[사례 첨부] Antigravity 하이브리드 자동화 파이프라인 소스코드

본 문서는 로컬/클라우드 하이브리드 소프트웨어 엔지니어링 파이프라인의 핵심 구조와 엔진을 증명합니다.

README.txt

```
================================================================
[사례 첨부] Antigravity 하이브리드 파이프라인 시스템 핵심 소스코드
================================================================

본 압축 파일은 『오프라인-온라인 경계를 허무는 로컬/클라우드 하이브
소프트웨어 엔지니어링 자동화 시스템』의 핵심 기술력과 무결성을 증명
6개의 원본 소스코드 및 아키텍처 문서입니다.

■ 1. 기획 및 아키텍처 문서 (시스템 철학 및 설계 증명)
----------------------------------------------------------------
[1] GEMINI.md
    - 시스템의 헌법입니다. 환각 방지를 위한 룰 베이스 다중 에이전트
      규칙과 '집도 잠금(Execution Lock)' 절차가 명시되어 있습니다.
[2] system_architecture_report.md
    - 전체 다중 에이전트 시스템의 데이터 전달 체계(Handoff) 및
      클라우드 ↔ 로컬 폴백(Fallback) 우회 아키텍처 기술 명세서입니

■ 2. 파이프라인 베이스 엔진 (기술 구현력 증명)
----------------------------------------------------------------
[3] auto_pipeline.py
    - 전체 작업(사전 RAG 검색 → 교차 검증 → 최종 라우팅)을 한 번에
      파이프라인 최상위 래퍼이자 오케스트레이터입니다.
[4] relay_review.py
    - 본 시스템의 핵심인 '2-Phase 교차 검증' 코드가 담긴 파일입니다
      클라우드 AI(Gemini)와 로컬 AI(Llama-3.1/Gemma4)가 인터넷
      어떻게 위기를 넘기고 협업하는지 보여줍니다.
[5] ollama_bridge.py
    - 사내 폐쇄망 환경을 상정한 로컬 모델(Ollama) 통신 브릿지이자
      자가 지식 구축(KI System) 로직이 담긴 엔진입니다.

■ 3. 안정성 및 완성도 증명 (소프트웨어 공학적 성취)
----------------------------------------------------------------
[6] test_pipeline_e2e.py
    - 실제 운영 파일 보호를 위해 테스트용 더미 파일을 안전하게 동적
      삭제하며 통합 파이프라인 시스템 전체를 실증하는 E2E 테스트 스

================================================================
이 코드들은 특정 산업(AutoCAD Plant 3D 등)을 넘어, 보안이 요구되는
```

어떤 폐쇄망 현장에서도 자율형 AI 엔지니어로 작동할 수 있도록 설계
==========

GEMINI.md

⚡ GEMINI CORE ABSOLUTE RULES (최우선 적용)
1. **[모드 앵커 강제]**: 매 턴 첫 응답 텍스트(Line 1)에는 무조건
2. **[집도 잠금]**: 명시적 승인(Yes) 없이는 코드 수정 툴 호출 절
3. **[작업 분기]**: 기본 모드는 `/분석`(수정 금지).
4. **[출력 순서 강제]**: 응답의 구조는 항상 다음 순서를 따른다.
   (1) [모드 확인] 앵커 (무조건 Line 1)
   (2) [자가 진단] 앵커 (툴 호출 승인 직후 진입하는 특수 턴에 한정
   (3) [컨텍스트 리프레시] (해당 갱신 요건 만족 시 출력)
   (4) [자체 비판 (Self-Criticism)] 블록 (초안/계획 제안 턴 한정
   (5) 텍스트 본문 (해설, 마크다운 등)

> **[❗ AI 인지 스키마: 파라미터 구조적 한계의 자가 인정]**
> 소형화 모델(Flash)의 구조상, 문서가 길어지거나 후반 로드 시 인

[명령어 및 워크플로우 시스템 변수 매핑]
- 명령어 별칭: `/ㅁㅇ` == `/auto-dev` == `/ad`
- 시스템 루트 변수: `<appDataDir>` = `C:\Users\HT노승환\.gemini
- 삼위일체(Triad Sync) 보관소: `<appDataDir>\global_workflows\`

[명령어 기반 모드 통제]
- `/분석`: 코드 수정 없음, 로그/데이터 확인만 진행 (잠금: ON)
- `/계획`: 실물 코드가 포함된 계획서 작성, 집도 금지 (잠금: ON)
- `/리뷰`: 기존 코드 비판 수행. 텍스트 형태의 마크다운 '개선 제안
- `/집도`: "집도 권한 진입을 요청"하는 턴 전환. (명령어를 썼다고
- `/리프레시`: 현재까지의 작업 컨텍스트를 `Conversation_Summary_

[최상위 원칙 / Top Principle]
> Implementation Plan, Task, Walkthrough, 그리고 모든 대화는 **
> 과장이나 비유적 표현을 배제하고, 있는 사실 그대로 명확하게 전달
> 대화 첫 문장에 `/ㅁㅇ`, `/ad`, `/auto-dev` 등의 명령어가 포함

0. 집도 절대 원칙 (Execution Lock Rules)
코드를 수정하는 행위(집도)는 다음의 엄격한 시간적 흐름과 통제
1. ***집도 잠금(Execution Lock)**: 사용자가 "진행해줘", "해줘"*
2. ***집도 승인 요청(Request)**: 사용자와의 피드백 교환을 거쳐 **
3. ***승인 확인 및 자가 진단(Checklist)**: 사용자가 승인 또는 거*
   - [승인 시]: `[자가 진단] 명시적 집도 승인 여부: Yes → 툴 호출
   - [거절/보류 시]: `[자가 진단] 명시적 집도 승인 여부: No → 툴
4. ***단계별 완전 분리**: 학습(`--force learn`), 분석(`pre_anal*
5. **단일 명령-단일 실행 (Literal Execution)**: 지시한 1개의 사
6. **실물 코드 등재 원칙**: 계획서 작성 및 모든 코드 제안 대화 시

7. **워크플로우 삼위일체**: 규칙, 특히 **[⚡ GEMINI CORE ABSOL
   - **[Sync 장애 대응]**: 만약 스크립트가 에러 나거나 동기화에

1. 개발 태도 (Attitude)
* **"가능성 및 추측은 금지하고, 디버그 로그를 통해 상세 내용을
* **No Assumption**: "이럴 것이다"라는 가정으로 로직을 변경하면

2. 스킬 활용 (Skills) 및 자가 지식 의무화 (Knowledge Items)
**작업 착수 전 해당 분야의 스킬 파일(SKILL.md)을 `view_file`
스킬 경로: `<appDataDir>\skills\plugins\antigravity-awesome-s
작업 성격에 맞는 스킬 디렉토리를 `list_dir`로 탐색하여 적합한

오케스트레이터가 학습(`--force learn`)한 통찰력은 `<appDataDir

3. 하이브리드 워크플로우 정책 (Hybrid Workflow Policy)
로컬 생태계를 활용하여 비용을 절감하고 역할을 분담한다.
* 💻 **코드 코파일럿 (gemma4)**: 코드 타이핑과 핵심 로직 초안
* 📝 **리뷰어 (Llama-3.1)**: 매끄러운 문서 작성 및 아키텍처 코
* 👁 **QA 엔지니어 (Llama-Vision)**: 에러가 발생한 수식/도면
* 🧠 **데이터베이스 (Nomic-Embed)**: 프로젝트 내부 텍스트 벡터
* 👹 **자율 에이전트 (Hermes)**: 오케스트레이터의 위임을 받아

⚡ 완전 자동화 파이프라인 (2025-04 설계, 2026-04 갱신)
명령어 실행 시 **반드시** 아래 순서를 따른다:

Step 0. [자동] `pre_analyze.py` 실행
  → Nomic RAG 및 Llama-3.1 파일 요약 분석 → `context_summary.m

