FlowIt

Multi-Agent 기반 에이전틱 워크플로우 자동화 플랫폼

"암묵지를 시스템속의 스킬로, 스킬을 워크플로우로 AI가 만들고 하네스가 품질을 보장한다"

2026 · 06 · 11

목차

About Us

황대원
PM/ 백엔드/ 프론트엔드

김진형
백엔드

이가원
백엔드/ 프론트엔드

박아름
백엔드

신정혜
백엔드

PART 01

프로젝트 개요

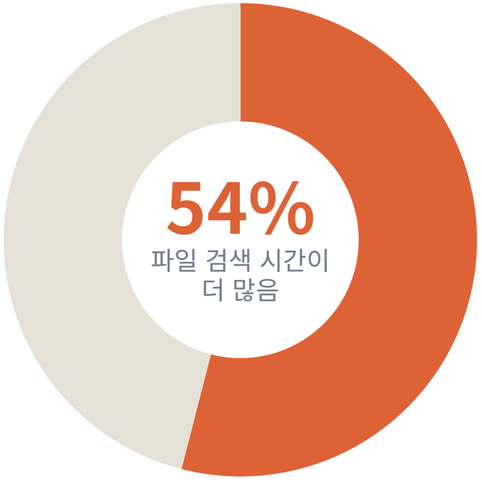
자동화 도구는 늘었지만, 업무 노하우는 여전히 사람에게 갇혀 있다

전체 기술지도 — 요청 처리부터 재사용 학습까지

자연어 요청 → AI 조립 → 자산화 → 학습. 그 중심에 온톨로지 그래프.

파편화된 업무 환경의 현실

직장인의 54%가 실제 업무보다 파일을 찾는 데 더 많은 시간을 사용합니다

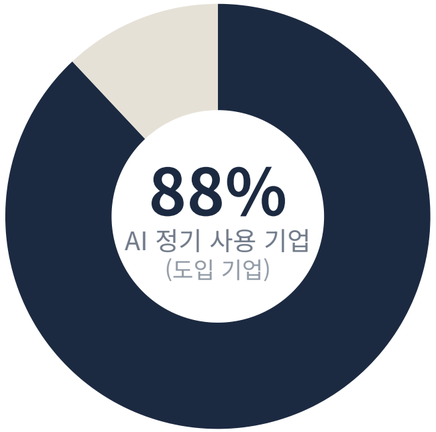


파일 검색 › 맥락 전환 › 시간 낭비 › 생산성 저하

Source: ShareFile Research, 2025

기술 장벽 (비개발자 한계)

88%의 기업이 AI를 도입했지만, 스케일업에 성공한 건 33%에 불과하다



스케일업 성공 33% 미성공 67%

가장 큰 실패 원인 — **통합 복잡성과 모니터링 부재**

통합 복잡성 | 모니터링 부재 | 권한·보안 관리 | 측정 지표 부족

출처: Arcade.dev Enterprise Automation Report, 2025

전문가의 노하우를 팀 전체의 실행 역량으로 확장한다

개인의 업무 방식이 재사용 가능한 Skill이 되고, 팀은 검증된 Skill을 Workflow로 실행한다

01 SOP 기반 Skill 자동 등록

SOP 문서를 업로드하면
AI가 절차를 분석해 재사용 가능한
Skill Node 후보를 생성한다

Skill Node | Metadata

02 Marketplace Skill 공유

도메인 전문가가 검토한 Skill을
Marketplace에 등록해
팀 전체가 재사용한다

Skill Catalog | Marketplace

03 자연어 기반 Workflow 생성·실행

사용자는 자연어로 목표를 말하고
AI는 Skill과 Node를 조합해
Workflow를 생성·실행한다

Workflow Draft | Run Status | SSE Stream

"업무를 직접 조합하는 사람" 에서 **"노하우를 Skill로 공유하고, 팀 전체의 역량을 높이는 사람"** 으로 전환

Agentic AI 기반 업무 자동화 기대효과

국내 정책·적용 자료와 해외 실증연구를 연결해, AX 워크플로우의 생산성 개선 효과를 검증

국내 적용 2023

NIA

공공분야 생성형 AI 활용 방안

아이디어 탐색·문서 초안·업무 자동화·민원 응대 등
실제 공공업무 적용 영역을 제시

효용성과 한계를 함께 진단
공공 활용 시 고려사항 중심 보고서

활용 포인트 **공공 적용 시나리오 정리**

글로벌 조사 2023

Slack

**State of Work 2023
자동화 업무시간 절감 조사**

글로벌 지식근로자 18,149명 조사에서
자동화 활용자의 반복업무 시간 절감 효과 제시

주당 3.6시간 절감
자동화 활용자 평균 응답

활용 포인트 **반복업무 시간절감 근거**

해외 실증 2023

NBER

**Generative AI at Work
고객지원 업무 실증연구**

상담원 5,179명 대상
AI 지원 도구 도입 효과 분석

+14% 평균 생산성

+34% 초보·저숙련

활용 포인트 **생산성 향상 수치 확보**

추정 기대 효과

- 평균 생산성 24% 상승
- 업무 시간 절감 3.6 시간
- 업무 연속성 개선

출처: NIA(2023), Slack State of Work(2023), NBER Working Paper 31161(2023)

국내 적용 → 시간 절감 → 생산성 향상

GitHub PR과 Slack 피드백으로 연결한 협업 흐름

구현 결과를 GitHub에서 검증하고, Slack 피드백을 다음 작업에 반영했다

STEP 01 — REVIEW

Code Review

GitHub PR — 구현 결과 검증

PR 단위로 구현 결과를 공유하고 리뷰 이력을 남겼다.
변경 사항·충돌·Merge 상태를 한 화면에서 추적했다.

구현 결과 검증 · 변경 이력 관리

STEP 02 — FEEDBACK

Feedback

Slack — 피드백 공유·반영

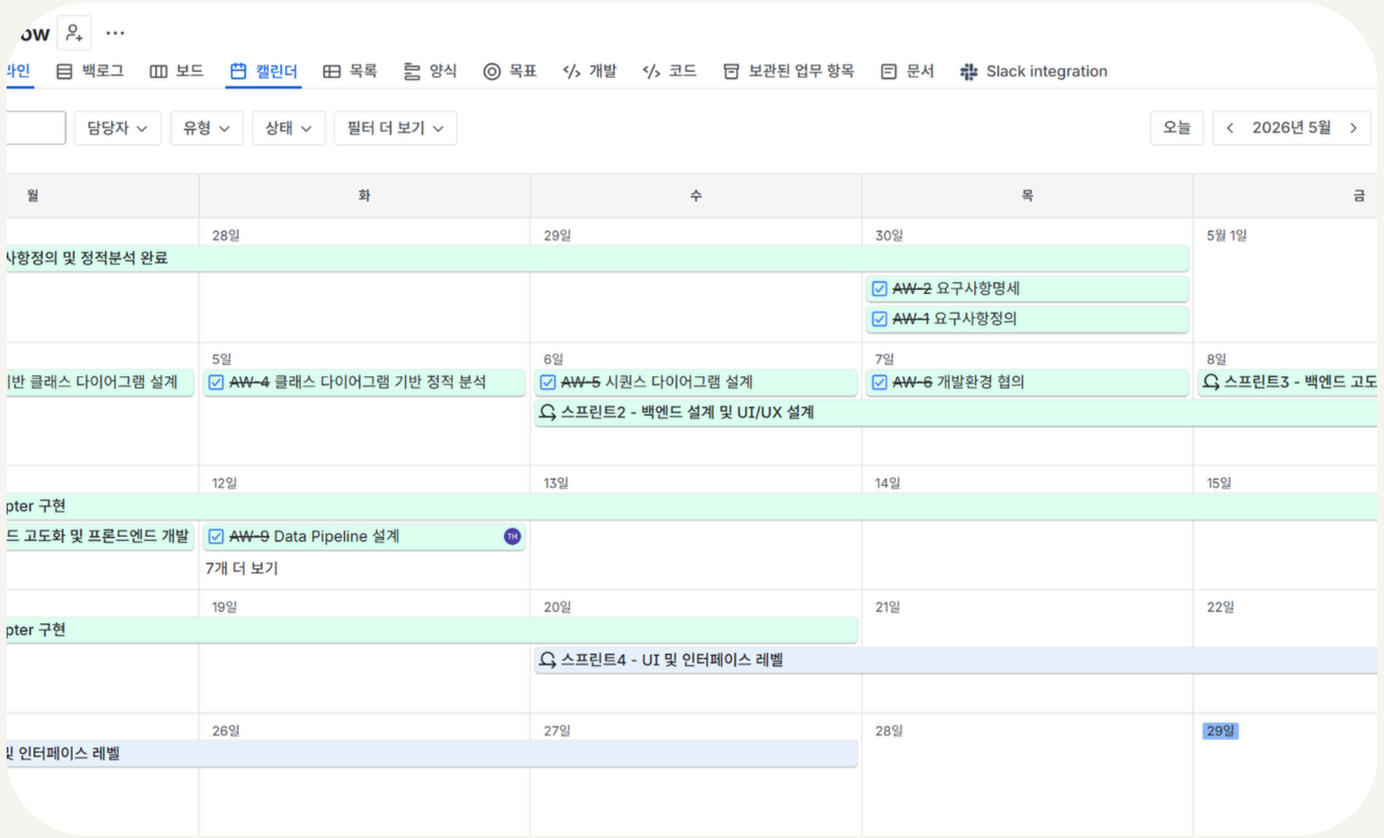
이슈와 리뷰 결과를 빠르게 공유해 의사결정을 정리했다.
확정된 피드백은 다음 개발 작업과 PR에 즉시 반영했다.

