전 고정으로 쓰던 환경은 3월 침해의 영향이 없었음

| 점검 항목      | 조치                                          | 근거                      |
|------------|---------------------------------------------|-------------------------|
| 설치 경로      | pip 직접 설치 대신 버전을 고정한 공식 Docker 이미지 사용       | 3월 침해에서 공식 이미지는 영향 없음   |
| 서명 검증      | cosign 으로 이미지 서명 확인                         | v1.83.0 부터 GHCR 이미지에 서명 |
| 새 버전 적용 유예 | uv exclude-newer 또는 pip 유예 옵션으로 3~7일 지연     | 악성 버전은 몇 시간 안에 제거됨      |
| 침해 흔적 점검   | site-packages 의 litellm_init.pth 와 설치 이력 확인 | 전이 의존성으로 설치된 경우 포함      |
| CI 의 자격증명  | 배포 토큰과 클라우드 키를 단기 발급으로 전환                   | LiteLLM 도 CI 토큰 유출로 침해됨 |

### Proxy 이미지와 의존성

- 버전을 고정한 이미지와 해시 고정 설치
- 새 버전은 유예 기간 뒤 반영
- 설치 스크립트 차단만으로는 import 시점의 악성 코드를 막지 못함

### Proxy 가 보유한 자격증명

- 프로바이더 키와 마스터 키가 한곳에 집중
- 침해 시 교체할 키 목록을 미리 작성
- 빌드 환경에 운영 키를 두지 않음

### CI 와 개발자 PC

- GitHub Actions 는 커밋 SHA 로 고정
- pull\_request\_target 사용 여부 점검
- 다른 패키지가 litellm 을 의존성으로 설치한 경우까지 점검



KEY INSIGHT

게이트웨이는 모든 프로바이더 키와 호출 본문이 지나는 지점임  
— 침해 한 번의 피해가 조직 전체의 LLM 사용으로 확대됨

| 통제 항목                                                                                     | 해야 할 일                                                      | 근거가 된 사실                       |
|-------------------------------------------------------------------------------------------|-------------------------------------------------------------|--------------------------------|
|  노출 범위   | 내부망에만 배치. 관리 UI 와 /metrics 는 인증 뒤에 두고 인터넷에 직접 공개하지 않기       | 원격 취약점 두 건이 모두 인증 전 공격         |
|  버전      | 이미지 태그 고정과 cosign 서명 검증. 새 Stable 버전은 유예 기간을 두고 적용          | 악성 버전은 게시 직후 몇 시간 안에 받은 환경만 피해 |
|  자격증명    | 프로바이더 키는 프로젝트 범위로 좁혀 발급하고 교체 주기 운영. 마스터 키와 솔트 키는 시크릿 저장소 보관 | 공격이 자격증명 테이블을 직접 겨냥            |
|  데이터베이스  | Virtual Key와 자격증명 테이블의 접근 제한. 지출 로그 보존 기간 지정                | DB 한 곳에 키와 설정이 모두 저장           |
|  로그 본문 | 본문 기록 여부를 개인정보 정책에 맞춰 결정. 필요하면 메타데이터만 기록                    | 모든 요청과 응답이 게이트웨이를 통과           |
|  패치    | 보안 권고 구독과 긴급 패치 적용 절차 사전 합의                                 | 권고 공개 뒤 약 36시간 만에 공격 관측        |