Step 0.5. [자동] `relay_review.py` 실행 → `relay_result.md` 생
  → **Phase 1 (클라우드-로컬 하이브리드 검토 릴레이)**: Step 0
  - **양방향 폴백(Fallback) 조치**: (1) 클라우드 연결 및 생성이
  → **Phase 2 (Antigravity 최종 아키텍트 검열)**: 릴레이를 거쳐

Step 1. [자동] `orchestrator.py` 분기 위임 (`relay_result.md`
Step 2. [판단] Antigravity 취합 및 설계 (섹션 0 집도 원칙 적용)
Step 3. [조건부 백그라운드] 스킬 학습 진행
  → 단순 코드 오타 수정이 아닌, '새로운 아키텍처 패턴 도출'이나 '

4. Gemini 모델 품질 극대화 (Reflective Thinking)
속도보다 **논리적 완결성과 지속적 검토**를 최우선으로 합니다.

1단계: 요구사항 분해 및 반추 (Reflection)
 - 도구 호출 전, "이 호출이 반드시 필요한가? 더 효율적인 방법은 없
2단계: 초안 작성 및 텍스트 시각화 자체 비판 (Self-Criticism)
 - 결과가 정리되면, 계획서 제안 직전 반드시 아래의 포맷을 텍스트로
  ```
  ```

```
## [자체 비판 (Self-Criticism)]
- 잠재적 버그 판단: (없으면 "없음" 명시)
- 개선점 1:
- 개선점 2:
- 개선점 3:
## [최종 반영 여부]: 위 개선점 중 기획안에 반영한 것 / 제외한
```

3단계: 최종 보완 (Final Polish)
- 자체 비판을 통해 완성된 결과물을 도출하여 사용자에게 단일 계획

5. 로컬 에이전트 모니터링 및 장애 대응

로컬 에이전트와의 협업 중 발생할 수 있는 비정상 종료 대응:

1. **상태 감지**: 배경 작업이 `done`이나 결과가 없으면 `command_
2. **강제 분석**: 에이전트 비정상 종료 시 에러 로그를 분석하여 사
3. **유동적 타임아웃**: 작업 성질(단순 검색 1분, 대규모 RAG/코드

system_architecture_report.md

```
# Antigravity 로컬/클라우드 하이브리드 파이프라인 아키텍처 보고

본 문서는 `GEMINI.md`의 통제 하에 가동되는 7개의 핵심 스크립트가

---

## 1. 코어 통제 규칙 (`GEMINI.md`)
시스템의 "헌법"과 같은 역할을 하며, AI 에이전트(본체)가 수행해야 할

- **역할**: 파이프라인의 **논리적 지휘소**.
- **연동 방식**:
  - 에이전트는 사용자의 프롬프트 이전에 이 파일을 인라인으로 주입받
  - 문서 내 섹션 3에서 기술된 순서(`Step 0.5`, `Step 1` 등)가 실

### 📜 워크플로우 진입점 (`auto-dev.md`)
`GEMINI.md`가 글로벌 헌법이라면, `auto-dev.md`는 트리거 워크플로

- **역할**: 오케스트레이터 파이프라인의 물리적인 **진입점(Entry P
- **연동 방식**:
  - 사용자가 `/auto-dev`, `/ㅁㅇ`, `/ad` 명령어를 입력하면 에이전
  - 이 워크플로우 안에 선언된 `// turbo-all` 및 스크립트 실행 지시
  - `claude_wrapper.py`와 같은 실험적 기능이나, 특정 시스템 특수

- **로컬 생태계를 활용하여 비용을 절감하고 역할을 분담한다.
* 💻 **코드 코파일럿 (gemma4)**: 코드 타이핑과 핵심 로직 초안
* 📝 **리뷰어 (Llama-3.1)**: 매끄러운 문서 작성 및 아키텍처 코
* 👁️ **QA 엔지니어 (Llama-Vision)**: 에러가 발생한 수식/도면 캡
* 🧠 **데이터베이스 (Nomic-Embed)**: 프로젝트 내부 텍스트 벡터
* 👹 **자율 에이전트 (Hermes)**: 오케스트레이터의 위임을 받아

---

## 2. 파이프라인 3단계 체인: The "Auto Pipeline"

모든 작업은 `auto_pipeline.py` 래퍼에 의해 자동화된 3단계 체인

### ⚡ [래퍼] `auto_pipeline.py`
- **역할**: 오케스트레이터의 오케스트레이터. 사용자가 하달한 1
- **연동 구현**: 각 스크립트를 독립된 `subprocess`로 실행하며

### 🔍 Step 0: 사전 분석 (`pre_analyze.py`)
```

- **역할**: RAG 벡터 검색 기반 컨텍스트 수집가
- **구현 방식**:
  - Nomic Embed 기반 로컬 벡터 검색기로 쿼리 연관 코드를 탐색
  - 검색된 코드 스니펫들을 묶어 `ollama_bridge.py`의 `search/c
- **데이터 흐름**: 출력물 `scratch/context_summary.md` 생성

⚖️ Step 0.5: 릴레이 검증 (`relay_review.py`)
- **역할**: 클라우드 고속 생성기(Flash) & 로컬 비판자(Llama)의
- **구현 방식**:
  - [Phase 1-A] `context_summary.md`를 바탕으로 `GEMINI_API_KI
  - [Phase 1-B] 생성된 초안을 `ollama_bridge.py`의 `review` 도
- **데이터 흐름**: 입력 `context_summary.md` → 출력 `scratch/r

🔀 Step 1: 라우터 (`orchestrator.py`)
- **역할**: 최종 실행 노드 라우터
- **구현 방식**:
  - `relay_result.md`에 쌓인 초안과 비판의 결과물을 최종 프롬
  - "분석", "구조 파악", "자율" 등 자연어 작업 지시어의 **정규

---

3. 베이스 컴포넌트: 모델 & 대시보드

위의 파이프라인 체인이 구동될 수 있도록 뒤를 받치는 코어 엔진

🧱 모델 연동기 (`ollama_bridge.py`)
- **역할**: 파이프라인 전체의 로컬 지능 엔진. **모든 스크립트는
- **구현 방식**:
  - `generate(Gemma4)`, `review(Llama-3.1)`, `search(Nomic)`
  - `urllib`를 이용해 `localhost:11434` (Ollama 서버)로 직접 RE
  - **KI(Knowledge Item) 저장소 역할**: `learn` 모드 실행 시 로

📊 모니터링 (`agent_dashboard.py`)
- **역할**: 다중 에이전트 네트워크의 실시간 상태 확인. (FastAP
- **구현 방식**:
  - `Working`, `Done`, `Error` 상태를 각 에이전트에서 `POST /r
  - `ollama_bridge.py`와 `orchestrator.py` 실행 시 백그라운드

🤖 Claude 호환 래퍼 (`claude_wrapper.py`)
- **역할**: Anthropic CLI(`claude -p`) API를 가로채어 로컬 Oll
- **구현 방식**: 환경변수 `ANTHROPIC_BASE_URL`을 로컬호스트(`1

---

4. 폴백 (Fallback) 메커니즘 & 장애 저항성 설계

1. **클라우드 서버 다운/API Key 누락 (`relay_review.py`)**
   - 증상: REST API 호출 실패 (`HTTPError` 등)
   - 극복: 클라우드 파이프라인을 끊고, 즉각 **Gemma4** 단독 생
2. **로컬 모델 응답 시간 초과 (`auto_pipeline.py`)**
   - 증상: Llama-3.1의 리뷰 생성에 600초가 경과
   - 극복: 즉시 작업을 중단(SIGKILL)하고, 이전 단계까지의 성공
3. **DB 접속 실패 (`ollama_bridge.py`)**
   - 증상: `learn` 모드 중 SQLite Lock
   - 극복: DB 저장 분기에만 예외(Exception)를 격리하고 다음 분

