Day 8: LLM Gateway + 외부 API 연결 - 완전 가이드

> **GitHub Repository**: RAG-on-Local-CPU-minimal

> 실제 LLM 연결 - Gateway 패턴

한 줄 정의

> **"LLM Gateway는 기능이 아니라 책임 위치다. 모델을 바꾸는 것이 아니라 구조를 고정한다."**

학습 목표

Day 8의 핵심은 다음과 같습니다:

> **"외부 LLM API와 로컬 LLM을 Gateway로 통합하고, 실제로 연결한다"**

이 Day를 마치면:

- ☑ LLM Gateway 패턴 이해
- ☑ 외부 LLM API 연결 (OpenAI, Claude, Gemini, SOLAR)
- ☑ 로컬 LLM 통합
- ☑ API Key 안전한 관리
- ☑ 모델 교체 용이한 구조
- ☑ Token 사용량 추적
- ☑ 비용 추정
- ☑ 상태 모니터링

핵심: LLM을 교체 가능한 자원으로 다루기

프로젝트 구조

```
Rag_minimal_day8/
├─ docker-compose.yml     # Docker 실행
├─ config.yaml            # 설정 파일
├─ .gitignore             # Git 제외 파일
│
├─ gateway/               # LLM Gateway
│  ├─ main.py            # FastAPI 서버
│  ├─ router.py          # 라우팅 (핵심)
│  ├─ settings.py        # 설정 관리
│  ├─ usage.py           # 사용량 추적
│  ├─ metrics.py         # 메트릭
│  ├─ cost.py            # 비용 계산
│  ├─ logger.py          # 로깅
│  └─ llm/               # LLM 구현체
│     ├─ base.py         # 추상 클래스 + LLMResult
```

```
│     ├─ openai_llm.py   # OpenAI
│     ├─ claude_llm.py   # Claude (Anthropic)
│     ├─ gemini_llm.py   # Gemini (Google)
│     ├─ solar_llm.py    # SOLAR (Upstage)
│     └─ local_llm.py    # Local (llama.cpp)
│
├─ local_llm/            # 로컬 LLM 서버
│  ├─ Dockerfile
│  └─ run.sh
│
├─ ui/                   # 간단한 UI
│  ├─ index.html
│  ├─ app.js
│  └─ Dockerfile
│
├─ models/               # GGUF 모델 (Git 제외)
│  └─ tinyllama-1.1b-chat-v1.0.Q4_K_M.gguf
│
└─ secrets/              # API Keys (Git 제외)
   └─ api_keys.yaml
```

🤔 왜 이런 구조인가?

Day 7까지의 상태

RAG 파이프라인 완성:

```python
def run_pipeline(query):
    r = step_retrieve(query)
    rr = step_rerank(query, r)
    g = step_generate(query, rr)  # ← LLM Stub
    return PipelineTrace(...)
```

문제점:

```python
# step_generate에서
def generate(prompt):
    return "더미 응답"  # ← 실제 LLM 없음!
```

남은 질문:

- "이제 이걸 어떤 모델로 돌려야 하지?"
- "로컬이랑 외부 LLM은 어떻게 같이 쓰지?"
- "비용이 얼마나 나오는지는 누가 보지?"

Day 8에서 추가되는 것

☑ LLM Gateway

```
# 통합된 인터페이스
gateway = LLMGateway()

# 모델만 바꾸면 됨
response = gateway.generate(prompt, model="openai")
response = gateway.generate(prompt, model="claude")
response = gateway.generate(prompt, model="local")
```

개선점:

- ☑ 모델 교체 용이
- ☑ 사용량 추적
- ☑ 비용 추정
- ☑ 상태 모니터링

1. 왜 Gateway가 필요한가?