빠른 피드백 · 다음 작업 연결

구현 결과는 **GitHub PR**로 검증하고, **Slack 피드백**을 다음 개발 작업에 반영했다.

Jira Calendar로 추적한 실제 Sprint 흐름

REQ 단위 작업을 일정에 배치하고, Sprint별 산출물을 반복 개선했다



1 **SPRINT 1 — 요구사항 정의**

REQ 분석 · 핵심 사용자 흐름 정리 · 정적 분석

2 **SPRINT 2 — 구조 설계**

백엔드 설계 · UI/UX 설계 · 클래스 다이어그램

3 **SPRINT 3 — 핵심 기능 구현**

Workflow Composer · **Skills Builder** · Data Pipeline

4 **SPRINT 4 — 통합 및 보완**

Adapter 구현 · **E2E 테스트** · UI/인터페이스 개선

Jira Calendar를 기준으로 **Sprint 범위와 마감일을 고정**하고, 산출물을 반복 개선했다.

PART 02

개발환경

개발환경

프로젝트에 사용된 언어 · 프레임워크 · 데이터베이스 · AI · 인프라 전체 기술 스택

Language


Python 3.12 TypeScript

Backend




FastAPI Uvicorn Celery Redis

Database


PostgreSQL 16



SQLAlchemy


asyncpg

Neo4j
Graph DB


pgvector

AI


LangGraph Modal GPU

Gemma 4 + BGE-M3

Frontend


Next.js 14


React Flow


Zustand

Schema


Pydantic v2


TypeScript

자동 코드젠 (pydantic2ts)

Code Standard

Python 스타일
Ruff lint (line-length=120) · 타입 힌트 전 함수 필수

타입 시스템
모든 ID 필드 UUID · StrEnum으로 JSON 직렬화 호환

테스트
pytest + pytest-asyncio · 94+ 테스트

하네스 엔지니어링 — 9종 에이전트시스템 아키텍처 명세

TDD 자동화 하네스 Red → Green → Refactor 자동화

2 클로드코드 규칙 강제 — 룰베이스

경계 강제
Clean Architecture 의존성 단방향 · domain import 금지 · 모듈 간 ports만

단일 진실원
common_schemas 단일 정의 · 타입힌트·UUID·UtcDatetime 통일

변경 · 실행 가드레일
development PR·머지 조장만 · permissions 화이트리스트 · .env 금지

경계(Architecture) · 단일원천(SSOT) · 가드레일(Guardrail)

3 컨텍스트 엔지니어링

DOCS 브랜치 프로젝트 내부의 위키피디아 생태계

브랜치 컨텍스트 main·dev·feature·hotfix

타입 일치 보장 백/프론트 불일치 차단

시스템 개요 — Clean Architecture · 모노레포

자연어 요청 → 노드 카탈로그(62클래스 / 62 node_type) 조합 → 워크플로우 자동 생성 → 위상정렬 실행

LAYER 01 — CONTRACT

SSOT

공유 스키마 단일 출처

common_schemas (Python · TS 타입 동기화)

LAYER 02 — MODULES

Hexagonal

도메인 · 포트/어댑터

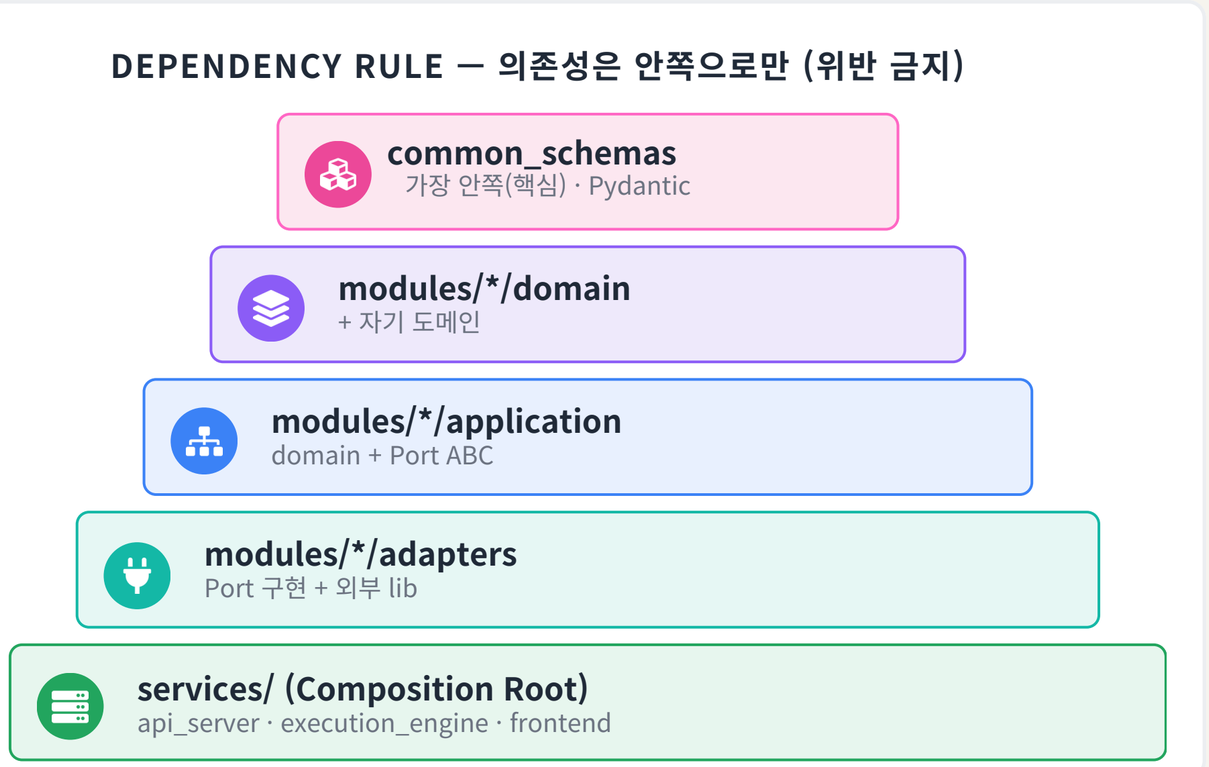
modules/* — domain · application · adapters