이 모든 절차가 **`sys.exit()` 코드 관리와, `command_status` JS

auto_pipeline.py

```
"""
auto_pipeline.py - 완전 자동화 파이프라인 체인 래퍼
==================================================
GEMINI.md Step 0 → Step 0.5 → Step 1 을 하나의 명령으로 순차 실

Pipeline:
  Step 0   : pre_analyze.py  → context_summary.md 생성 (RAG +
  Step 0.5 : relay_review.py → relay_result.md 생성 (Cloud 초
  Step 1   : orchestrator.py → 에이전트 위임 (코드 생성/수정)

각 Step이 실패하면 즉시 중단하고 JSON 요약을 출력합니다.

Usage:
  python auto_pipeline.py --task "GD2.py TYPE A 로직 활성화" -
  python auto_pipeline.py --task "FS.py 리팩토링" --project-di
  python auto_pipeline.py --task "코드 분석" --skip-relay
  python auto_pipeline.py --task "빠른 검토" --skip-pre-analyz
"""

import argparse
import json
import os
import subprocess
import sys
import datetime

# ── 경로 상수 ──────────────────────────────────────────────
SCRIPT_DIR = os.path.dirname(os.path.abspath(__file__))
PRE_ANALYZE = os.path.join(SCRIPT_DIR, "pre_analyze.py")
RELAY_REVIEW = os.path.join(SCRIPT_DIR, "relay_review.py")
ORCHESTRATOR = os.path.join(SCRIPT_DIR, "orchestrator.py")
PYTHON_EXE = sys.executable

CONTEXT_SUMMARY = os.path.join(SCRIPT_DIR, "scratch", "contex
RELAY_RESULT = os.path.join(SCRIPT_DIR, "scratch", "relay_res


def print_banner(task: str):
    print()
    print("╔═══════════════════════════════════════════════════
    print("║      ⚡ Auto Pipeline - 완전 자동화 체인 실행
```

```
    print("╠═══════════════════════════════════════════
    print(f"║  📋 Task : {task[:50]:<50}  ║")
    print("║  🔗 Chain: pre_analyze → relay_review → orchestr
    print("╚═══════════════════════════════════════════
    print()


def run_step(step_name: str, cmd: list, timeout: int = 600) -
    """Step을 실행하고 결과를 반환합니다."""
    print(f"\n{'='*60}")
    print(f"  ▶ {step_name}")
    print(f"{'='*60}")

    try:
        result = subprocess.run(
            cmd,
            timeout=timeout,
            capture_output=False,  # stdout/stderr를 실시간 출
        )
        success = result.returncode == 0
        status = "SUCCESS" if success else f"FAILED (exit={re
        print(f"\n  {'✅' if success else '❌'} {step_name}:
        return {"step": step_name, "status": status, "success
    except subprocess.TimeoutExpired:
        print(f"\n  ❌ {step_name}: TIMEOUT ({timeout}초)")
        return {"step": step_name, "status": f"TIMEOUT ({time
    except Exception as e:
        print(f"\n  ❌ {step_name}: ERROR ({e})")
        return {"step": step_name, "status": f"ERROR: {e}", "


def main():
    parser = argparse.ArgumentParser(
        description="auto_pipeline.py - 완전 자동화 파이프라인
        formatter_class=argparse.RawDescriptionHelpFormatter,
        epilog=(
            "사용 예시:\n"
            '  python auto_pipeline.py --task "GD2.py TYPE A
            '  python auto_pipeline.py --task "FS.py 리팩토링"
            '  python auto_pipeline.py --task "코드 분석" --sk
        )
    )
    parser.add_argument("--task", "-t", required=True,
                        help="작업 요구사항 설명")
    parser.add_argument("--target-file", "-f", default=None,
                        help="수정 대상 파일 경로")
```

```
    parser.add_argument("--project-dir", "-d", default=".",
                        help="프로젝트 디렉토리 (pre_analyze 다
    parser.add_argument("--top-k", "-k", type=int, default=5,
                        help="RAG 탐색 파일 수 (기본: 5)")
    parser.add_argument("--skip-pre-analyze", action="store_t
                        help="Step 0 (pre_analyze) 건너뜀")
    parser.add_argument("--skip-relay", action="store_true",
                        help="Step 0.5 (relay_review) 건너뜀")
    parser.add_argument("--skip-orchestrator", action="store_
                        help="Step 1 (orchestrator) 건너뜀")
    parser.add_argument("--no-cloud", action="store_true",
                        help="relay_review에서 클라우드 생성 건
    args = parser.parse_args()

    print_banner(args.task)

    results = []
    start_time = datetime.datetime.now()

    # ── Step 0: pre_analyze.py ──────────────────────────
    if not args.skip_pre_analyze:
        cmd = [
            PYTHON_EXE, PRE_ANALYZE,
            "--task", args.task,
            "--project-dir", args.project_dir,
            "--top-k", str(args.top_k),
        ]
        if args.target_file:
            cmd.extend(["--file", args.target_file])

        step = run_step("Step 0: Pre-Analyze (RAG + Llama 요약
        results.append(step)

        if not step["success"]:
            print("\n⚠️  Step 0 실패 → 파이프라인 중단")
            print_summary(results, start_time)
            sys.exit(1)
    else:
        print("\n  ⏭️  Step 0 건너뜀 (--skip-pre-analyze)")
        results.append({"step": "Step 0", "status": "SKIPPED"

    # ── Step 0.5: relay_review.py ───────────────────────
    if not args.skip_relay:
        cmd = [PYTHON_EXE, RELAY_REVIEW, "--task", args.task]

        if args.target_file:
```

```
            cmd.extend(["--target-file", args.target_file])

        # Step 0의 산출물이 있으면 자동 연결
        if os.path.isfile(CONTEXT_SUMMARY):
            cmd.extend(["--context-file", CONTEXT_SUMMARY])

        if args.no_cloud:
            cmd.append("--no-cloud")

        step = run_step("Step 0.5: Relay Review (Cloud 초안 +
        results.append(step)

        if not step["success"]:
            print("\n⚠️  Step 0.5 실패 → 파이프라인 중단")
            print_summary(results, start_time)
            sys.exit(1)
    else:
        print("\n  ⏭️  Step 0.5 건너뜀 (--skip-relay)")
        results.append({"step": "Step 0.5", "status": "SKIPPE

    # ── Step 1: orchestrator.py ──────────────────────────
    if not args.skip_orchestrator:
        # relay_result.md의 내용을 orchestrator 프롬프트에 포함
        relay_context = ""
        if os.path.isfile(RELAY_RESULT):
            with open(RELAY_RESULT, "r", encoding="utf-8", er
                relay_context = f.read(3000)

        ork_prompt = args.task
        if relay_context:
            ork_prompt = f"{args.task}\n\n[릴레이 검토 결과 요

        cmd = [
            PYTHON_EXE, ORCHESTRATOR,
            ork_prompt,
        ]
        if args.target_file:
            cmd.extend(["--file", args.target_file])
        cmd.extend(["--project-dir", args.project_dir])

        step = run_step("Step 1: Orchestrator (에이전트 위임)"
        results.append(step)
    else:
        print("\n  ⏭️  Step 1 건너뜀 (--skip-orchestrator)")
        results.append({"step": "Step 1", "status": "SKIPPED"
```

```
    # ── 최종 요약 ──────────────────────────────────────
    print_summary(results, start_time)

    # 전체 성공 여부
    all_ok = all(r["success"] for r in results)
    sys.exit(0 if all_ok else 1)


def print_summary(results: list, start_time: datetime.datetim
    """파이프라인 최종 요약을 출력합니다."""
    elapsed = (datetime.datetime.now() - start_time).total_se

    print()
    print("╔═══════════════════════════════════════════════
    print("║              📊 Pipeline Summary
    print("╠═══════════════════════════════════════════════
    for r in results:
        icon = "✅" if r["success"] else "❌"
        print(f"║  {icon} {r['step']:<30} {r['status']:<20}
    print(f"║  ⏱  총 소요 시간: {elapsed:.1f}초{' '*(36-len(f
    print("╚═══════════════════════════════════════════════
    print()

    # JSON 요약 (Antigravity가 command_status로 파싱 가능)
    summary = {
        "pipeline": "auto_pipeline",
        "elapsed_seconds": round(elapsed, 1),
        "steps": results,
        "all_success": all(r["success"] for r in results),
    }
    print("[PIPELINE_RESULT]")
    print(json.dumps(summary, ensure_ascii=False))


if __name__ == "__main__":
    main()
```