문제 상황 (Gateway가 없을 때):

```
# ❌ LLM 호출이 흩어진 코드
def step_generate_openai(query, contexts):
    import openai
    response = openai.chat.completions.create(...)
    return response

def step_generate_claude(query, contexts):
    import anthropic
    response = anthropic.messages.create(...)
    return response

# 모델 바꿀 때마다 함수 교체
if model == "openai":
    result = step_generate_openai(query, contexts)
elif model == "claude":
    result = step_generate_claude(query, contexts)
```

Gateway 도입:

```
# ☑ Gateway로 통합
class LLMGateway:
    def generate(self, prompt, model="openai"):
        # 1. 라우팅
        llm = self.router.route(model)

        # 2. 호출
        response = llm.generate(prompt)
```

```
        # 3. 사용량 기록
        self.usage.record(model, response.usage)

        # 4. 비용 계산
        cost = self.cost.calculate(model, response.usage)

        return response
```

2. 왜 LLMResult 객체를 사용하나?

문제 (dict 사용):

```
# ❌ dict는 타입 체크 불가
return {
    "text": "...",
    "usage": {...}
}

# 오타 발생 가능
result["txt"]  # KeyError!
```

해결 (LLMResult 사용):

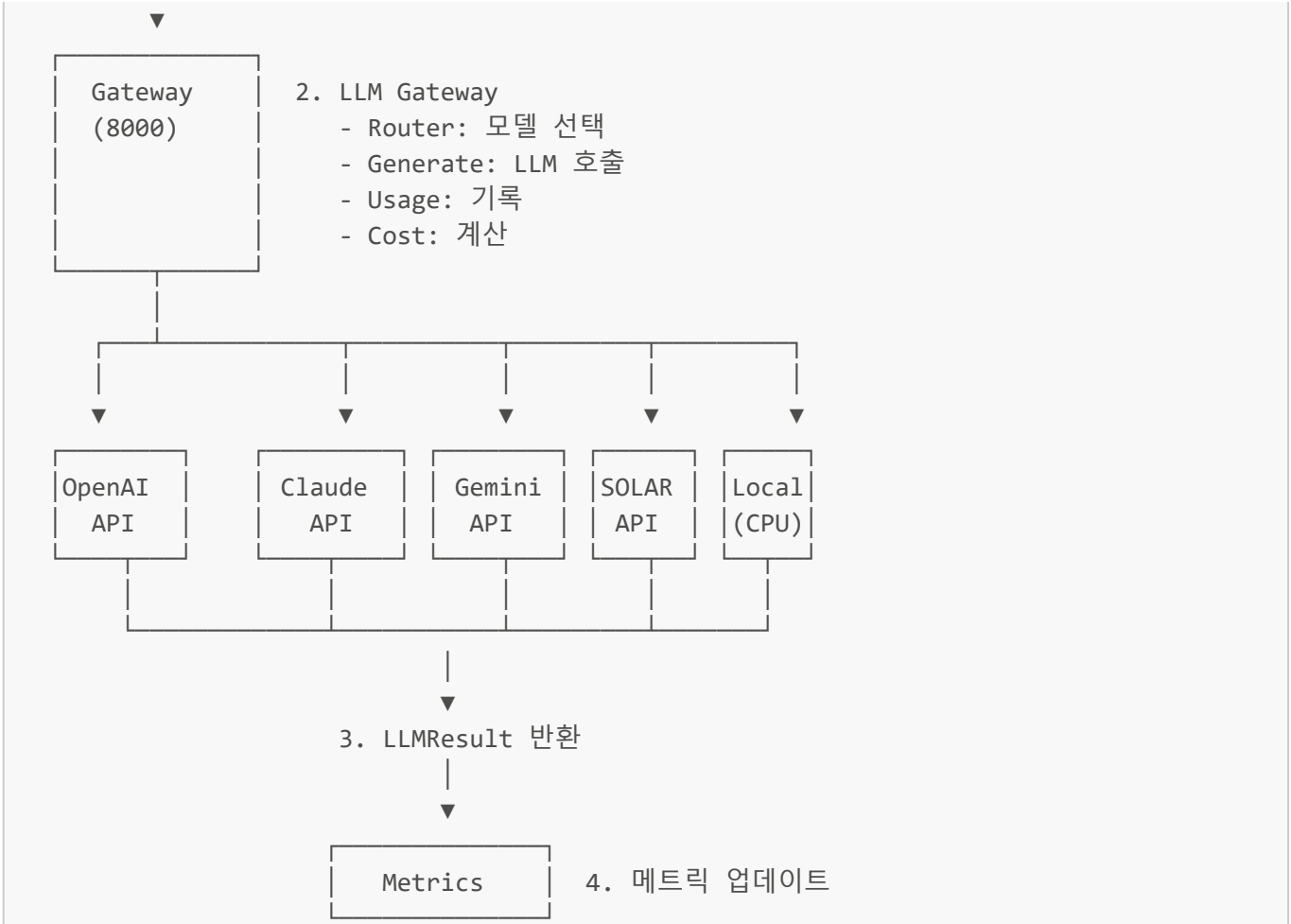
```
# ☑ dataclass로 타입 안전
@dataclass
class LLMResult:
    text: str
    usage: Optional[Dict[str, int]] = None

return LLMResult(
    text="...",
    usage={...}
)

# IDE 자동완성, 타입 체크
result.text  # ☑
result.txt   # ❌ 컴파일 에러
```