LAYER 03 — ASSEMBLY

Mono-repo

서비스 = 조립 루트

api_server · execution_engine · frontend



● **핵심 원칙** 의존성은 항상 안쪽(common_schemas)으로만 → 모듈·LLM·DB 교체 시 핵심 로직 무변경, 테스트 용이

CI/CD · 배포

PR 게이트 통과 → 자동 배포 → 모노레포로 일관 관리

① CI 자동화

PR 게이트 3종 — 통과해야 merge

drift 감지
TS 불일치 차단

pytest
94+ 테스트

Ruff lint
스타일 검사

→

② CD 파이프라인

git push → 자동 배포

키 없는 배포
Workload Identity

→

Cloud Run
dev→stg / main→prod

→

modal deploy
에이전트 재배포

③ 모노레포 구조

4개 워크스페이스 분리

common_schemas/

공유 타입 SSOT
Python + TS

modules/

도메인 모듈
ai_agent · auth · nodes_graph ···

services/

배포 단위
api_server · frontend · agents

database/

순수 SQL
DDL · 마이그레이션

PART 03

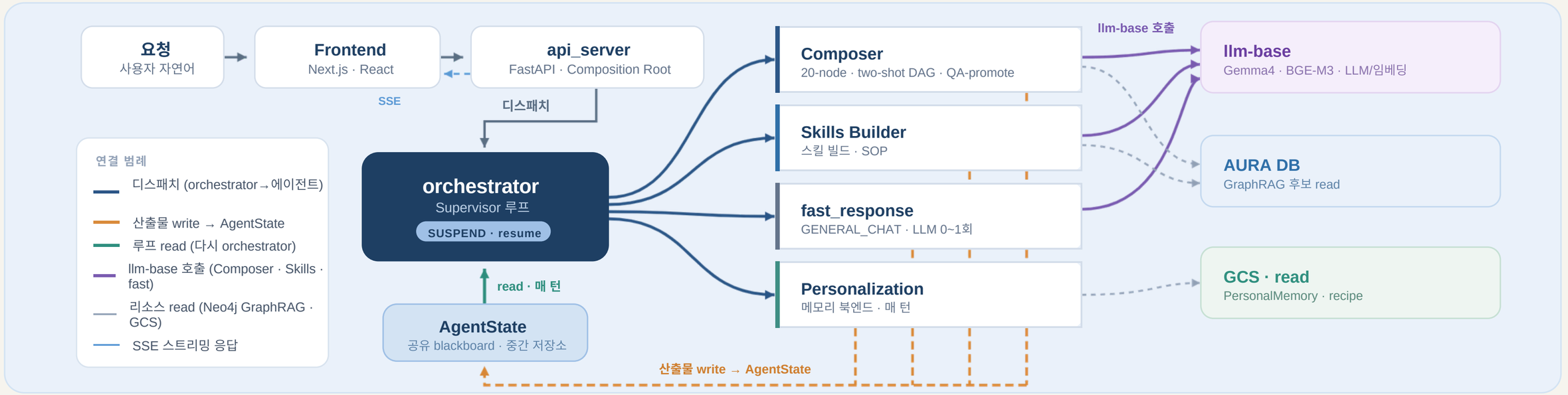
아키텍처 설계

Clean Architecture · DIP · AI Native Engineering · 모노레포 의존성 · GCP 배포.

Agentic AI Workflow System - 전체 아키텍처

요청 → 생성(Supervisor 멀티에이전트 루프) → 위상정렬 실행까지, 전 모듈이 어떻게 이어지는가

① 생성 · GENERATION — Supervisor 멀티에이전트 루프



② 산출물 → 도착 DB 쓰기 · 색 = 도착지 (쓰기 전용)

GCS · 파일 · 세션

초안 · 세션/재개상태 · 스킬 문서 · 개인메모리

PostgreSQL 16 + pgvector · 정형 · 벡터

workflows · skills · agent_memory(개인메모리) · document_chunks(768d · HNSW) · executions · node_logs

AURA DB · 관계 · 그래프

(:Skill)-[:BINDS]->(:Node) · 게시 BINDS · GraphRAG 후보

③ 실행 · 위상정렬 workflow_id → 위상정렬 실행 → 실행 결과 → executions (PostgreSQL)

흐름 핵심 · 요청 → Supervisor 루프 생성 → 산출물별 도착 DB 저장(파일/세션 GCS · 정형/벡터 PostgreSQL · 관계 Neo4j) → 위상정렬 실행 → executions

데이터 파이프라인

PostgreSQL 16 · pgvector — Raw → 변환 → 저장 → 이벤트의 4계층 흐름

4계층 데이터 흐름

SOURCES
① 입력
OAuth 토큰 · 자연어 발화 · 파일 업로드(SOP) · 스킬 요청

TRANSFORM
② 변환
토큰 암호화(AES-256-GCM) · 의도분석+노드검색 · 파서+PII 마스킹 · BGE-M3 임베딩(768d)

STORAGE
③ 저장 · PG16+pgvector / GCS
credentials(BYTEA) · node_logs(월 파티션) · document_chunks(vec768·HNSW) · skills 3-scope · executions(실행 결과) · GCS

EVENT
④ 이벤트
상태변경 → Redis Pub/Sub → SSE → 프론트(실시간) · GCS 세션 이력

핵심 파이프라인 (코드 검증)

1. **인증**
   OAuth 토큰 → AES-256-GCM 암호화 → credentials(BYTEA)
2. **워크플로우 생성**
   자연어 → 의도분석 → 노드검색(벡터) → QA → workflows
3. **문서 파싱**
   PDF/DOCX/HWP → PII 마스킹 → 청킹 → BGE-M3 768d → document_chunks(HNSW)
4. **스킬 마켓**
   임베딩 → 3-scope 승격(개인→팀→회사) → skills(HNSW)
5. **실시간 이벤트**
   상태변경 → Redis Pub/Sub → SSE → 프론트
6. **온톨로지 투영**
   노드 카탈로그(Postgres) → Neo4j 재투영 → CAN_FOLLOW 관계 계산
   + 스켈레톤 HAS_SLOT·FILLED_BY · 스킬 BINDS

- **벡터 검색** pgvector HNSW (cosine) — 전 테이블(skills · document_chunks · node_definitions) **vector(768) 통일** · FTS 하이브리드 아님

인프라

GCP · Modal · Terraform · IaC — 콘솔 클릭이 아니라 코드 리뷰로 관리되는 기반

개발자 — git push → **terraform apply** (콘솔 클릭 ✗ · Git 이력 · 리뷰 추적)

① **GCP PROJECT** · Cloud Run (Terraform 관리)

Artifact Registry
컨테이너 이미지

이미지 배포

CLOUD RUN · 서비스 컨테이너

Frontend	API Server	Worker
Next.js·공개	FastAPI·공개	Celery·내부

secret 주입 / Private IP·VPC

Secret Manager ×17
neo4j 3 · db · oauth · *-url

DATA LAYER · Private IP
Cloud SQL(PG16) · Memorystore(Redis) · GCS

★ ② **Modal — AI 에이전트 서빙** (GPU 서버리스 · 별도 배포)

orchestrator · composer · skills-builder · personalization · llm-base
Gemma4 · BGE-M3 ◀ api_server [orchestrator-url 시크릿]

★ ③ **Neo4j AuraDB**
온톨로지 그래프 · 외부 SaaS
◀ neo4j-* 시크릿 연결

SA 최소 권한 격리 (서비스별 전용 · 3계정)

API · workflow-api-{env}-sa
DB ✓ · Secret ~15 · GCS ✓ · Worker ✗

Worker · workflow-worker-{env}-sa
DB ✓ · Secret 7 · GCS 암호화 키만 · Worker —

Frontend · workflow-frontend-{env}
DB ✗ · Secret 0 · GCS ✗ · Worker ✗

SA 1개 유출돼도 다른 서비스 영향 없음 (권한 격리)

Terraform 7 모듈 (코드 검증)

artifact-registry · cloud-run · cloud-sql · gcs · memorystore · networking · secret-manager
envs: staging / production

수동
콘솔 클릭 → 재현·추적·일관성 어려움

IaC (Terraform)
terraform apply 한 방 · Git 이력 · 동일 모듈+변수

***-url 시크릿** = GCP ↔ Modal을 잇는 접착제 · 한 시스템이 세 곳에 분산

● **운영 · 보안** SA 최소 권한 3계정 격리 · 콘솔클릭 ✗ → **terraform apply** · AI 에이전트 = **Modal**, 데이터 = **GCP·Neo4j**

PART 04

핵심 기술 소개

자연어 요청이 자동화 자산이 되는 6단계 구조 — 이제 온톨로지 그래프로 더 똑똑하게.