relay_review.py

```
"""
relay_review.py - 2-Phase 릴레이 검증 파이프라인
================================================
GEMINI.md Step 0.5에 명시된 "클라우드-로컬 하이브리드 검토 릴레이
실제로 구현하는 스크립트입니다.

파이프라인:
  Phase 1-A: 클라우드(Gemini Flash API) 초안 생성
  Phase 1-B: 로컬(Llama-3.1) 비판 릴레이
  출력: relay_result.md (초안 + 비판 + 취합)

양방향 폴백(Fallback):
  (1) 클라우드 실패 → 로컬(Gemma4) 단독 초안 생성
  (2) 로컬 비판 실패 → 비판 생략, 초안만 반환
  (3) 전면 장애(양측 모두 실패) → 에러 보고 후 즉시 중단

Usage:
  python relay_review.py --task "GD2.py TYPE A 로직 활성화" --
  python relay_review.py --task "FS.py 리팩토링" --context-fil
  python relay_review.py --task "코드 분석" --target-file foo.
"""

import argparse
import datetime
import json
import os
import subprocess
import sys
import urllib.request
import urllib.error

# ── 경로 상수 ──────────────────────────────────────────────
SCRIPT_DIR = os.path.dirname(os.path.abspath(__file__))
OLLAMA_BRIDGE = os.path.join(SCRIPT_DIR, "ollama_bridge.py")
PYTHON_EXE = sys.executable
DEFAULT_OUTPUT = os.path.join(SCRIPT_DIR, "scratch", "relay_r

# ── Gemini Flash API ──────────────────────────────────────
GEMINI_API_URL = "https://generativelanguage.googleapis.com/v
MAX_CHARS_PER_FILE = 6000  # 대상 파일에서 읽을 최대 문자 수
```

```
def print_banner(task: str, target: str):
    print()
    print("╔═══════════════════════════════════════════════
    print("║      🔬 2-Phase Relay Review Pipeline
    print("╠═══════════════════════════════════════════════
    print(f"║  📋 Task   : {task[:50]:<50}  ║")
    if target:
        fname = os.path.basename(target)
        print(f"║  📄 Target : {fname:<50}  ║")
    print("║  🔄 Flow   : Cloud Draft → Local Critique → Merg
    print("╚═══════════════════════════════════════════════
    print()


# ── 컨텍스트 로딩 ──────────────────────────────────────

def load_context(context_file: str = None, target_file: str =
    """context_summary.md와 대상 파일 내용을 결합하여 프롬프트 컨
    parts = []

    if context_file and os.path.isfile(context_file):
        with open(context_file, "r", encoding="utf-8", errors
            content = f.read(10000)
        parts.append(f"[사전 분석 요약 (context_summary.md)]\n

    if target_file and os.path.isfile(target_file):
        with open(target_file, "r", encoding="utf-8", errors=
            code = f.read(MAX_CHARS_PER_FILE)
        truncated = len(code) >= MAX_CHARS_PER_FILE
        label = f" (처음 {MAX_CHARS_PER_FILE}자)" if truncatec
        parts.append(f"[대상 파일: {os.path.basename(target_fi

    return "\n\n---\n\n".join(parts) if parts else "(컨텍스트


# ── Phase 1-A: 클라우드 초안 생성 (Gemini Flash API) ──────

def generate_draft_cloud(task: str, context: str, api_key: st
    """Gemini Flash REST API로 코드 초안을 생성합니다."""
    print("  [Phase 1-A] ☁  클라우드(Gemini Flash) 초안 생성

    prompt = (
        f"당신은 AutoCAD Plant 3D 커스텀 스크립트 전문 개발자입니
        f"아래 작업 요구사항과 컨텍스트를 분석하여 구체적인 코드 
        f"수정해야 할 코드 블록을 정확히 지정하고, 변경 전/후를 명
```

```
        f"[작업 요구사항]\n{task}\n\n"
        f"[컨텍스트]\n{context}"
    )

    url = f"{GEMINI_API_URL}?key={api_key}"
    payload = json.dumps({
        "contents": [{"parts": [{"text": prompt}]}],
        "generationConfig": {
            "temperature": 0.3,
            "maxOutputTokens": 4096,
        }
    }).encode("utf-8")

    req = urllib.request.Request(
        url, data=payload,
        headers={"Content-Type": "application/json"},
        method="POST",
    )

    try:
        with urllib.request.urlopen(req, timeout=120) as resp
            body = json.loads(resp.read().decode("utf-8"))
            text = body["candidates"][0]["content"]["parts"][
            print(f"  ✅ 클라우드 초안 생성 완료 ({len(text)}자
            return text
    except urllib.error.HTTPError as e:
        error_body = e.read().decode("utf-8", errors="ignore"
        print(f"  ❌ 클라우드 HTTP 에러: {e.code} - {error_boc
        return None
    except urllib.error.URLError as e:
        print(f"  ❌ 클라우드 연결 실패: {e.reason}")
        return None
    except Exception as e:
        print(f"  ❌ 클라우드 예외: {type(e).__name__}: {e}")
        return None


# ── Phase 1-A 폴백: 로컬 초안 생성 (Gemma4) ──────────────────

def generate_draft_local(task: str, context: str) -> str | No
    """Gemma4(로컬)로 코드 초안을 생성합니다. 클라우드 실패 시 폴
    print("  [FALLBACK] 💻 로컬(Gemma4) 단독 초안 생성 중..."

    prompt = (
        f"작업: {task}\n\n"
        f"컨텍스트:\n{context[:3000]}\n\n"
```

```
        f"위 내용을 분석하여 구체적인 코드 수정안을 제시하세요."
    )

    cmd = [
        PYTHON_EXE, OLLAMA_BRIDGE,
        "--mode", "generate",
        "--model", "gemma4:e2b",
        "--prompt", prompt,
    ]

    try:
        result = subprocess.run(
            cmd, capture_output=True, text=True,
            encoding="utf-8", timeout=300,
        )
        output = result.stdout.strip()
        if output and "[ERROR]" not in output:
            # bridge 헤더 줄 제거
            lines = [l for l in output.splitlines()
                     if not l.startswith("—") and not l.start
            clean = "\n".join(lines).strip()
            if clean:
                print(f"  ✅ 로컬 초안 생성 완료 ({len(clean)}
                return clean
        print(f"  ❌ 로컬 초안 생성 실패: {result.stderr[:200]
        return None
    except subprocess.TimeoutExpired:
        print("  ❌ 로컬 초안 타임아웃 (300초)")
        return None
    except Exception as e:
        print(f"  ❌ 로컬 초안 예외: {e}")
        return None


# ── Phase 1-B: 로컬 비판 릴레이 (Llama-3.1) ──────────────────

def critique_local(draft: str, task: str) -> str | None:
    """Llama-3.1로 초안을 비판합니다."""
    print("  [Phase 1-B] 🔍 로컬(Llama-3.1) 비판 릴레이 중..."

    prompt = (
        f"당신은 시니어 코드 리뷰어입니다.\n"
        f"아래는 '{task}' 작업에 대해 생성된 코드 수정 초안입니디
        f"이 초안의 논리적 결함, 누락된 엣지 케이스, 잠재적 버그를
        f"각 문제에 대해 구체적인 개선 방안을 제시하세요.\n\n"
        f"[초안]\n{draft[:3000]}"
```

```
    )

    cmd = [
        PYTHON_EXE, OLLAMA_BRIDGE,
        "--mode", "review",
        "--model", "llama3.1:8b",
        "--prompt", prompt,
    ]

    try:
        result = subprocess.run(
            cmd, capture_output=True, text=True,
            encoding="utf-8", timeout=180,
        )
        output = result.stdout.strip()
        if output and "[ERROR]" not in output:
            lines = [l for l in output.splitlines()
                     if not l.startswith("—") and not l.start
            clean = "\n".join(lines).strip()
            if clean:
                print(f"  ✅ 로컬 비판 완료 ({len(clean)}자)")
                return clean
        print(f"  ⚠️  로컬 비판 실패 (빈 응답)")
        return None
    except subprocess.TimeoutExpired:
        print("  ⚠️  로컬 비판 타임아웃 (180초)")
        return None
    except Exception as e:
        print(f"  ⚠️  로컬 비판 예외: {e}")
        return None


# ── 결과 저장 ──────────────────────────────────────────

def save_relay_result(
    task: str, draft: str, critique: str | None,
    draft_source: str, output_path: str
):
    """릴레이 결과를 relay_result.md로 저장합니다."""
    now = datetime.datetime.now().strftime("%Y-%m-%dT%H:%M:%S

    lines = [
        "# 2-Phase Relay Review Result",
        f"> Generated: {now}",
        f"> Task: \"{task}\"",
        f"> Draft Source: {draft_source}",
```

```
        f"> Critique Source: {'Llama-3.1 (로컬)' if critique e
        "",
        "---",
        "",
        "## Phase 1-A: 코드 초안",
        "",
        draft,
        "",
        "---",
        "",
        "## Phase 1-B: 비판 결과",
        "",
        critique if critique else "(로컬 비판 모델 응답 실패. A
        "",
        "---",
        "",
        "## Antigravity 최종 검열 대기",
        "",
        "> 위 초안과 비판을 종합하여 Antigravity(Phase 2)가 최종
        "> `command_status`로 이 파일의 내용을 확인한 뒤,",
        "> 기하학 시뮬레이션 / 교차 파일 비교 / 데이터 컨트랙트 등
        "",
    ]

    os.makedirs(os.path.dirname(output_path), exist_ok=True)
    with open(output_path, "w", encoding="utf-8") as f:
        f.write("\n".join(lines))

    return output_path


# ── Main ──────────────────────────────────────────────────

def main():
    parser = argparse.ArgumentParser(
        description="relay_review.py - 2-Phase 릴레이 검증 파이
        formatter_class=argparse.RawDescriptionHelpFormatter,
        epilog=(
            "사용 예시:\n"
            '  python relay_review.py --task "GD2.py TYPE A 도
            '  python relay_review.py --task "FS.py 리팩토링"
            '  python relay_review.py --task "코드 분석" --no-
        )
    )
    parser.add_argument("--task", "-t", required=True,
                        help="작업 요구사항 설명")
```

```
    parser.add_argument("--target-file", "-f", default=None,
                        help="수정 대상 파일 경로")
    parser.add_argument("--context-file", "-c", default=None,
                        help="context_summary.md 경로 (pre_ana
    parser.add_argument("--output", "-o", default=DEFAULT_OUT
                        help=f"결과 저장 경로 (기본: {DEFAULT_(
    parser.add_argument("--no-cloud", action="store_true",
                        help="클라우드 생성을 건너뛰고 로컬(Gemm
    args = parser.parse_args()

    print_banner(args.task, args.target_file)

    # ── 0. 컨텍스트 로딩 ──────────────────────────────────
    context = load_context(args.context_file, args.target_fil
    print(f"  [컨텍스트] {len(context)}자 로드 완료")
    print()

    # ── 1. Phase 1-A: 초안 생성 ──────────────────────────
    draft = None
    draft_source = ""
    api_key = os.environ.get("GEMINI_API_KEY", "")

    if not args.no_cloud and api_key:
        draft = generate_draft_cloud(args.task, context, api_
        if draft:
            draft_source = "Gemini Flash (클라우드)"

    if draft is None:
        if not api_key and not args.no_cloud:
            print("  ⚠️  GEMINI_API_KEY 환경변수 미설정 → 로컬
        draft = generate_draft_local(args.task, context)
        if draft:
            draft_source = "Gemma4 (로컬 폴백)"

    if draft is None:
        # 전면 장애: 양측 모두 실패
        error_report = {
            "status": "TOTAL_FAILURE",
            "task": args.task,
            "reason": "클라우드(Gemini Flash)와 로컬(Gemma4) 도
            "action": "파이프라인을 강제 중지합니다. Ollama 서버
        }
        print()
        print("╔═══════════════════════════════════════════════
        print("║  ❌ 전면 장애 (TOTAL FAILURE)
        print("╚═══════════════════════════════════════════════
```

```
            print(json.dumps(error_report, ensure_ascii=False, in
            sys.exit(1)

        print()

        # ── 2. Phase 1-B: 로컬 비판 릴레이 ──────────────────────
        critique = critique_local(draft, args.task)
        if critique is None:
            print("  [FALLBACK] 비판 생략 → Antigravity 본체가 직접

        print()

        # ── 3. 결과 저장 ──────────────────────────────────────
        output_path = save_relay_result(
            args.task, draft, critique, draft_source, args.output
        )

        print("╔═══════════════════════════════════════════════════
        print("║  ✅ 2-Phase Relay Review 완료
        print("╚═══════════════════════════════════════════════════
        print()
        print(f"  📄 초안 출처   : {draft_source}")
        print(f"  🔍 비판 출처   : {'Llama-3.1 (로컬)' if critique
        print(f"  💾 결과 저장   : {output_path}")
        print()
        print("  ▶ Antigravity Phase 2 검열을 위해 이 결과를 확인하
        print()


    if __name__ == "__main__":
        main()
```