🔄 실행 흐름도

```
┌──────────────┐
│   UI         │  1. 모델 선택 + 질문
│ (3000)       │
└──────┬───────┘
       │
```



⚙ 설치 및 실행

1. API Key 설정

secrets/api_keys.yaml 생성:

```yaml
openai:
  api_key: "sk-proj-..."

claude:
  api_key: "sk-ant-api03-..."

gemini:
  api_key: "AIza..."

solar:
  api_key: "upstage-..."
```

주의:

- ⚠ `secrets/` 폴더는 `.gitignore`에 포함됨
- ⚠ API Key는 **절대** Git에 올리지 마세요!

2. config.yaml 설정

```yaml
llm:
  default: solar  # 기본 모델

  providers:
    openai:
      model: gpt-4o-mini
      temperature: 0.7
      max_tokens: 500

    claude:
      model: claude-3-sonnet-20240229
      temperature: 0.7
      max_tokens: 500

    gemini:
      model: gemini-1.5-flash
      temperature: 0.7
      max_tokens: 500

    solar:
      model: solar-1-mini-chat
      temperature: 0.7
      max_tokens: 500

    local:
      endpoint: http://local_llm:11434
      temperature: 0.7
      max_tokens: 500
```

3. Docker Compose 실행

Windows (PowerShell):

```
cd Rag_minimal_day8
docker compose up --build
```

macOS/Linux:

```
cd Rag_minimal_day8
docker compose up --build
```

4. 접속

UI:

```
http://localhost:3000
```

Gateway API:

```
http://localhost:8000/docs
```

Status:

```
http://localhost:8000/status
```

🧠 핵심 개념 Deep Dive

1. base.py (핵심 기반)

```python
# llm/base.py
from __future__ import annotations

from dataclasses import dataclass
from typing import Dict, Optional


@dataclass
class LLMResult:
    """
    LLM 응답 결과를 담는 데이터 클래스

    Attributes:
        text: 생성된 텍스트
        usage: 토큰 사용량 정보 (선택적)
               {
                   "prompt_tokens": int,
                   "completion_tokens": int,
                   "total_tokens": int
               }
    """
    text: str
    # Some providers return usage (prompt/completion/total tokens)
    usage: Optional[Dict[str, int]] = None


class BaseLLM:
    """
    모든 LLM 구현체의 기본 클래스
    """
    name: str  # "openai", "claude", "gemini", "solar", "local"
```

```
    def generate(
        self,
        prompt: str,
        max_tokens: int = 512,
        temperature: float = 0.2
    ) -> LLMResult:
        """
        프롬프트를 받아 응답 생성

        Args:
            prompt: 사용자 질문
            max_tokens: 최대 생성 토큰 수
            temperature: 생성 온도 (0.0 ~ 1.0)

        Returns:
            LLMResult: 생성 결과

        Raises:
            NotImplementedError: 하위 클래스에서 구현 필요
        """
        raise NotImplementedError
```