① 프레임 · ② Composer · ③ 노드 · ④ Skills Builder · ⑤ 마켓플레이스 · ⑥ Personalization + ⭐ **온톨로지 GraphRAG**

전체 기술지도 — 요청 처리부터 재사용 학습까지

자연어 요청 → AI 조립 → 자산화 → 학습. 그 중심에 온톨로지 그래프.

⚙️ ① 프레임 — REQ-004 · 신정혜

Main + 3 Sub-Agent · **5 Modal 앱**

Supervisor · 9 intent 결정론적 라우팅

AgentProtocol · VPC 내부 HTTP

★ 장애 격리 · 독립 배포·스케일

🎨 ② Composer — REQ-004 · 신정혜

LangGraph + GraphRAG · **22노드 fixed DAG**

vector seed → 그래프 확장

2중 검증 — LLM + 코드 (QA≥8)

★ 끊긴 워크플로우 23%→0%

🧩 ③ 노드 — REQ-003 · 박아름

62종 · 8 카테고리 · 53→62 대칭화

GraphValidator 7종 (유한순환 포함)

BGE-M3 768d · pgvector HNSW

★ 실행엔진과 검증 1:1 미러

🛠️ ④ Skills Builder — REQ-013 · 박아름

문서 → 스킬 · **tool 5종 + 결정적 스켈레톤**

2단계 추출 (메타→상세)

HITL 게이트

★ intent=build_skill · 9섹션 런북

🏪 ⑤ 마켓플레이스 — REQ-013 · 박아름

산업 6(활성 1)·직무 5 · **3 scope · 5 state**

RBAC · fail-closed

2-md (SKILL.md + COMPOSER.md)

★ 승격=복제 · publish 시 BINDS 투영

💾 ⑥ Personalization — REQ-005 · 이가원

GCS + CAS · MEMORY.md + 토픽 + 임베딩

if_generation_match 원자적 선점

debounce 5분

★ 노이즈 5.8배 절감 · min_score 0.5

자연어 요청 → ① 프레임 → ② Composer 조립 (← ③ 노드 · ⑤ 기존 스킬 · ⭐ 온톨로지 확장) **→ ④ Skills Builder → ⑤ 마켓 저장 ↻ 재사용 → ⑥ 학습**

Supervisor — 단방향이 아니라 루프로 도는 오케스트레이션

LLM은 intent 분석에만 · 라우팅은 결정론적 — 그리고 끝나면 다시 supervisor로 돌아온다.

라우팅 루프

🔁 결과 반환 → 다음 턴

사용자 발화 → **Supervisor · intent 분류 (9종)**

- → 즉시 응답 (chitchat · info · control · workflow_execute)
- → **Composer relay** (draft · refine · clarify)
- → finalize (propose)
- → **Skills Builder relay** (build_skill)
- → Monitoring (AgentState 6상태)

↺ sub-agent SSE 스트리밍으로 **결과 반환 → 다시 Supervisor로 복귀** → 다음 턴 라우팅 — **대화가 루프로 누적**

↗ **단방향이 아니다 — 매 분기 결과가 supervisor로 되돌아오는 닫힌 루프**

복합 발화 skill_then_compose → build_skill 완료 후 compose로 **체이닝** (루프의 실제 예)

9 INTENT · IntentType enum

chitchat · info_question · control · workflow_execute · draft · refine · clarify · propose · build_skill

LLM 없이 intent로 분기 = **fast-path** (즉시 응답군은 sub-agent 미호출).
AgentState 6상태 = Monitoring 루프.

5 Modal 앱 독립 배포 · 장애 격리

- orchestrator · Supervisor · intent 분류·라우팅
- agent-composer · 워크플로우 생성 (LangGraph+온톨로지)
- agent-skills-builder · 문서 → 스킬 (HITL)
- agent-personalization · 메모리 (GCS+CAS)
- llm-base · Gemma 4 + BGE-M3 공통 추론

VPC 내부 HTTP · AgentProtocol Pydantic v2 SSOT · scaledown 300s

자연어 요청을 22개의 툴로 워크플로우를 조립하는 에이전트 - 계획·도구·실행·평가의 한 사이클

5단계 전처리 → 분석 → 검색·생성 → 검증·재시도 → 확정·저장 = 멀티에이전트 4요소로 동작

① **PLANNING** 계획
intent → consultant → slot_fill
요청 해석 + 슬롯 계획

② **TOOL-CALLING** 도구 호출
search_nodes · suggest_skill_select
재료를 그래프에서 — ③ 62종 + ⭐ 온톨로지 · ⑤ 마켓

① **PLANNING** 계획
bind_skill → draft_workflow
요청 해석 + 슬롯 계획

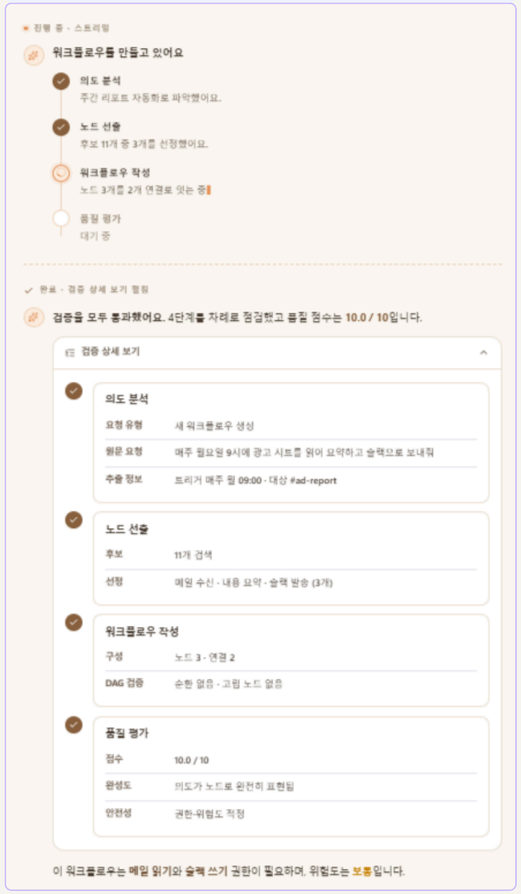
③ **ACTION** 실행
intent → consultant → slot_fill
연결 가능한 노드만으로 조립

↺ Evaluation → Planning 재시도 루프

④ **EVALUATION** 평가 🔁
validate_workflow + qa_evaluator
2중 검증 · QA < 8 → retry_draft (최대 3회)