ollama_bridge.py

```
"""
Ollama Bridge - Antigravity Hybrid Workflow Helper
==================================================
Antigravity가 로컬 Ollama 모델에 작업을 위임할 때 사용하는 브릿지
토큰 절약을 위해 단순 분석/리뷰/생성 작업을 로컬 모델에 위임합니다

Usage:
  python ollama_bridge.py --mode review --file <path>
  python ollama_bridge.py --mode analyze --file <path>
  python ollama_bridge.py --mode generate --prompt "..."
  python ollama_bridge.py --mode ask --prompt "..." --file <p
  python ollama_bridge.py --mode vision --file <image> --prom
  python ollama_bridge.py --mode search --prompt "..." --proj
"""

import argparse
import base64
import json
import math
import sys
import os
import urllib.request
import urllib.error
import sqlite3
import datetime

OLLAMA_URL = "http://localhost:11434/api/generate"
OLLAMA_EMBED_URL = "http://localhost:11434/api/embeddings"
DEFAULT_MODEL = "gemma4:e2b"

# Model Mapping (Hybrid Workflow Policy)
MODEL_MAP = {
    "review":   "llama3.1:8b",
    "analyze":  "llama3.1:8b",
    "generate": "gemma4:e2b",
    "ask":      "llama3.1:8b",
    "vision":   "llama3.2-vision:latest",
    "search":   "nomic-embed-text:latest",
    "learn":    "llama3.1:8b",
    "agent":    "llama3.1:8b",
}
```

```
# ── Mode-specific system prompts ──────────────────────────

SYSTEM_PROMPTS = {
    "review": (
        "You are a senior code reviewer. Analyze the provided
        "1. Potential bugs or logic errors\n"
        "2. Performance concerns\n"
        "3. Code style issues\n"
        "4. Security vulnerabilities (if any)\n"
        "Be concise. Use bullet points. Focus on actionable f
        "Reply in the SAME language as the user's prompt (Kor
    ),
    "analyze": (
        "You are a code analysis expert. Read the provided co
        "1. A brief summary of what the code does (2-3 senten
        "2. Key classes/functions and their roles\n"
        "3. Dependencies and external interactions\n"
        "4. Any notable patterns or anti-patterns\n"
        "Be concise and structured.\n"
        "Reply in the SAME language as the user's prompt (Kor
    ),
    "generate": (
        "You are an expert programmer. Generate clean, well-c
        "Follow best practices. Output ONLY the code with bri
        "Reply in the SAME language as the user's prompt (Kor
    ),
    "ask": (
        "You are a helpful programming assistant. Answer the
        "If code is provided as context, reference it in your
        "Reply in the SAME language as the user's prompt (Kor
    ),
    "vision": (
        "You are a QA engineer with expertise in UI/UX analys
        "Analyze the provided screenshot or wireframe image.
        "Identify any visual bugs, layout issues, error messa
        "If the user asks about a specific issue, focus on th
        "Reply in the SAME language as the user's prompt (Kor
    ),
    "learn": (
        "You are an expert at extracting reusable programming
        "Identify the core problem, the solution, and extract
        "Summarize logically. Focus on reusability."
    ),
    "agent": (
        "You are an autonomous AI agent capable of using tool
```

```
        "When requested, utilize the provided tools to execut
        "Always explain what you are about to do before using
    ),
}

def report_dashboard(model: str, status: str, file_path: str,
    """작업 관리자 서버로 에이전트의 현재 상태를 보고합니다."""
    try:
        data = json.dumps({
            "model": model,
            "status": status,
            "file": file_path,
            "task_description": (task_desc[:100] + "...") if
        }).encode("utf-8")
        req = urllib.request.Request(
            "http://localhost:8080/report",
            data=data,
            headers={"Content-Type": "application/json"},
            method="POST"
        )
        with urllib.request.urlopen(req, timeout=0.5):
            pass
    except Exception:
        pass # 대시보드 서버가 꺼져 있어도 무시

def query_ollama(model: str, system: str, prompt: str, contex
    """Send a request to the local Ollama API and stream the

    full_prompt = prompt
    if context_text:
        full_prompt = f"[CODE CONTEXT]\n```\n{context_text}\n

    payload = json.dumps({
        "model": model,
        "system": system,
        "prompt": full_prompt,
        "stream": False,
        "keep_alive": "24h",  # 모델을 메모리에 24시간 유지 (수
        "options": {
            "temperature": 0.3,
            "num_predict": 4096,
            "num_ctx": 4096,      # 추가: 8GB VRAM 한계 초과 방
        }
    }).encode("utf-8")

    req = urllib.request.Request(
```

```
        OLLAMA_URL,
        data=payload,
        headers={"Content-Type": "application/json"},
        method="POST",
    )

    try:
        with urllib.request.urlopen(req, timeout=300) as resp
            body = json.loads(resp.read().decode("utf-8"))
            return body.get("response", "(no response)")
    except urllib.error.URLError as e:
        return f"[ERROR] Ollama 서버 연결 실패: {e}. 'ollama s
    except Exception as e:
        return f"[ERROR] {type(e).__name__}: {e}"


# ── Vision Mode ──────────────────────────────────────────

def query_ollama_vision(model: str, system: str, prompt: str,
    """이미지 파일을 base64로 인코딩하여 Vision 모델에 전송합니다

    try:
        with open(image_path, "rb") as f:
            image_b64 = base64.b64encode(f.read()).decode("ut
    except Exception as e:
        return f"[ERROR] 이미지 파일 읽기 실패: {e}"

    payload = json.dumps({
        "model": model,
        "system": system,
        "prompt": prompt or "이 이미지를 분석해주세요.",
        "images": [image_b64],
        "stream": False,
        "keep_alive": "24h",  # 비전 모델 렌더링 유지 속도 개선
        "options": {
            "temperature": 0.3,
            "num_predict": 4096,
            "num_ctx": 4096,
        }
    }).encode("utf-8")

    req = urllib.request.Request(
        OLLAMA_URL,
        data=payload,
        headers={"Content-Type": "application/json"},
        method="POST",
```

```
    )

    try:
        with urllib.request.urlopen(req, timeout=300) as resp
            body = json.loads(resp.read().decode("utf-8"))
            return body.get("response", "(no response)")
    except urllib.error.URLError as e:
        return f"[ERROR] Ollama 서버 연결 실패: {e}"
    except Exception as e:
        return f"[ERROR] {type(e).__name__}: {e}"


# ── Embed / Search Mode (RAG via SQLite FTS5) ──────────────

EMBED_EXTENSIONS = {".cs", ".py", ".ts", ".tsx", ".js", ".jsx
EMBED_SKIP_DIRS = {"node_modules", ".venv", "bin", "obj", "di
EMBED_CHUNK_CHARS = 500  # 각 파일에서 임베딩할 최대 문자 수
DB_PATH = os.path.expanduser("~/.gemini/antigravity/ollama_me

def init_db():
    """SQLite FTS5 통합 메모리 DB를 초기화합니다."""
    os.makedirs(os.path.dirname(DB_PATH), exist_ok=True)
    conn = sqlite3.connect(DB_PATH)
    cur = conn.cursor()
    # 파편화된 파일 코드 보관 테이블 (Hybrid RAG 용)
    cur.execute('''
        CREATE VIRTUAL TABLE IF NOT EXISTS docs_fts USING fts
            project_id UNINDEXED,
            file_path UNINDEXED,
            snippet,