## KEY INSIGHT

계획과 실행을 모델이 맡는 구조가 원인임  
— 부작용이 생기는 지점마다 방어 수단이 달라야 함

### → 입력

- 사용자 지시
- 웹 페이지와 문서
- RAG 와 지식 베이스
- 도구 실행 결과

### 🤖 AI 에이전트

- LLM 엔진  
추론과 계획
- 오케스트레이터  
도구 선택과 실행 루프

### 🔧 도구

- 브라우저 조작
- 셸과 코드 실행
- HTTP 와 외부 API 호출
- 다른 앱과 서비스 조작

### 🌐 외부 영향

- 파일과 저장소 변경
- 외부 전송과 공개
- 과금과 구매
- 운영 시스템 조작

위험 3 · AI 에이전트

## 환각

그럴듯하지만 틀린 판단과 결과

위험 4 · AI 에이전트

## 토큰 사용량 폭증

반복과 재시도로 비용 급증

위험 2 · 도구

## 위험한 명령 실행

영향을 예측할 수 없는 명령 실행

위험 1 · 외부 영향

## 범위 이탈

대상이 아닌 시스템에 영향



## KEY INSIGHT

프롬프트 지시는 발생 빈도만 낮춤  
— 토큰 예산처럼 모델 밖에서 강제해야 과다 사용이 멈춤

### 범위 이탈

허용 대상 밖의 시스템까지 접근하고 조작

### 위험한 처리의 실행

실행해 보기 전에는 영향을 알 수 없는 명령

### 환각

불필요한 조작을 실행하거나 없는 결과를 보고

### 토큰 소비 폭주

같은 처리 반복, 끝없는 탐색, 재시도 중첩

## 두 종류의 통제

### 소프트한 통제

빈도를 낮추지만 보장하지 못함

- 프롬프트의 지시와 금지 사항
- 다른 모델의 행동 검토
- 모델 자체의 안전장치

### 하드한 통제

모델 바깥에서 강제

- 샌드박스 격리
- 접근 대상과 권한 제한
- 실행 전 승인, 토큰 상한

승인 흐름의 예 (kagent): 설정 변경과 리소스 삭제 전 사용자 승인 요청, 승인된 뒤에만 실행

## 토큰 상한을 단계로 거는 예 (Takumi)



**80%** 마무리 지시. 진행 중인 작업을 정리하도록 유도

**90%** 경고. 남은 예산 안에서 결과를 내도록 압박

**100%** 강제 제한. 도구 실행 금지, 결과 생성만 허용

### 게이트웨이와의 연결 (MSAP.ai 해석, 발표의 주장 아님)

키벌 예산과 RPM · TPM 한도는 에이전트 바깥에서 강제되는 하드한 통제의 한 층

## SECTION 13

# 쿠버네티스 배포와 이중화

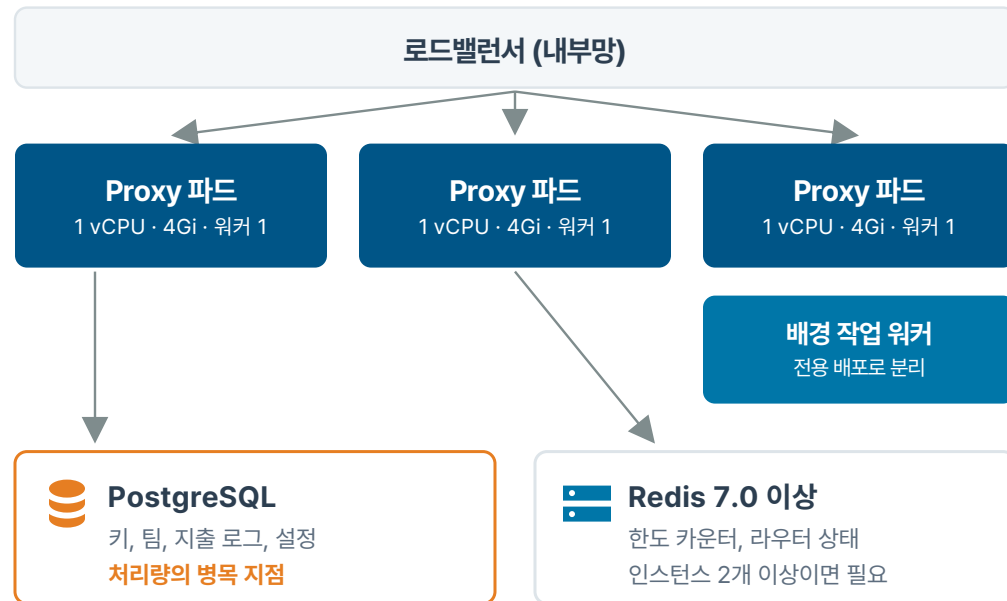
운영 배포 구성, Helm, Redis, Postgres 전환 절차



## KEY INSIGHT

병목은 Proxy CPU 보다 DB 연결 수 한도에서 먼저 생김  
— 인스턴스를 늘리기 전에 지출 로그 쓰기 방식을 정해야 함

### 운영 권장 구성



- CPU 60% 기준 수평 확장, 메모리는 파드 수명 동안 고정 유지
- LITELLM\_SALT\_KEY 는 모델 등록 뒤 변경 불가
- 배경 작업(예산 초기화, 로그 정리)은 전용 워커로 분리, 이미지 태그는 버전 고정