이전 줄글 덤프

이후 단계별 = 4요소

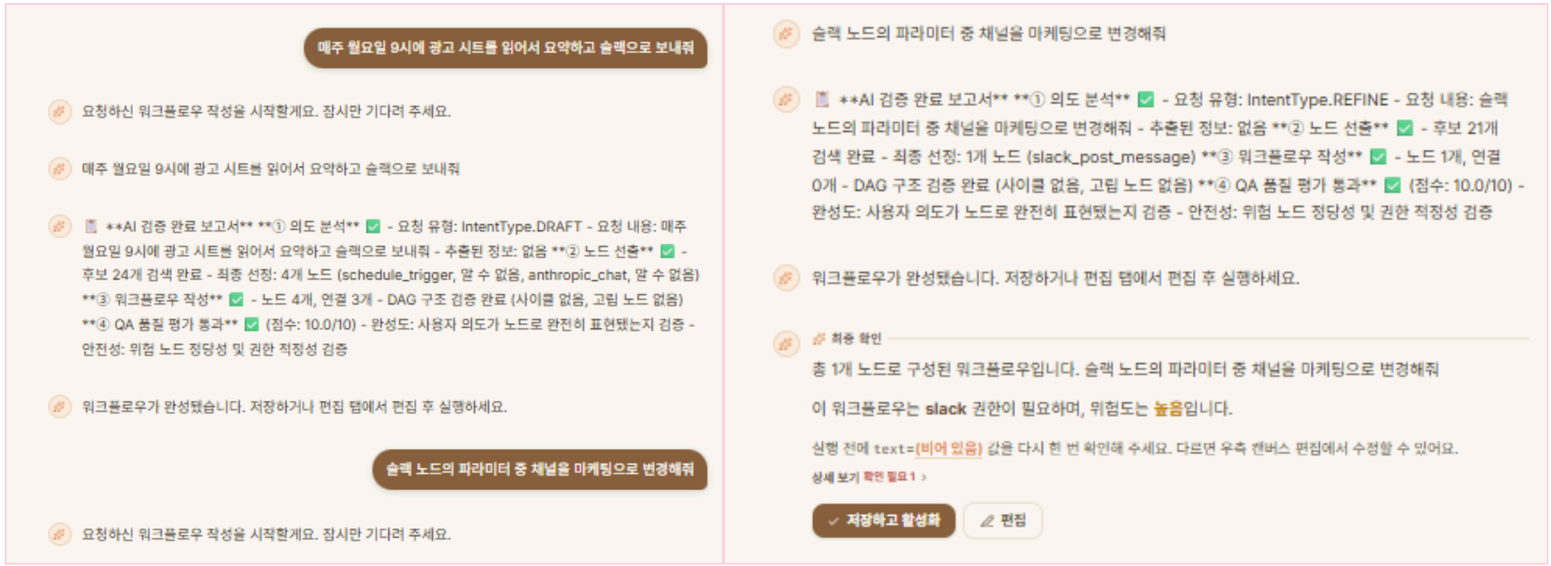


이후 캡처의 **의도분석 · 노드선출 · 워크플로우작성 · 품질평가** 가 곧 **Planning · Tool · Action · Evaluation**

온톨로지 - 워크플로우 품질 보장을 위한 하네스 엔지니어링의 출발점

top-k 임베딩만으로는 노드 호환성·필수 연결·반복 패턴을 표현할 수 없다. · 동일 발화 “매주 월요일 9시에 광고 시트를 읽어서 요약하고 Slack으로 보내줘”

온톨로지 도입 전 평탄 벡터 검색



온톨로지 도입 후 평탄 벡터 검색

🔴 실패 원인 (PAINPOINT)

① 호환성·연결 구조를 몰라 google_sheets_read를 못 찾고 **“알 수 없음” ×2**로 채움

② 실행 불가능한 조합인데도 **QA 10/10 통과** — 검증이 못 걸러냄

🟢 **google_sheets_read** 정확히 선정 (+ http_request 보강) **— 실제 실행 가능**

같은 발화 · 같은 QA 점수(10/10)인데 — 도입 전은 **“알 수 없음”이 통과**, 도입 후는 **정확한 노드**. 차이는 **노드 선택의 정확성**이다.

vector seed → graph expand(1-hop) → 제약 후보 → 그래프 안에서만 생성

에이전트에게 하네스(작업지침서)를 전달하기 위한 GraphRAG와 ground truth

벡터는 seed로 그대로 쓰고, Neo4j는 "다음에 올 수 있는 노드" 구조 확장만 — ground truth는 허용 node_type 집합으로 결정적 제약이 된다.

도입 후 구조 vector seed pgvector → 1-hop expand CAN_FOLLOW → ground truth 허용집합 → 제약 후보 → 결정적 조립 skeleton → 검증 OntologyRetrieverPort · Neo4jOntologyAdapter

🧩 결정적 스켈레톤 라이브러리

8 종

skeleton_library.py → Neo4j (:Skeleton)

- scheduled_pipeline
- event_response
- quality_loop
- branch_on_classification
- conditional_action
- fan_out_map
- retry_backoff
- approval_gate

동작: intent에 "검증·분류·분기·반복" 감지 → 스켈레톤 슬롯을 실제 node_type으로 결정적 바인딩 — **구조는 코드, LLM은 파라미터만.**

2중 검증

⚖️ 품질 검증

초안 단계 · LLM-as-Judge

의도 반영 평가 → score ≥ 8.0

→ promote FAIL → retry_draft (최대 3회)

추적 · motif-correctness

🛡️ 실행 검증

실행 직전 · GraphValidator 7종

비실재 · 중복 ID · 사이클 (유한순환 허용)

· 타입 호환 · 고립 · 필수 연결 · 필수 파라미터

엔진 실행과 동일 기준으로 사전 차단

의도 반영 → 품질 평가 score ≥ 8.0 → promote FAIL → retry ×3

motif-correctness로 스켈레톤 조립 정확도 추적

도입 전 vs 후 — 측정으로 증명한 온톨로지 효과

"측정 없는 고도화는 추측" — §6.5 평가 하니스로 Before/After 실측.

지표	BEFORE 벡터만		AFTER GraphRAG
끊긴 워크플로우율	23.2%	→	0.0%
qa_pass	0.45	→	0.75
motif-correctness	75%	→	100%
노드 풍부화	3.0	→	3.25

실패에서 배운 것

- × **soft 힌트만 = 효과 0** — 작은 LLM은 힌트로 구조를 안 바꾼다
- × **ADD-all 확장 = 회귀** — 후보 비대
  → Gemma JSON 잘림 (drafter 실패 6→23, qa 64→29%)
- ✓ **cap 필수** (seed 5 + add ≤3) ·
  구조는 코드가 강제, LLM은 파라미터만

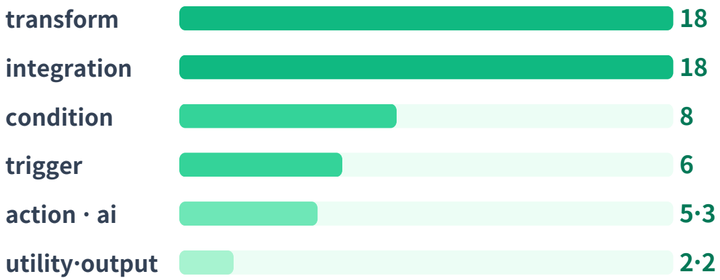
연결 **S5(정성)** "알 수 없음 노드가 통과 vs google_sheets_read 정상" → **S7(정량)** qa 0.45→0.75 — 두 장이 "노드 선택 정확성"을 정성+정량으로 받친다.

노드 · 노드 그래프 · 실행 엔진 — 셋이 맞물려 동작한다

검증(노드 그래프)이 곧 실행(엔진)의 거울 — 같은 기준으로 검사하고 실행한다.