            embedding UNINDEXED
        )
    ''')
    # 과거 대화/히스토리 테이블 (Closed-loop Session Recall)
    cur.execute('''
        CREATE VIRTUAL TABLE IF NOT EXISTS memory_logs USING
            project_id UNINDEXED,
            session_id UNINDEXED,
            role UNINDEXED,
            content,
            timestamp UNINDEXED
        )
    ''')
    # 추출된 자동 스킬 테이블 (Autonomous Skill Generation)
    cur.execute('''
        CREATE VIRTUAL TABLE IF NOT EXISTS skills USING fts5(
```

```
            skill_name,
            description,
            pattern_code UNINDEXED
        )
    ''')
    conn.commit()
    return conn

def get_embedding(text: str, model: str = "nomic-embed-text:l
    """Ollama Embedding API를 호출하여 텍스트 벡터를 반환합니다.
    payload = json.dumps({
        "model": model,
        "prompt": text,
    }).encode("utf-8")

    req = urllib.request.Request(
        OLLAMA_EMBED_URL,
        data=payload,
        headers={"Content-Type": "application/json"},
        method="POST",
    )

    with urllib.request.urlopen(req, timeout=30) as resp:
        body = json.loads(resp.read().decode("utf-8"))
        return body.get("embedding", [])

def cosine_similarity(a: list, b: list) -> float:
    """두 벡터 간 코사인 유사도를 계산합니다."""
    dot = sum(x * y for x, y in zip(a, b))
    norm_a = math.sqrt(sum(x * x for x in a))
    norm_b = math.sqrt(sum(x * x for x in b))
    if norm_a == 0 or norm_b == 0:
        return 0.0
    return dot / (norm_a * norm_b)

def scan_project_files(project_dir: str) -> list:
    """프로젝트 디렉토리에서 인덱싱 대상 소스 파일 목록을 수집합니
    files = []
    for root, dirs, fnames in os.walk(project_dir):
        dirs[:] = [d for d in dirs if d not in EMBED_SKIP_DIR
        for fname in fnames:
            ext = os.path.splitext(fname)[1].lower()
            if ext in EMBED_EXTENSIONS:
                files.append(os.path.join(root, fname))
    return files
```

```
def build_or_load_index(project_dir: str, reindex: bool = Fal
    """프로젝트 파일들을 임베딩하고 SQLite FTS5 데이터베이스에 인
    conn = init_db()
    cur = conn.cursor()

    if reindex:
        cur.execute("DELETE FROM docs_fts WHERE project_id=?"
        conn.commit()
    else:
        # 캐시된(저장된) 데이터 존재 여부 확인
        cur.execute("SELECT COUNT(*) FROM docs_fts WHERE proj
        if cur.fetchone()[0] > 0:
            print(f"[EMBED] 데이터베이스 인덱스 로드 완료: {pro
            conn.close()
            return

    source_files = scan_project_files(project_dir)
    print(f"[EMBED] {len(source_files)}개 소스 파일 발견. SQLi

    count = 0
    for fpath in source_files:
        try:
            with open(fpath, "r", encoding="utf-8", errors="r
                snippet = f.read(EMBED_CHUNK_CHARS)

            if len(snippet.strip()) < 10:
                continue

            embedding = get_embedding(snippet)
            cur.execute(
                "INSERT INTO docs_fts(project_id, file_path,
                (project_dir, fpath, snippet[:200], json.dump
            )
            count += 1
            if count % 10 == 0:
                print(f"  [{count}/{len(source_files)}] index
                conn.commit()
        except Exception as e:
            print(f"  [SKIP] {fpath}: {e}", file=sys.stderr)

    conn.commit()
    conn.close()
    print(f"[EMBED] SQLite DB 인덱스 저장 완료: {count}개 저장됨


def search_similar(query: str, project_dir: str, top_k: int =
```

```
    """FTS5 하이브리드 RAG (키워드 + 코사인 벡터) 탐색을 수행합니
    conn = init_db()
    cur = conn.cursor()
    query_embedding = get_embedding(query)

    # 1. FTS5 BM25 키워드 검색 (단순 하위 집합 필터링)
    # 공백을 기준으로 OR 쿼리 구성
    words = query.replace('"', '').split()
    if words:
        fts_query = " OR ".join([f'"{w}"' for w in words])
        try:
            cur.execute('''
                SELECT file_path, snippet, embedding
                FROM docs_fts
                WHERE project_id=? AND docs_fts MATCH ?
                LIMIT 200
            ''', (project_dir, fts_query))
            rows = cur.fetchall()
        except Exception:
            rows = []
    else:
        rows = []

    # 2. 벡터 전수 탐색으로 대체 (FTS 결과가 없거나 FTS 구문 에러
    if not rows:
        cur.execute("SELECT file_path, snippet, embedding FRO
        rows = cur.fetchall()

    results = []
    for row in rows:
        fpath, snippet, emb_str = row
        emb = json.loads(emb_str)
        score = cosine_similarity(query_embedding, emb)
        results.append({
            "path": fpath,
            "snippet": snippet,
            "score": score,
        })

    conn.close()
    results.sort(key=lambda x: x["score"], reverse=True)
    return results[:top_k]


# ── Utility & Tools Execution (HITL) ──────────────────────
```

```
def read_file_safe(path: str, max_chars: int = 30000) -> str:
    """Read a file with size limit to avoid overwhelming the
    try:
        with open(path, "r", encoding="utf-8", errors="replac
            content = f.read(max_chars)
        if len(content) >= max_chars:
            content += f"\n\n... (truncated at {max_chars} ch
        return content
    except Exception as e:
        return f"[ERROR] 파일 읽기 실패: {e}"

def execute_code_sandboxed(command: str) -> str:
    """터미널 명령어 실행 (Human-in-the-loop 안전장치 적용됨)"""
    import subprocess
    print(f"\n[AGENT 도구: execute_code] 모델이 다음 명령을 실행
    print("⚠ 백그라운드 환경에서 이 명령을 실행하시겠습니까? (y/
    user_input = sys.stdin.readline().strip().lower()

    if user_input == 'y':
        try:
            print("[진행 중] 스크립트 실행...")
            res = subprocess.run(command, shell=True, capture
            output = res.stdout if res.stdout else res.stderr
            return output if output else "성공 (출력 없음)"
        except Exception as e:
            return f"실행 에러: {e}"
    else:
        return "사용자에 의해 실행 거부됨 (Human-in-the-loop 취


def save_as_ki(result_text: str, prompt: str):
    """학습 결과를 KI 규격(metadata.json + artifacts/skill.md)
    import hashlib

    ki_base = os.path.expanduser("~/.gemini/antigravity/knowl

    # 프롬프트 해시로 고유 ID 생성 (중복 학습 방지)
    ki_id = hashlib.md5(prompt.encode("utf-8")).hexdigest()[:
    ki_dir = os.path.join(ki_base, f"ki_{ki_id}")
    artifacts_dir = os.path.join(ki_dir, "artifacts")

    # 동일 해시의 KI가 이미 존재하면 덮어쓰기
    os.makedirs(artifacts_dir, exist_ok=True)

    # metadata.json
    now = datetime.datetime.now().isoformat()
```

```
    metadata = {
        "id": ki_id,
        "summary": result_text[:200],
        "created": now,
        "updated": now,
        "references": [
            {"type": "learn_prompt", "hash": ki_id, "source":
        ]
    }
    meta_path = os.path.join(ki_dir, "metadata.json")
    with open(meta_path, "w", encoding="utf-8") as f:
        json.dump(metadata, f, ensure_ascii=False, indent=2)

    # artifacts/skill.md
    skill_path = os.path.join(artifacts_dir, "skill.md")
    with open(skill_path, "w", encoding="utf-8") as f:
        f.write(f"# Extracted Skill\n\n> Generated: {now}\n>

    print(f"[KI] Knowledge Item 저장: {ki_dir}")

# ── Main ──────────────────────────────────────────────────

def main():