### 공식 벤치마크

|         | P95   | P99     | RPS   | 오버헤드    |
|---------|-------|---------|-------|---------|
| 인스턴스 4대 | 150ms | 240ms   | 1,170 | 2~8ms   |
| 인스턴스 2대 | 630ms | 1,200ms | 1,036 | 12~29ms |

인스턴스당 4 CPU · 8GB, 동시 사용자 1,000, Redis 없음. 운영 권장 사양과 조건이 다름

### 처리량

| 처리량         | DB 권장 사양       | 추가 조치             |
|-------------|----------------|-------------------|
| 1K RPS 이하   | 4 vCPU · 16GB  | 지출 로그 일괄 쓰기 (60초) |
| 1K ~ 5K RPS | 8 vCPU · 32GB  | Redis 트랜잭션 버퍼 사용  |
| 5K RPS 이상   | 16 vCPU · 64GB | 읽기 복제본 추가         |

### 병목이 생기는 자리

- CPU 포화보다 DB 연결 수 한도에 먼저 도달 (too many clients)
- 연결 풀 크기는 최대 연결 수를 인스턴스와 워커 수로 나눠 설정
- DB 가 멈춰도 Virtual Key는 최대 60초 캐시로 동작



**KEY INSIGHT** 헬스 체크 경로를 프로브에 연결해야 장애 파드가 트래픽에서 제외됨

## Helm 차트 설치

```
helm install litellm \
  oci://ghcr.io/berriai/litellm-helm \
  -f values.yaml
```

## 헬스 체크 프로브

```
livenessProbe:
  httpGet: {path: /health/liveliness, port: 4000}
  initialDelaySeconds: 120
  periodSeconds: 15
readinessProbe:
  httpGet: {path: /health/readiness, port: 4000}
```

## 파드 사양 권장값

- 파드당 1 vCPU · 4Gi, 워커 1개
- 부하는 파드 수를 늘려 처리

## 필수 환경변수

## 용도

|                           |                                          |
|---------------------------|------------------------------------------|
| <b>DATABASE_URL</b>       | PostgreSQL 연결. 인증과 비용 추적에 사용             |
| <b>LITELLM_MASTER_KEY</b> | 관리용 키. sk- 로 시작                          |
| <b>LITELLM_SALT_KEY</b>   | DB 에 저장하는 프로바이더 키를 암호화 모델을 등록한 뒤에는 변경 불가 |

## 배포할 때 확인할 점

- 차트와 이미지 모두 버전 태그를 고정
- 스키마 마이그레이션은 별도 Job 으로 먼저 실행  
(Proxy 에는 DISABLE\_SCHEMA\_UPDATE 설정)
- readiness 가 실패한 파드는 트래픽에서 제외됨
- 시작에 시간이 걸려 초기 지연을 120초로 지정