핵심 포인트:

- ☑ `@dataclass`: 자동으로 `__init__`, `__repr__` 생성
- ☑ `Optional[Dict[str, int]]`: usage는 선택적 (Local LLM은 None)
- ☑ `name` 클래스 변수: 각 LLM 식별
- ☑ `BaseLLM`: 추상 클래스, 모든 LLM이 상속

2. settings.py (설정 관리)

```
# gateway/settings.py
import yaml
from pathlib import Path

# Docker 컨테이너에서는 /app이 BASE_DIR
BASE_DIR = Path(__file__).resolve().parent.parent

def load_config():
    """config.yaml 로드"""
    config_path = BASE_DIR / "config.yaml"
    if not config_path.exists():
        raise FileNotFoundError(f"config.yaml not found at {config_path}")

    with open(config_path, "r", encoding="utf-8") as f:
        return yaml.safe_load(f)


def load_api_keys():
    """secrets/api_keys.yaml 로드"""
```

```
    secrets_path = BASE_DIR / "secrets" / "api_keys.yaml"
    if not secrets_path.exists():
        raise FileNotFoundError(
            f"api_keys.yaml not found at {secrets_path}"
        )

    with open(secrets_path, "r", encoding="utf-8") as f:
        return yaml.safe_load(f)


# 모듈 import 시 자동 로드
CONFIG = load_config()
API_KEYS = load_api_keys()
```

핵심 포인트:

- ☑ `BASE_DIR`: 프로젝트 루트 경로
- ☑ `CONFIG`, `API_KEYS`: 모듈 레벨 변수 (import 시 자동 로드)
- ☑ 파일 없으면 명확한 에러 메시지

⚠ 중요:

```
# 마지막에 반드시 추가!
CONFIG = load_config()
API_KEYS = load_api_keys()

# 이래야 다른 파일에서 import 가능
# from gateway.settings import API_KEYS
```

3. OpenAI LLM 구현

```
# llm/openai_llm.py
from __future__ import annotations

import requests

from .base import BaseLLM, LLMResult
from gateway.settings import API_KEYS


class OpenAILLM(BaseLLM):
    name = "openai"

    def __init__(self, model: str):
        self.model = model
        self.api_key = (
            API_KEYS.get("openai", {})
            .get("api_key", "")
            .strip()
```

```
        )

    def generate(
        self,
        prompt: str,
        max_tokens: int = 512,
        temperature: float = 0.2
    ) -> LLMResult:
        if not self.api_key:
            raise RuntimeError("OPENAI_API_KEY is not set")

        url = "https://api.openai.com/v1/chat/completions"
        headers = {
            "Authorization": f"Bearer {self.api_key}",
            "Content-Type": "application/json",
        }
        payload = {
            "model": self.model,
            "messages": [
                {"role": "user", "content": prompt}
            ],
            "temperature": temperature,
            "max_tokens": max_tokens,
        }

        r = requests.post(
            url,
            headers=headers,
            json=payload,
            timeout=60
        )
        if r.status_code >= 400:
            raise RuntimeError(
                f"OpenAI API error {r.status_code}: {r.text}"
            )

        data = r.json()
        text = data["choices"][0]["message"]["content"]
        usage = data.get("usage") or {}

        return LLMResult(
            text=text,
            usage={
                "prompt_tokens": int(usage.get("prompt_tokens", 0)),
                "completion_tokens": int(usage.get("completion_tokens", 0)),
                "total_tokens": int(usage.get("total_tokens", 0)),
            },
        )
```

핵심 포인트:

- ☑ `from gateway.settings import API_KEYS`: 설정에서 API Key 로드

- ☑ `.strip()`: 공백 제거
- ☑ `timeout=60`: 60초 타임아웃
- ☑ `raise RuntimeError`: 에러 발생 시 명확한 메시지
- ☑ `LLMResult` 반환: dict가 아님!