    parser = argparse.ArgumentParser(description="Ollama Brid
    parser.add_argument("--mode", choices=["review", "analyze
                        help="작업 모드: review, analyze, gene
    parser.add_argument("--model", default=None, help="사용할
    parser.add_argument("--file", help="분석할 파일 경로 (visi
    parser.add_argument("--prompt", default="", help="추가 프
    parser.add_argument("--max-chars", type=int, default=3000
    parser.add_argument("--project-dir", help="search/agent 도
    parser.add_argument("--reindex", action="store_true", hel
    parser.add_argument("--top-k", type=int, default=5, help=

    args = parser.parse_args()

    # 모드별 기본 모델 자동 선택
    model = args.model or MODEL_MAP.get(args.mode, DEFAULT_MO

    # ── Learn Mode (자체 스킬 개선) ──────────────────────
    if args.mode == "learn":
        print(f"── Ollama Bridge ({model}) ── Mode: learn ──
        if not args.prompt:
            print("[ERROR] 추출할 문맥(에러로그/채팅내용)을 --pr
            sys.exit(1)
        system = SYSTEM_PROMPTS["learn"]
```

```
        report_dashboard(model, "Working", "learn_mode", "학습
        result = query_ollama(model, system, args.prompt)

        # SQLite 에 스킬 저장
        try:
            conn = init_db()
            cur = conn.cursor()
            cur.execute("INSERT INTO skills(skill_name, descr
                        ("Extracted Skill", result[:50], resu
            conn.commit()
            print("[LEARN] DB 내 skills 테이블에 성공적으로 추출
        except Exception as e:
            print(f"[ERROR] DB 저장 실패: {e}")

        # KI 규격 파일 자동 생성 (metadata.json + artifacts/ski
        save_as_ki(result, args.prompt)

        report_dashboard(model, "Done", "learn_mode", "추출 성
        print("\n[추출 결과]\n" + result)
        return

    # ── Agent Mode (하위 에이전트 도구 위임) ──────────────
    if args.mode == "agent":
        print(f"─── Ollama Bridge ({model}) ── Mode: agent ──

        tools = [
            {
                "type": "function",
                "function": {
                    "name": "execute_code",
                    "description": "운영체제 시스템 명령어 스크
                    "parameters": {
                        "type": "object",
                        "properties": {
                            "command": {
                                "type": "string",
                                "description": "실행할 명령어
                            }
                        },
                        "required": ["command"]
                    }
                }
            }
        ]

        messages = [
```

```
            {"role": "system", "content": SYSTEM_PROMPTS["age
            {"role": "user", "content": args.prompt or "로컬 
        ]

        payload = {
            "model": model, "messages": messages, "tools": to
            "options": {"temperature": 0.2, "num_ctx": 4096}
        }

        req = urllib.request.Request(
            "http://localhost:11434/api/chat",
            data=json.dumps(payload).encode("utf-8"),
            headers={"Content-Type": "application/json"},
            method="POST"
        )
        try:
            with urllib.request.urlopen(req, timeout=300) as
                body = json.loads(resp.read().decode("utf-8")
                msg = body.get("message", {})

                if "tool_calls" in msg and msg.get("tool_call
                    for tc in msg["tool_calls"]:
                        fn = tc.get("function", {})
                        if fn.get("name") == "execute_code":
                            cmd = fn.get("arguments", {}).get
                            res = execute_code_sandboxed(cmd)
                            print(f"\n[에이전트 실행 결과 피드
                else:
                    print(f"[AGENT 응답]\n{msg.get('content',

        except urllib.error.URLError as e:
            print(f"[ERROR] Ollama 서버 연결 실패 (Chat API):

        print("-" * 50)
        return

    # ── Search Mode (FTS5 RAG) ──────────────────────────
    if args.mode == "search":
        if not args.project_dir:
            print("[ERROR] --project-dir 인자가 필요합니다.", f
            sys.exit(1)
        if not os.path.isdir(args.project_dir):
            print(f"[ERROR] 디렉토리를 찾을 수 없습니다: {args.
            sys.exit(1)
        if not args.prompt:
            print("[ERROR] --prompt 인자가 필요합니다.", file=s
```

```
            sys.exit(1)

        print(f"── Ollama Bridge ({model}) ── Mode: search ─
        report_dashboard(model, "Working", args.project_dir,

        build_or_load_index(args.project_dir, args.reindex)
        results = search_similar(args.prompt, args.project_di

        print(f"\n[SEARCH] Query: \"{args.prompt}\"")
        print(f"[SEARCH] Top {len(results)} results (Hybrid):
        print("─" * 50)
        for i, r in enumerate(results):
            print(f"  {i+1}. [{r['score']:.4f}] {r['path']}")
            print(f"     {r['snippet'][:80]}...")
        print("─" * 50)

        report_dashboard(model, "Done", args.project_dir, arg
        return

    # ── Vision Mode ──────────────────────────────────────
    if args.mode == "vision":
        if not args.file:
            print("[ERROR] --file 인자(이미지 파일 경로)가 필요
            sys.exit(1)
        if not os.path.isfile(args.file):
            print(f"[ERROR] 파일을 찾을 수 없습니다: {args.file
            sys.exit(1)

        print(f"── Ollama Bridge ({model}) ── Mode: vision ─
        print(f"🖼  Image: {args.file}")
        print("─" * 50)

        report_dashboard(model, "Working", args.file, args.pr

        system = SYSTEM_PROMPTS["vision"]
        result = query_ollama_vision(model, system, args.prom

        status_label = "Error" if "[ERROR]" in result else "D
        report_dashboard(model, status_label, args.file, args

        print(result)
        print("─" * 50)
        return

    # ── Standard Modes (review, analyze, generate, ask) ────
    context = ""
```

```
    if args.file:
        if not os.path.isfile(args.file):
            print(f"[ERROR] 파일을 찾을 수 없습니다: {args.file
            sys.exit(1)
        context = read_file_safe(args.file, args.max_chars)

    prompt = args.prompt
    if not prompt:
        if args.mode == "review":
            prompt = "이 코드를 리뷰해주세요."
        elif args.mode == "analyze":
            prompt = "이 코드를 분석해주세요."
        else:
            print("[ERROR] --prompt 인자가 필요합니다.", file=s
            sys.exit(1)

    system = SYSTEM_PROMPTS[args.mode]

    # Print header
    print(f"── Ollama Bridge ({model}) ── Mode: {args.mode}
    if args.file:
        print(f"📄 File: {args.file}")
    print("─" * 50)

    # Report Start to Dashboard
    report_dashboard(model, "Working", args.file, prompt)

    # Query
    result = query_ollama(model, system, prompt, context)

    # Report Done to Dashboard
    status_label = "Error" if "[ERROR]" in result else "Done"
    report_dashboard(model, status_label, args.file, prompt)

    print(result)
    print("─" * 50)


if __name__ == "__main__":
    main()
```