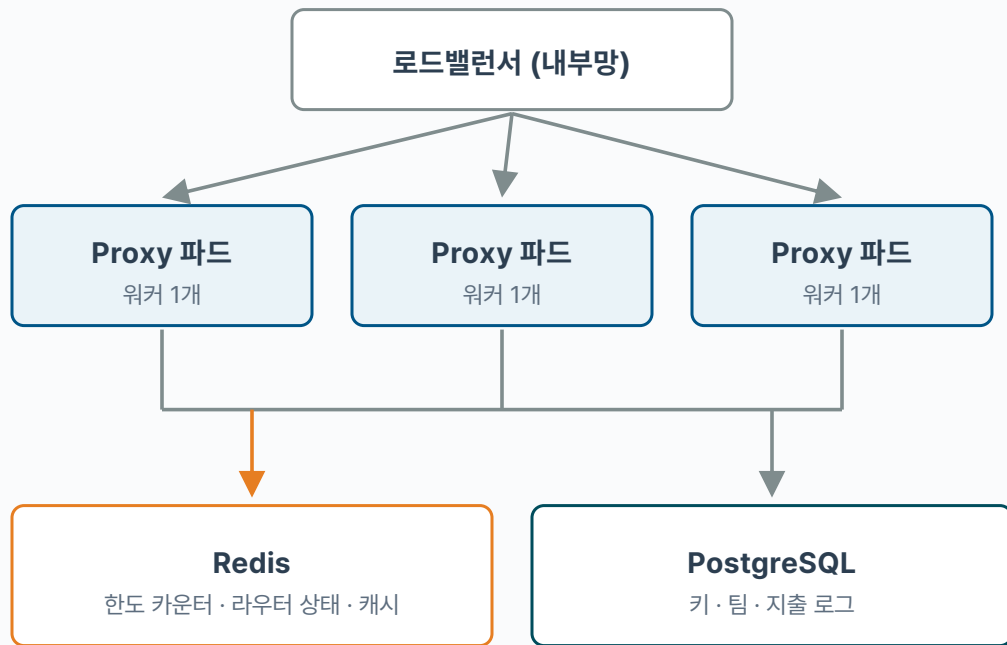


## KEY INSIGHT

Proxy 가 둘 이상이면 Redis 가 필요함

— Redis 가 없으면 RPM·TPM 한도가 인스턴스마다 따로 계산됨

Proxy 를 둘 이상 실행하는 구성



Redis 는 인스턴스가 둘 이상일 때부터 필요

## config.yaml 설정

```
router_settings:
  redis_host: os.environ/REDIS_HOST
  redis_port: os.environ/REDIS_PORT
  redis_password: os.environ/REDIS_PASSWORD
```

## Redis 가 없을 때

- 인스턴스마다 RPM · TPM 을 따로 계산
- 실제 허용량이 인스턴스 수만큼 늘어남
- 쿨다운 상태도 인스턴스마다 달라짐

## DB 부하와 장애 대비

- proxy\_batch\_write\_at: 지출 로그를 60초마다 일괄 기록
- database\_connection\_pool\_limit: 워커별 연결 수 제한
- allow\_requests\_on\_db\_unavailable: DB 장애 때도 요청을 처리 (내부망 전용 구성에서만 사용)



## KEY INSIGHT

마스터 키는 관리용으로만 남김

— 클라이언트마다 Virtual Key 를 발급해야 사용량이 구분됨

### 1 Postgres 연결

DATABASE\_URL 지정  
설정에 database\_url 추가

### 2 마스터 키 지정

sk- 로 시작하는 값  
관리 작업에만 사용

### 3 팀과 키 발급

/team/new 로 팀 생성  
/key/generate 로 키 발급

### 4 클라이언트 교체

호출이 많은 클라이언트부터  
Virtual Key 로 교체

### 5 결과 확인

/key/info 로 키별 지출 확인  
마스터 키 요청 수 감소 확인

#### Virtual Key 발급 요청

```
curl http://<Proxy 주소>:4000/key/generate \
-H "Authorization: Bearer <마스터 키>" \
-H "Content-Type: application/json" \
-d '{
  "models": ["chat-default"],
  "max_budget": 100, "duration": "30d",
  "rpm_limit": 1000, "tpm_limit": 100000,
  "team_id": "<팀 ID>", "key_alias": "<클라이언트 이름>"
}'
```

#### 키를 발급할 때 정하는 값

- 허용 모델과 최대 예산
- RPM · TPM 한도와 유효 기간
- 소속 팀과 키 별칭

#### 전환할 때 주의할 점

- LITELLM\_SALT\_KEY 는 처음에 정하고 바꾸지 않음
- 마스터 키는 전환이 끝난 뒤 교체
- Proxy 밖 직접 호출 경로도 함께 이전

## SECTION 14

# 도입 판단 기준과 실행 순서

LiteLLM Proxy 의 전제 조건, 결정 항목, 조치 순서



## KEY INSIGHT

보고된 한계는 기능 부족보다 운영 부담 쪽이 많음  
— 잦은 릴리스 검증과 자체 호스팅을 맡을 운영 담당이 있어야 함

### ✓ 적합한 경우

#### 사내 개발 도구와 자동화의 공용 Gateway

Claude Code, Codex, Dify 를 한 엔드포인트로 (벤고시닷컴, primeNumber)