① 노드

62 · 8 카테고리



선정 — MVP 핵심 자동화 단위 선별 · **read/write 대칭(+9)**

② 노드 그래프

Validator 7종

- ✓ 비실재 노드 · 중복 ID
- ✓ 사이클 (Tarjan) — **유한순환 허용**
- ✓ 고립 노드 · 필수 연결
- ✓ 필수 파라미터 (input_schema)
- △ 타입 호환 — 검증 전용(엔진 미구현)

6종 = 엔진 실행과 1:1 미러 · 타입 호환만 검증 전용. 사이클 수용 기준은 검증=실행 동일 판정(parity 테스트).

③ 실행 엔진

TopologicalScheduler
비순환 — 위상정렬

CyclicScheduler
Tarjan SCC → LoopBody · back-edge 판정 → 반복 · max 10

↻ 조건 노드 ≥1 = is_brancher → 분기·반복

왜 이렇게 짰나 — 조건문·제어문·검증 루프를 모두 지원하려고. 단순 DAG가 아니다.

업무 문서를 재사용 가능한 스킬로 — 구조는 코드가 결정한다

현재 = 결정적 스켈레톤 위저드(가동) · 목표 = tool-calling 루프(프레임 대기). 어느 쪽이든 구조는 코드가 결정.

멀티에이전트 4요소 — Composer와 동일 프레임

① PLANNING **입력·스킬 결정**	분석 문서 선택 / seed 합성 → 무엇을 만들지	✅
② TOOL-CALLING **콜러블 5종**	오른쪽 **콜러블 5종** 참조 →	**3 가동 / 5**
③ ACTION **confirm → DRAFT**	2-md GCS 저장	✅
④ EVALUATION **HITL 게이트 🔁**	사용자 검토·선택 (카탈로그 오염 차단) + QA	✅

정직 표기: 5종 콜러블은 준비됨 — 에이전트 루프(wrap)는 프레임(O1) 결정 후. **현재는 결정적 위저드가 이 콜러블을 직접 호출.**

콜러블 TOOL 5종

✅	**extract_skill_candidates**	메타 추출
✅	**search_skeleton**	SOP→스켈레톤·결정적
✅	**assemble_skill**	결정적 슬롯 채움
⏳	**search_user_documents**	storage 어댑터 대기
⚠️	**parse_document**	호출 자리 없음·재검토

§ 6.6 원칙 상속 (Composer가 입증)

Composer가 입증한 온톨로지 효과 — **Skills Builder도 같은 원칙 상속**

끊긴 워크플로우 23% → 0%

qa_pass 0.45 → 0.75

soft 힌트 = 효과 0 · **구조는 코드가 강제**

⚠ 수치는 Composer 측정값 · SB는 온톨로지 Phase 2 배선 후 자체 측정

in-code 결정적 조립에서, 온톨로지 ground truth로

지금은 키워드 매칭으로 스켈레톤을 박는다 — 다음은 그래프가 도메인 정답 구조를 안다. 단, 그래프는 "결정적 구조 선택"에만.

현재 · 가동 ✅

in-code 결정적 조립

- skeleton_library.py·**SKELETONS 8종**(도메인 무관 범용 모티프)
- SkeletonEntityExtractor·**키워드 매칭**
- SkeletonComposerMapper→ COMPOSER.md + 정밀 BINDS

한계 — 도메인 제약·필수·금지 규칙층이 없다.
현 그래프엔 REQUIRES + SlotSpec.required 뿐.

목표 · ➡️ SOON § 8

온톨로지 통합 3 지점

① **도메인 스켈레톤 + ground truth (필수·금지)**

```
(:Domain)-[:USES_SKELETON]→(:Skeleton)
(:Domain)-[:REQUIRES_NODE]→(:Node)
(:Domain)-[:FORBIDS]→(:Node) ← 음수 제약(신규)
⚠ :Domain / :FORBIDS 는 미확정 제안 — 명명 합의 필요
```

② **스킬 후보 임베딩 → GraphRAG → SKILL.md**
BGE-M3 · EmbedderPort 재사용 → 도메인 최적 구조 결정

③ **composer와 동일 온톨로지 조회 → COMPOSER.md**
OntologyRetrieverPort.expand_candidates · 같은 Neo4j 1개 공유

⚠ :Domain/:FORBIDS 미확정 — 명명 합의

✅ Neo4j 1개 DB · 투영 경로 2개 (2026-06-11 확인)

⚠️ **§ 6.6** GraphRAG 회수를 **LLM 힌트로만 = 효과 0**. 반드시 **"결정적 구조 선택"**에 사용 — LLM은 파라미터·설명만. QA_PASS 0.45 → 0.75

같은 SOP, 같은 모델 — 프롬프트만 바꿔 런북 수준 지침서로

소형 LLM(Gemma 26B)도 “무엇을 시키느냐”를 9섹션으로 구조화하면 Claude Skill급 SKILL.md. · 2026-06-10 무배포 실측

입력 동일 현실적으로 얕은 **7줄짜리 온보딩 SOP.docx** · 같은 모델 Gemma 26B

BEFORE 3섹션 권장

강제 아님

① description “무엇을” 1문장뿐 → 트리거 부족
② 섹션 권장일 뿐 → 형식·깊이 불안정
③ 얕은 SOP → 얕은 결과

```
## When to use
온보딩 안내문 작성 시.
## Steps
1. 정보 확인 2. 안내문 작성
## Inputs/Outputs
입력: 입사자 정보 / 출력: 안내문
```

그럴듯하지만 실행 못 하는 “한 줄짜리” 지침

AFTER 9섹션 강제

같은 Gemma 26B

① description(트리거) **② 목적** **③ 언제 사용(+안 함)** **④ 사전 조건** **⑤ 처리 절차** **⑥ 판단 규칙** **⑦ 입력/출력** **⑧ 예시(정상/엣지)** **⑨ 제약·주의**

```
# 환불 요청 매니저 알림
## 목적
환불 접수 시 매니저 빠른 대응 자동 통보.
## 판단 규칙
금액 5만원 초과면 ‘승인 필요’ 표기.
## 제약·주의
민감정보 있으면 원문 노출 금지, 분류만.
```

비전문가도 따라 하는 런북 — ① description 한 줄로 호출 판단

“모델이 아니라, 시키는 법을 바꿨다” 구조(9섹션)는 프롬프트가 강제 · 내용은 LLM이 채움 · 2단계 추출(메타→상세)

⚠ 9섹션은 무배포 실측 — 코드 반영은 후속 (현 build_from_sop는 3섹션)

한 번 만든 스킬이 다음 워크플로우를 더 빠르게 만든다

SOP·SEED → 2종 지침서 → 마켓 → 재사용. 자기 강화 루프.

SEED 표준 스킬

검증된 시드 자산

산업	**6** (활성 1)
직무	**5**

customer_support · it_ops · document_data · hr · marketing

company scope 자동 PUBLISHED · uuid5 idempotent

2-MD 합성

스킬당 두 지침서

SKILL.md
노드 **실행** 시 LLM 주입

COMPOSER.md
워크플로우 **생성** 시 drafter 주입

같은 스킬, 소비처가 다름

재사용 루프

자기 강화

1. **Skills Builder** SOP → DRAFT
2. ⑤ **마켓플레이스** 자산 축적
3. ② **Composer** suggest_skill 재사용
4. **조립·학습** ↶ 새 SOP → 재투입

company scope 자동 PUBLISHED · uuid5 idempotent · composer_instructions 배선

개인 자동화를 팀·전사 재사용 자산으로 확산

한 번 만든 자동화를 개인 → 팀 → 전사 범위로 안전하게 확산.