test_pipeline_e2e.py

```
"""
test_pipeline_e2e.py - 하이브리드 파이프라인 통합 테스트 (E2E)
============================================================
이 스크립트는 auto_pipeline.py의 구동 무결성을 안전하게 검증하기
실제 소스 코드 대신 '_test_dummy.py' 파일을 생성하여 파이프라인을
테스트가 완료되면 더미 파일을 정리(삭제)합니다.

Usage:
  python test_pipeline_e2e.py --case full
  python test_pipeline_e2e.py --case nocloud
  python test_pipeline_e2e.py --case search
"""

import os
import sys
import argparse
import subprocess
import json

BASE_DIR = os.path.dirname(os.path.abspath(__file__))
PIPELINE_SCRIPT = os.path.join(BASE_DIR, "auto_pipeline.py")
DUMMY_FILE = os.path.join(BASE_DIR, "_test_dummy.py")

# 테스트용 더미 코드
DUMMY_CODE = '''\
def hello_world():
    prnt("Hello World")  # 의도적인 오타 (prnt -> print)
'''

def setup():
    """테스트 전 더미 파일을 생성합니다."""
    print(f"\n[SETUP] 테스트용 더미 파일 생성: {DUMMY_FILE}")
    with open(DUMMY_FILE, "w", encoding="utf-8") as f:
        f.write(DUMMY_CODE)

def teardown():
    """테스트 후 더미 파일을 삭제합니다."""
    if os.path.exists(DUMMY_FILE):
        print(f"\n[TEARDOWN] 더미 파일 삭제: {DUMMY_FILE}")
        os.remove(DUMMY_FILE)
```

```
def run_pipeline(case: str):
    """auto_pipeline.py를 지정된 케이스로 실행합니다."""

    cmd = [sys.executable, PIPELINE_SCRIPT]

    if case == "full":
        print("\n🧪 [TEST CASE: FULL] 클라우드 ↔ 로컬 완전 자동
        cmd.extend([
            "--task", "hello_world 함수 내의 문법적 오타(prnt)를
            "--target-file", DUMMY_FILE
        ])

    elif case == "nocloud":
        print("\n🧪 [TEST CASE: NOCLOUD] 클라우드 접속 실패 가
        cmd.extend([
            "--task", "이 코드의 개선점을 리뷰해줘.",
            "--target-file", DUMMY_FILE,
            "--no-cloud"
        ])

    elif case == "search":
        print("\n🧪 [TEST CASE: SEARCH] Orchestrator 탐색(Sea
        cmd.extend([
            "--task", "hello_world 함수의 정의 위치를 프로젝트
            "--target-file", DUMMY_FILE
        ])

    else:
        print(f"❌ 알 수 없는 케이스: {case}")
        sys.exit(1)

    print("-" * 60)
    print(f"💻 실행 명령: {' '.join(cmd)}")
    print("-" * 60)

    try:
        # 서브프로세스 실행 (출력 표시)
        result = subprocess.run(cmd)

        print("-" * 60)
        if result.returncode == 0:
            print("✅ 파이프라인 정상 완료 (Return Code: 0)")
        else:
            print(f"⚠️ 파이프라인 에러 종료 (Return Code: {res

    except Exception as e:
```

```
        print(f"❌ 예외 발생: {e}")

def main():
    parser = argparse.ArgumentParser(description="하이브리드 ☁
    parser.add_argument("--case", choices=["full", "nocloud",
                        help="테스트 시나리오 선택 (full = 전체

    args = parser.parse_args()

    if not os.path.exists(PIPELINE_SCRIPT):
        print(f"❌ 오토 파이프라인 스크립트를 찾을 수 없습니다:
        sys.exit(1)

    # 사전 준비
    setup()

    try:
        # 테스트 실행
        run_pipeline(args.case)
    finally:
        # 테스트 종료 후 항상 정리
        teardown()

if __name__ == "__main__":
    main()
```