#### 팀별 비용 통제와 모델 단계적 허용

Virtual Key 와 예산으로 과다 사용 차단

#### 에이전트 기반의 LLM 호출 계층 분리

핵심 로직과 분리해 독립 서비스로 운영 (RAKUS)

#### 로컬과 클라우드 모델 혼용

앱 수정 없이 설정으로 분기

### X 맞지 않는 경우

#### 자연에 민감한 단일 서비스에 라이브러리로 내장

업데이트 회귀(regression)가 서비스에 직접 영향 (IVRy 는 이전 결정)

#### 운영 담당 없이 인터넷에 노출한 배포

인증 전(pre-auth) 취약점이 공개 직후 공격받음

#### SSO 와 감사 로그가 필수인데 구독 예산이 없는 경우

해당 기능은 Enterprise 버전

#### 정액 구독 도구 위주의 사용 환경

Proxy 를 거치지 않아 통제 효과가 작음

### 전제 조건

매주 나오는 릴리스의 검증, 자체 호스팅 운영, 보안 권고 대응을 맡을 운영 담당이 있어야 위 적합 사례가 성립



## KEY INSIGHT

결정할 것은 호출 경로를 LiteLLM Proxy 한 곳으로 모으는 것 하나임  
— 키 분리와 지표 수집을 후속 단계로 정해 두지 않으면 관리 수준은 달라지지 않음

| 장점                                                         | 전제 조건                                                     |
|------------------------------------------------------------|-----------------------------------------------------------|
| OpenAI 형식 하나로 100개 이상 프로바이더 호출                             | 프로바이더마다 지원 파라미터가 달라 미지원 파라미터 처리 필요                        |
| Virtual Key, 비용 추적, 예산, 폴백, 로깅, Prometheus 지표가 오픈소스 버전에 포함 | Virtual Key 와 예산은 Postgres 필요, SSO 와 감사 로그는 Enterprise 버전 |
| MIT 라이선스로 상업적 이용 가능                                        | enterprise/ 디렉터리는 별도 라이선스                                 |
| 별칭 정의 변경만으로 모델 교체                                          | 모든 클라이언트가 프록시를 경유해야 함                                     |

### 맞지 않는 경우

- 파이썬 애플리케이션 하나가 여러 프로바이더를 호출하는 정도면 SDK 로 충분
- 이미 API 게이트웨이를 운영하면 확장형 후보가 운영 부담 면에서 유리
- 보안 규정이 SSO 나 감사 로그를 요구하면 Enterprise 버전 함께 검토

| 판단 기준    | 점검 항목                                     |
|----------|-------------------------------------------|
| 리스크 수용   | 프록시 정지가 전체 정지로 이어지는 구성을 이중화 전까지 감수할 수 있는가 |
| 리스크 수용   | 버전을 고정하고 릴리스를 확인할 담당자가 있는가                |
| 운영 부담    | Virtual Key 를 위해 Postgres 를 운영할 수 있는가     |
| 운영 부담    | 지표를 받을 Prometheus 수집 환경이 있는가              |
| 대안 대비 장점 | 필요한 기능이 오픈소스 버전 범위 안에 있는가                 |
| 대안 대비 장점 | 클라이언트가 둘 이상이고 키를 한 곳에서 관리해야 하는가           |

### 결정 하나와 후속 단계 둘

- 결정: 호출 경로를 LiteLLM Proxy 한 곳으로 통합
- 후속: 클라이언트별 Virtual Key 로 키 분리
- 후속: 지연 · TTFT · 폴백 지표 수집