🏪 3 SCOPE · 승격 = 복제

personal
owner only
개인 작업 공간 · 본인만 접근

›

team
team_manager + dept
팀·부서 단위 공유 자산

›

company
company_manager
전사 표준 자동화 자산

♻️ 5 STATE LIFECYCLE · SKILLSTATE

DRAFT 초안 → submit → **REVIEW** 검토 → approve → **APPROVED** 승인 → publish → **PUBLISHED** 게시 → archive → **ARCHIVED** 보관

RBAC · fail-closed · AuthorizationError

ApprovalWorkflow 감사 · ADR-0020

벡터 검색 + scope/lifecycle 필터 **(+ BINDS 투영)**

하나의 스킬, 두 종류의 지침서

시스템이 실행할 메타 + LLM이 읽을 지침 — 분리 저장.

실행용 메타데이터

NodeDefinition · DB

PostgreSQL · node_definitions

- JSON Schema (**input / output**)
- risk_level · node_type
- 쿼리 · 승격 · 카탈로그 진입
- ADR-0024: 게시 시 신규 생성 폐기 → **검색 SSOT는 skill embedding**

2-md

분기 저장
ADR-0024

LLM / 사용자용 지침서

2-md · GCS

gs://{bucket}/skills/{id}/

SKILL.md
frontmatter + body → 노드 **실행** 시 주입 (execution_engine)

COMPOSER.md
composer_instructions → 워크플로우 **생성** 시 주입 (drafter)

confirm → SKILL.md + COMPOSER.md GCS 저장

publish → 검색 임베딩 + 온톨로지 (:Skill)-[:BINDS]->(:Node) 투영

GCS 개인화 RAG — 노이즈 5.8배 절감

쌓인 패턴을 통째로 넘기던 방식 → 발화와 관련된 것만 골라 주입

이전 방식의 문제

사용자 패턴이 쌓일수록 무관한 정보까지 AI에게 **통째로 전달**
→ 엉뚱한 노드 선택과 토큰 낭비
("주간보고" 요청에도 메일·시트 패턴까지 다 주입)

바뀐 가치 — 평소 습관대로

발화 의미와 가까운 패턴만 **선별 주입**
매번 설정을 반복하지 않아도 본인 습관(요일·형식)대로 워크플로우를 받습니다.

정답률은 그대로, 잡음만 제거

hit율 100% 유지 하면서 무관 패턴 주입을 5.8배 감소
업무와 무관한 잡담은 완전히 차단 합니다.

BEFORE / AFTER

전체 덤프 방식 → RAG 선별 주입의 실측 비교

지표	BEFORE · 전체덤프	AFTER · RAG
hit율 (정답 진입)	100%	**100%**
평균 노이즈 (무관 주입 수)	4.83	**0.83**
잡담("점심") 반환	6개 전량	**⌀ 차단**
정답·잡담 분리 마진	—	**+0.181**

평균 노이즈 **4.83 → 0.83** ▼5.8×

측정 : 6종 페르소나 코퍼스 · 7개 발화(업무 6 + 잡담 1) · BGE-M3 768차원
신뢰 : 라이브(실 GCS+Modal) ≡ 오프라인 골든 소수점까지 일치 · recall 0.32s/발화 · 골든 스냅샷 CI 회귀 10건 통과

데이터로 정한 운영값 — min_score 0.5 · top_k 3

정답을 놓치지 않으면서 잡담은 거르는 "안전대"를 실측으로 탐색

MIN_SCORE SWEEP · TOP_K = 3 고정

게이트 값을 0.3→0.7로 올리며 정답·잡담 거동을 실측

MIN_SCORE	HIT율	노이즈	잡담 게이트
0.3	100%	1.83	❌ 뚫림
0.4	100%	1.83	✓
0.5 운영값	100%	0.83	✓
0.6	50%	0.17	✓
0.7	17%	0.00	✓

- **0.3↓ 금지** — 잡담에도 패턴이 새어 들어감
- **0.4~0.5 안전대** — 정답 100% + 잡담 차단 동시 충족
- **0.6↑ 정답 손실** — hit율 50%→17% 급락

SEPARATION DISTRIBUTION

정답(primary) 최저	0.553	doc_tool_hwp
정답(primary) 최고	0.754	meeting_notes_notion
잡담(distractor) 최고	0.372	—
분리 마진	**+0.181**	

← 게이트 0.5가 이 골(0.372 ↔ 0.553)을 통과

top_k = 3인 이유

"보고서 만들어줘" 실측 순위 → 1위 비정답 **(0.608)**, 2위 정답**(0.553)**, 3위 비정답 **(0.548)**

top_k=1이면 정답이 잘리고 , top_k=2는 3위와 간격 0.005로 취약 → **3으로 안전마진 확보**

한계

정답 최저**(0.553)**와 게이트**(0.5)** 헤드룸이 **+ 0.053로 얇음**.

코퍼스가 크게 늘면 일부 약한 정답이 걸릴 수 있어 **재측정 트리거**로 보존

개인화 주입 A/B — 같은 발화, 파라미터를 사용자 값으로 채운다

personal_RAG OFF는 임의 placeholder라 경고 / ON은 사용자 패턴값으로 채워 경고 없음. · 동일 발화 “매주 월요일 9시에 광고 시트를 읽어서 요약하고 Slack으로 보내줘”

OFF corpus 없음 · 대조군

🔴 **spreadsheet_id = YOUR_SPREADSHEET_ID** (임의 placeholder) → “다시 확인” 경고

ON 패턴 주입 · 자동 충전

🟢 **spreadsheet_id = flowit01** (사용자 패턴값) → AI 자동 충전, 경고 없음

같은 발화인데 — OFF는 YOUR_SPREADSHEET_ID(빈 placeholder·경고), ON은 flowit01(과거 패턴에서 회수한 실제 값). **별도로 말 안 한 값도 사용자 습관대로 자동으로 채운다.**