4. Claude LLM 구현

```
# llm/claude_llm.py
from __future__ import annotations

import requests

from .base import BaseLLM, LLMResult
from gateway.settings import API_KEYS


class ClaudeLLM(BaseLLM):
    name = "claude"

    def __init__(self, model: str):
        self.model = model
        self.api_key = (
            API_KEYS.get("claude", {})
            .get("api_key", "")
            .strip()
        )

    def generate(
        self,
        prompt: str,
        max_tokens: int = 512,
        temperature: float = 0.2
    ) -> LLMResult:
        if not self.api_key:
            raise RuntimeError("CLAUDE_API_KEY is not set")

        url = "https://api.anthropic.com/v1/messages"
        headers = {
            "x-api-key": self.api_key,
            "anthropic-version": "2023-06-01",
            "content-type": "application/json",
        }
        payload = {
            "model": self.model,
            "max_tokens": int(max_tokens),
            "temperature": float(temperature),
            "messages": [
                {"role": "user", "content": prompt}
            ],
        }

        r = requests.post(
```

```
            url,
            headers=headers,
            json=payload,
            timeout=60
        )
        if r.status_code >= 400:
            raise RuntimeError(
                f"Anthropic API error {r.status_code}: {r.text}"
            )

        data = r.json()

        # content is a list of blocks
        text = ""
        try:
            blocks = data.get("content") or []
            text = "".join(
                b.get("text", "")
                for b in blocks
                if b.get("type") == "text"
            )
        except Exception:
            text = str(data)

        usage = data.get("usage") or {}
        input_tokens = int(usage.get("input_tokens", 0))
        output_tokens = int(usage.get("output_tokens", 0))

        return LLMResult(
            text=text,
            usage={
                "prompt_tokens": input_tokens,
                "completion_tokens": output_tokens,
                "total_tokens": input_tokens + output_tokens,
            },
        )
```

핵심 포인트:

- ☑ `x-api-key` 헤더: Claude 전용
- ☑ `anthropic-version`: API 버전 명시
- ☑ `content` 블록 처리: Claude는 list 형태
- ☑ `input_tokens`, `output_tokens`: Claude 필드명

5. Gemini LLM 구현

```
# llm/gemini_llm.py
from __future__ import annotations

import requests
```

```python
from .base import BaseLLM, LLMResult
from gateway.settings import API_KEYS


class GeminiLLM(BaseLLM):
    name = "gemini"

    def __init__(self, model: str):
        self.model = model
        self.api_key = (
            API_KEYS.get("gemini", {})
            .get("api_key", "")
            .strip()
        )

    def generate(
        self,
        prompt: str,
        max_tokens: int = 512,
        temperature: float = 0.2
    ) -> LLMResult:
        if not self.api_key:
            raise RuntimeError("GEMINI_API_KEY is not set")

        url = (
            "https://generativelanguage.googleapis.com/v1beta/"
            f"models/{self.model}:generateContent"
        )
        params = {"key": self.api_key}
        payload = {
            "contents": [
                {
                    "role": "user",
                    "parts": [{"text": prompt}],
                }
            ],
            "generationConfig": {
                "maxOutputTokens": int(max_tokens),
                "temperature": float(temperature),
            },
        }

        resp = requests.post(
            url,
            params=params,
            json=payload,
            timeout=60,
        )
        resp.raise_for_status()
        data = resp.json()

        text = ""
        try:
            text = (
```

```
                data["candidates"][0]
                ["content"]["parts"][0]["text"]
            )
        except Exception:
            text = str(data)

        # Gemini usage fields may vary
        usage = None
        meta = data.get("usageMetadata") or {}
        if meta:
            usage = {
                "prompt_tokens": int(
                    meta.get("promptTokenCount", 0)
                ),
                "completion_tokens": int(
                    meta.get("candidatesTokenCount", 0)
                ),
                "total_tokens": int(
                    meta.get("totalTokenCount", 0)
                ),
            }

        return LLMResult(
            text=text,
            usage=usage,  # None일 수 있음
        )
```