## KEY INSIGHT

버전 정책과 노출 범위는 보안 사고 이력과 직접 연결됨  
— 내부망·소수 팀·버전 고정으로 시작하는 순서가 안전함

| 결정 항목      | 정해야 할 값                                  | 이 항목이 필요한 이유                        |
|------------|------------------------------------------|-------------------------------------|
| 1 사용 형태    | Proxy 공용 서비스로 돌지, SDK 를 앱에 내장할지          | 사례의 장단점이 형태에 따라 갈림 (IVRy 와 나머지 네 곳) |
| 2 버전 정책    | 고정할 버전, 새 Stable 버전의 유예 기간, 서명 검증, 갱신 주기 | 매주 릴리스, 3월 침해, 업데이트 뒤 CPU 급증 사례     |
| 3 노출 범위    | 내부망 한정 여부, 관리 UI 와 지표 엔드포인트의 접근 경로       | 인증 전 원격 취약점 두 건                     |
| 4 키 계층과 한도 | 팀과 사용자 구분, 기본 예산, RPM · TPM 값, 키 유효 기간   | 제품이 기본값을 주지 않음                      |
| 5 로그와 가드레일 | 본문 저장 여부, 보존 기간, 개인정보 가림 방식              | 스트리밍에서는 가림 불가                       |
| 6 장애 대응    | 폴백 순서, 지연 감시 기준, 전환을 판단할 담당              | 오류 없는 지연에는 폴백 미발동                   |
| 7 구독 여부    | SSO, 감사 로그, 키 자동 교체가 필요한지                | 해당 기능은 Enterprise 쪽                 |

## 시작 순서 제안

내부망 · 소수 팀 · 버전 고정으로 시작 → 예산과 로그 정책 확정 → 대상 팀과 도구 확대 (MSAP.ai 제안)



## 다음 단계의 실행 순서와 선행 조건

14 · 도입 판단 기준과 실행 순서

### KEY INSIGHT

버전 고정과 지표 수집은 조사와 함께 바로 시작함

— 지표 수집을 Virtual Key 보다 먼저 시작하면 키 분리 전후 사용량을 같은 지표로 비교할 수 있음

#### 선행 조건 없이 바로 시작하는 조치

##### 1 호출 경로 조사

클라이언트 6종과 프록시 밖 경로 3건 도출  
Virtual Key 발급 단위와 이전 대상 목록으로 사용

##### 2 이미지 버전 · 다이제스트 고정

공식 사고 보고가 직접 권고한 조치  
릴리스 확인 절차 하나가 추가됨

##### 3 Prometheus 지표 수집

지연, TTFT, 폴백, 쿨다운, 배포 상태  
모니터링 요건이 미충족 상태를 벗어남

#### 데이터베이스 연결 이후의 조치

##### 4 Postgres 연결

선행: 데이터베이스 운영 준비  
백업과 가용성이 프록시 운영의 일부가 됨

##### 5 Virtual Key 발급과 지표 분리

선행: Postgres, 조사 결과, 지표 수집  
키 위치를 컨테이너에서 프로파일 단위로 이동

##### 6 프록시 밖 경로 이전

선행: 조사 결과, Virtual Key  
외부 호출이 모두 별칭 목록으로 관리됨

##### 7 이중화와 헬스체크

클라이언트가 늘수록 필요성이 커짐  
여러 배포 구성, 트레이스 연결, 쿠버네티스 이전 검토

#### Virtual Key 를 나누는 우선순위

호출이 많은 영업 · 마케팅 에이전트와 클라우드 별칭을 쓰는 보조 스크립트를 먼저 분리, 점검 호출만 하는 콘솔은 후순위



## KEY INSIGHT

호출 경로를 하나로 모으는 결정이 먼저임 — 버전 고정과 지표 수집이 그다음 단계의 전제임

1

### 호출 경로 단일화

게이트웨이 단일  
경로 확정

인증·비용 통제의  
출발점

결정

2

### 버전 고정과 지표

1.82.7, 1.82.8 제외  
1.83.7 이상으로 고정

도입 전 호출 지표  
수집 선행

선행

3

### Virtual Key 분리

팀별·용도별 Virtual Key  
발급

사용량과 예산을  
키 단위로 구분

분리

4

### Postgres 연동

키와 사용 기록의  
영구 저장 구성

재시작 후에도  
기록 유지

연동

5

### 이중화 구성

게이트웨이 다중  
인스턴스 운영

단일 장애점의  
제거

구성

## 권고 순서의 핵심

경로 단일화를 먼저 확정해야 이후 네 단계의 설정 기준이 정해짐

Source: AWS 기술 블로그 (2026), 명지전문대학 정보관리팀 공지 (2026-04-15)

THANK YOU

# 감사합니다

LiteLLM 제품 소개

**MSAP.ai**

msap.ai

 <https://www.msap.ai/>

 02-6953-5427

 [hello@msap.ai](mailto:hello@msap.ai)