핵심 포인트:

- ☑ URL에 `models/{model}:generateContent` 형태
- ☑ `params={"key": api_key}`: Query param으로 전달
- ☑ `usageMetadata`: Gemini 필드명
- ☑ `usage=None` 가능: 제공 안할 수도 있음

6. SOLAR LLM 구현

```
# llm/solar_llm.py
from __future__ import annotations

import requests

from .base import BaseLLM, LLMResult
from gateway.settings import API_KEYS


class SolarLLM(BaseLLM):
    name = "solar"

    def __init__(
        self,
        model: str,
```

```
        base_url: str = "https://api.upstage.ai/v1",
    ):
        self.model = model
        self.base_url = base_url.rstrip("/")
        self.api_key = (
            API_KEYS.get("solar", {})
            .get("api_key", "")
            .strip()
        )

    def generate(
        self,
        prompt: str,
        max_tokens: int = 512,
        temperature: float = 0.2,
    ) -> LLMResult:
        if not self.api_key:
            raise RuntimeError("UPSTAGE_API_KEY is not set")

        url = f"{self.base_url}/chat/completions"
        headers = {
            "Authorization": f"Bearer {self.api_key}",
            "Content-Type": "application/json",
        }
        payload = {
            "model": self.model,
            "messages": [
                {"role": "user", "content": prompt}
            ],
            "temperature": float(temperature),
            "max_tokens": int(max_tokens),
        }

        r = requests.post(
            url,
            headers=headers,
            json=payload,
            timeout=60,
        )
        if r.status_code >= 400:
            raise RuntimeError(
                f"Upstage API error {r.status_code}: {r.text}"
            )

        data = r.json()
        text = data["choices"][0]["message"]["content"]
        usage = data.get("usage") or {}

        return LLMResult(
            text=text,
            usage={
                "prompt_tokens": int(
                    usage.get("prompt_tokens", 0)
                ),
```

```
                "completion_tokens": int(
                    usage.get("completion_tokens", 0)
                ),
                "total_tokens": int(
                    usage.get("total_tokens", 0)
                ),
            },
        )
```

핵심 포인트:

- ☑ `base_url`: Upstage API 엔드포인트
- ☑ OpenAI 호환 API 형태
- ☑ 한글 특화 LLM

7. Local LLM 구현

```
# llm/local_llm.py
from __future__ import annotations

import requests

from .base import BaseLLM, LLMResult


class LocalLLM(BaseLLM):
    """
    Local CPU LLM adapter.

    This class acts as a thin HTTP adapter to a running llama.cpp server.
    No API key is required.

    Expected server:
    - llama.cpp with --server enabled
    - Classic endpoint: POST /completion

    Example endpoint:
    - http://local-llm:8080
    """

    name = "local"

    def __init__(self, endpoint: str):
        # e.g. http://local-llm:8080
        self.endpoint = endpoint.rstrip("/")

    def generate(
        self,
        prompt: str,
        max_tokens: int = 512,
        temperature: float = 0.2,
```

```
    ) -> LLMResult:
        url = f"{self.endpoint}/completion"
        payload = {
            "prompt": prompt,
            "n_predict": int(max_tokens),  # llama.cpp 파라미터
            "temperature": float(temperature),
        }

        r = requests.post(
            url,
            json=payload,
            timeout=60,
        )
        if r.status_code >= 400:
            raise RuntimeError(
                f"Local LLM error {r.status_code}: {r.text}\n"
                f"Hint: ensure llama.cpp server is running "
                f"and supports POST /completion"
            )

        data = r.json()

        # Known response shapes:
        # 1) {"content": "..."}
        # 2) {"completion": "..."}
        text = (
            data.get("content")
            or data.get("completion")
            or ""
        )

        return LLMResult(
            text=text,
            usage=None,  # llama.cpp does not expose token usage
        )
```

핵심 포인트:

- ☑ API Key 불필요
- ☑ `n_predict`: llama.cpp 파라미터 (max_tokens 대신)
- ☑ `usage=None`: llama.cpp는 토큰 정보 제공 안함
- ☑ `content` 또는 `completion` 필드 처리

📖 코드 Deep Dive: Router

```
# router.py
from llm.openai_llm import OpenAILLM
from llm.claude_llm import ClaudeLLM
from llm.gemini_llm import GeminiLLM
from llm.solar_llm import SolarLLM
```

```python
from llm.local_llm import LocalLLM
from settings import CONFIG


class Router:
    def __init__(self):
        self.cache = {}  # LLM 인스턴스 캐싱

    def route(self, model_name: str):
        """
        모델 이름을 받아 적절한 LLM 인스턴스 반환

        Args:
            model_name: "openai", "claude", "local" 등

        Returns:
            llm: BaseLLM 인스턴스
        """
        # 캐시 확인
        if model_name in self.cache:
            return self.cache[model_name]

        # 기본값 처리
        if model_name == "default":
            model_name = CONFIG["llm"]["default"]

        # LLM 인스턴스 생성
        if model_name == "openai":
            llm = OpenAILLM(
                model=CONFIG["llm"]["providers"]["openai"]["model"]
            )
        elif model_name == "claude":
            llm = ClaudeLLM(
                model=CONFIG["llm"]["providers"]["claude"]["model"]
            )
        elif model_name == "gemini":
            llm = GeminiLLM(
                model=CONFIG["llm"]["providers"]["gemini"]["model"]
            )
        elif model_name == "solar":
            llm = SolarLLM(
                model=CONFIG["llm"]["providers"]["solar"]["model"]
            )
        elif model_name == "local":
            llm = LocalLLM(
                endpoint=CONFIG["llm"]["providers"]["local"]["endpoint"]
            )
        else:
            raise ValueError(f"Unknown model: {model_name}")

        # 캐싱
        self.cache[model_name] = llm

        return llm
```

학습 효율을 높이는 팁

1. 연결 문제 해결 체크리스트

❌ 연결이 안될 때:

1. **settings.py 확인:**

```
# 마지막에 반드시 있어야 함!
CONFIG = load_config()
API_KEYS = load_api_keys()
```

2. **API Keys 확인:**

```
# secrets/api_keys.yaml 존재하는지
ls secrets/

# 내용 확인
cat secrets/api_keys.yaml
```

3. **Docker 로그 확인:**

```
docker compose logs gateway
```

4. **API Key 유효성:**

```
# OpenAI 테스트
curl https://api.openai.com/v1/chat/completions \
  -H "Authorization: Bearer YOUR_API_KEY" \
  -H "Content-Type: application/json" \
  -d '{"model":"gpt-4o-mini","messages":[{"role":"user","content":"test"}]}'
```

2. 모델별 테스트

```
# test_llm.py
from llm.openai_llm import OpenAILLM
from llm.claude_llm import ClaudeLLM
from llm.local_llm import LocalLLM

# OpenAI 테스트
openai_llm = OpenAILLM(model="gpt-4o-mini")
```

```
result = openai_llm.generate("Hello!")
print(f"OpenAI: {result.text}")
print(f"Usage: {result.usage}")

# Claude 테스트
claude_llm = ClaudeLLM(model="claude-3-sonnet-20240229")
result = claude_llm.generate("Hello!")
print(f"Claude: {result.text}")
print(f"Usage: {result.usage}")

# Local 테스트
local_llm = LocalLLM(endpoint="http://localhost:11434")
result = local_llm.generate("Hello!")
print(f"Local: {result.text}")
print(f"Usage: {result.usage}")  # None
```

🔧 직접 해보기: 실습

실습 1: settings.py 완성하기

```
# gateway/settings.py
import yaml
from pathlib import Path

BASE_DIR = Path(__file__).resolve().parent.parent

def load_config():
    config_path = BASE_DIR / "config.yaml"
    if not config_path.exists():
        raise FileNotFoundError(f"config.yaml not found at {config_path}")

    with open(config_path, "r", encoding="utf-8") as f:
        return yaml.safe_load(f)

def load_api_keys():
    secrets_path = BASE_DIR / "secrets" / "api_keys.yaml"
    if not secrets_path.exists():
        raise FileNotFoundError(
            f"api_keys.yaml not found at {secrets_path}"
        )

    with open(secrets_path, "r", encoding="utf-8") as f:
        return yaml.safe_load(f)

# ⚠ 이 부분이 없으면 연결 안됨!
CONFIG = load_config()
API_KEYS = load_api_keys()
```

실습 2: 단순 테스트 스크립트

```python
# simple_test.py
import sys
sys.path.append("/app/gateway")  # Docker 내부 경로

from llm.openai_llm import OpenAILLM

try:
    llm = OpenAILLM(model="gpt-4o-mini")
    result = llm.generate("Say hello in 5 words")

    print(f"☑ Success!")
    print(f"Text: {result.text}")
    print(f"Usage: {result.usage}")

except Exception as e:
    print(f"❌ Error: {e}")
```

실습 3: 모든 LLM 동시 테스트

```python
# test_all_llms.py
from gateway.router import Router

router = Router()
models = ["openai", "claude", "gemini", "solar", "local"]
prompt = "RAG 시스템을 한 문장으로 설명하세요."

for model_name in models:
    try:
        print(f"\n{'='*50}")
        print(f"Testing: {model_name}")

        llm = router.route(model_name)
        result = llm.generate(prompt, max_tokens=100)

        print(f"☑ Success")
        print(f"Response: {result.text[:100]}...")
        print(f"Usage: {result.usage}")

    except Exception as e:
        print(f"❌ Failed: {e}")
```

☑ Day 8 완료 기준

다음을 설명할 수 있으면 통과입니다:

5가지 필수 질문

1. **LLMResult를 왜 사용하나?** ☑ 타입 안전, IDE 지원, dict보다 안전

2. **settings.py 마지막에 꼭 필요한 코드는?** ☑ `CONFIG = load_config()`, `API_KEYS = load_api_keys()`
3. **Local LLM의 usage가 None인 이유는?** ☑ llama.cpp가 토큰 정보 제공 안함
4. **각 LLM별 특징은?** ☑ OpenAI(표준), Claude(긴 문맥), Gemini(무료), SOLAR(한글), Local(무료)
5. **연결 안될 때 체크 포인트는?** ☑ settings.py 마지막, API Keys 파일, Docker 로그, API Key 유효성

➡ SOON 다음 단계 (Day 9)

Day 9에서는 (전자책 전용):

Gateway + RAG 통합:

```
# Retrieve
contexts = retrieval_pipeline.search(query)

# Generate (Gateway 사용)
llm = router.route("claude")
response = llm.generate(
    prompt=f"문서: {contexts}\n질문: {query}"
)
```

완전한 RAG 시스템!

💬 학습 후 자가 점검

모두 답할 수 있다면 Day 8 완료! 🎉

1. **LLMResult 구조는?** ☑ text: str, usage: Optional[Dict[str, int]]
2. **BaseLLM의 역할은?** ☑ 추상 클래스, 모든 LLM이 상속, generate() 정의
3. **API_KEYS는 어디서 로드?** ☑ settings.py에서 load_api_keys(), 모듈 레벨 변수
4. **timeout은 왜 60초?** ☑ LLM 응답 시간 고려, 너무 짧으면 타임아웃
5. **실제 연결 안될 때 첫 체크는?** ☑ settings.py 마지막 2줄 있는지 확인!

📜 라이선스

허용

- ☑ 개인 학습 목적 사용

- ☑ 출처 표시 후 비영리 공유

금지

- ❌ 상업적 이용
- ❌ 내용 수정 및 2차 저작
- ❌ 저작자 허락 없는 재배포

상업적 이용 문의: j4angguiop@gmail.com

⭐ 이 프로젝트가 도움이 되었다면 Star를 눌러주세요!