PART 05

실제 동작 흐름

E2E 시퀀스 · LLM Supervisor · Tool-Calling 파이프라인 · SSE 프레임 매핑 · HITL Gate

전체 시스템 동작 흐름 — End-to-End

사용자 메시지 입력부터 워크플로우 실행 완료까지의 전체 시퀀스

STEP 01 — CLIENT TO API

SSE

사용자 요청 → API Server

ENDPOINT POST /ai/compose

STEP 02 — ROUTING

Orchestrator → Sub-Agent

- Composer
- Skills Builder
- Personalization

STEP 03 — EXECUTION

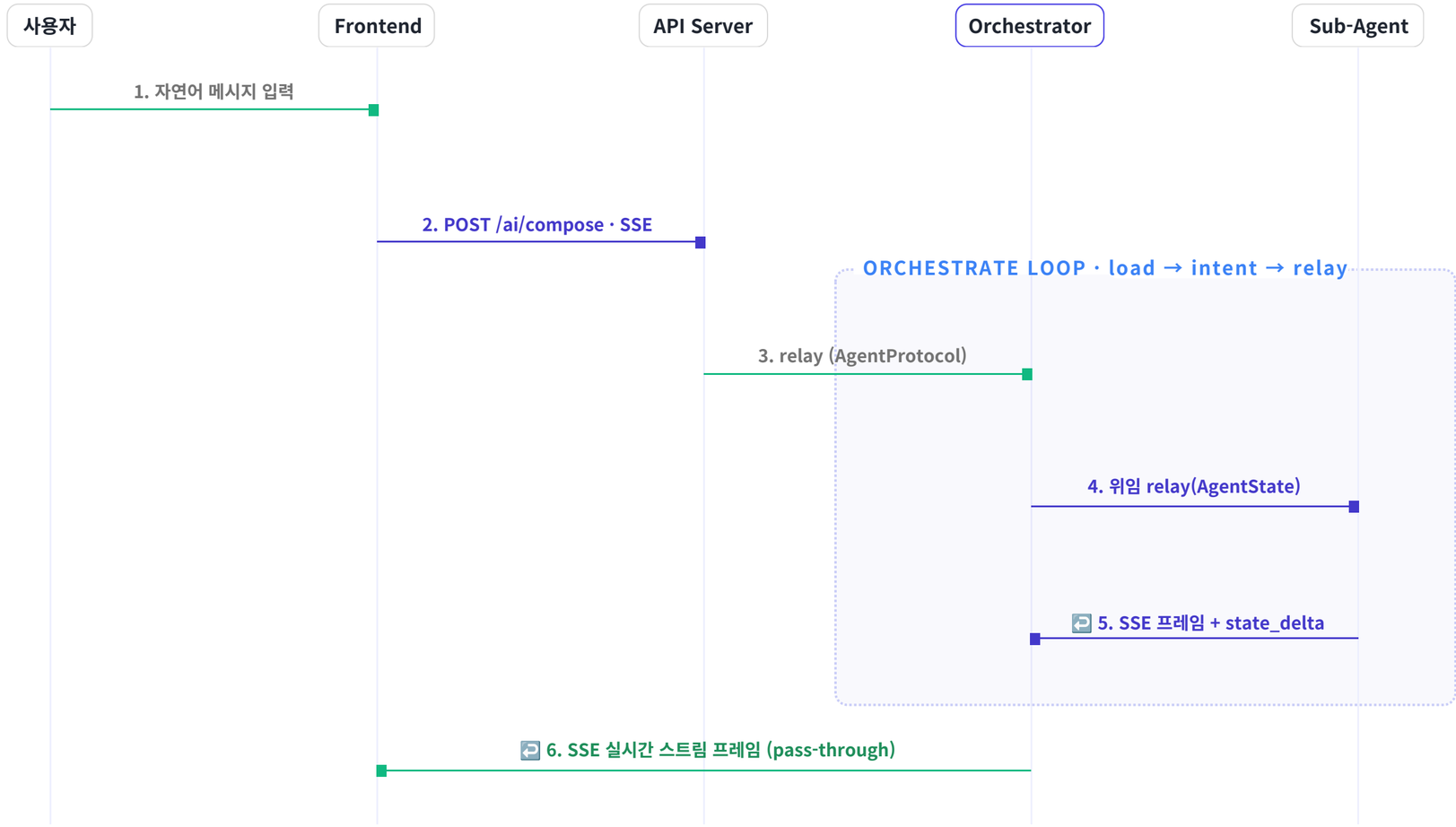
HITL

승인 → 병렬 실행 → 결과

SCHEDULER TopologicalScheduler

E2E EXECUTION SEQUENCE

생명주기별 에이전트 간 비동기 소통 구조



💡 **핵심** - Next.js를 프록시로 두어 AI 엔진의 SSE 프레임을 실시간 중계하고, Orchestrator는 의도에 따라 Composer · Skills Builder를 동적 위임한다. 그래프 DB(Neo4j)는 노드 조합 그라운딩에 쓰인다.

Main Orchestrator — Supervisor 패턴

LLM Supervisor가 Sub-Agent를 호출하고, 결과를 받아 계속 / 중단을 제어한다.

결정형 라우팅

LLM 없이 재현 가능

의도 분석만 LLM,
라우팅·제어는 순수 함수 **route / recovery_target**

STATE-MEDIATED 핸드오프

직접 통신 없음

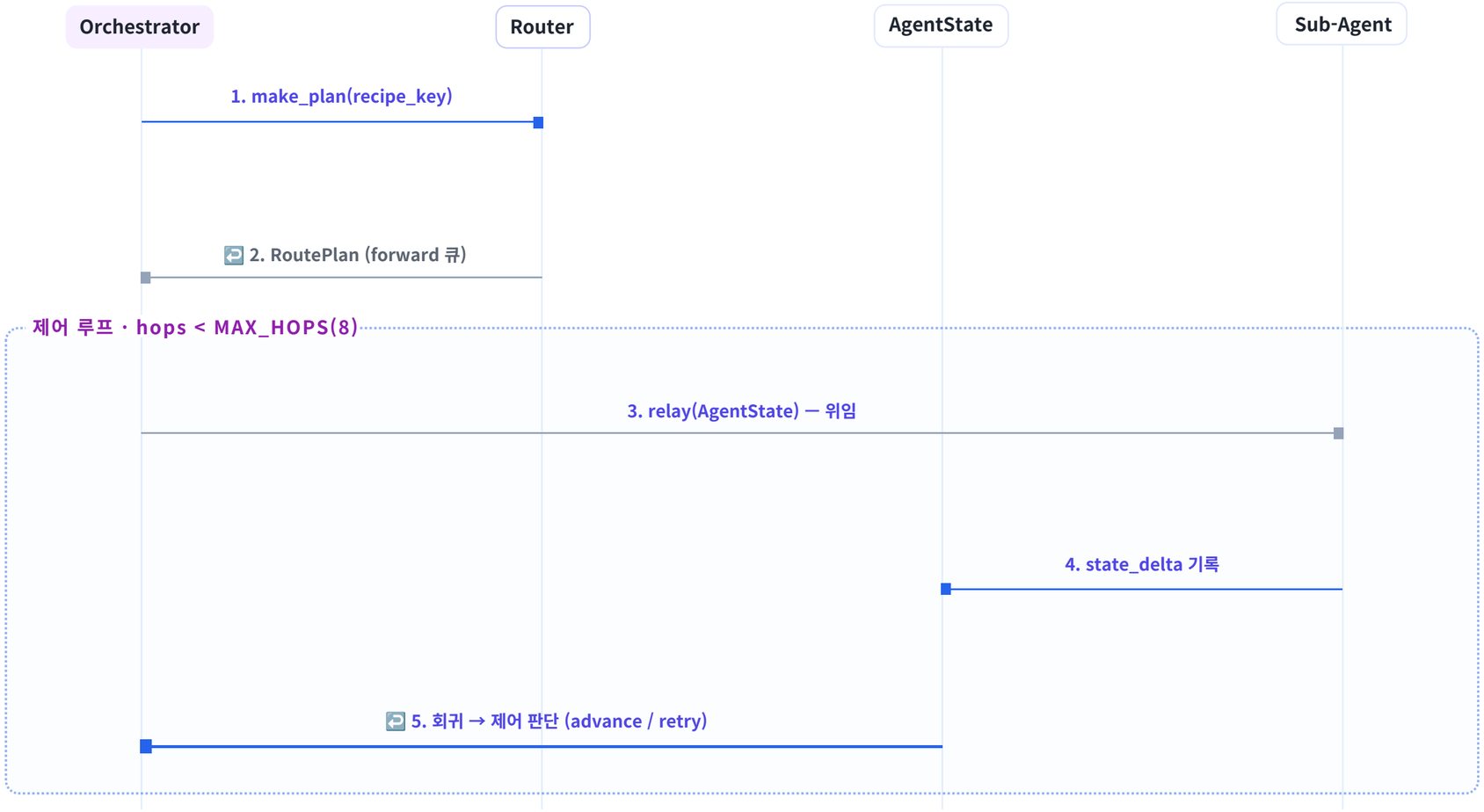
selected_skill_id 를 AgentState에 write
→ 다음 hop이 read

무한루프 3중 방어

- MAX_HOPS 8
- RETRIES 1
- 보정 1회

SUPERVISOR CONTROL LOOP

state 매개 핸드오프 · 결정형 제어 판단



💡 **Supervisor 패턴** - Sub-Agent는 AgentState에 결과만 쓰고, '다음에 무엇을 할지'는 Orchestrator가 단독으로 결정한다. 성공 → 다음 스텝, 연결 실패 → 재시도 ×1, 자체 오류 → Composer 보정 ×1

Workflow Composer — 생성 시퀀스

고정 DAG로 노드를 조합 · 검증 · 확정하는 내부 흐름

전처리 게이트

위험 · 권한 차단

compress 턴 압축 →
security 패턴·권한 pre-screen

노드 후보 강화

검색 + GraphRAG

의미검색 + 구조노드 + **CAN_FOLLOW** 확장
+ 개인 패턴 RAG

품질 루프 · HITL

validate

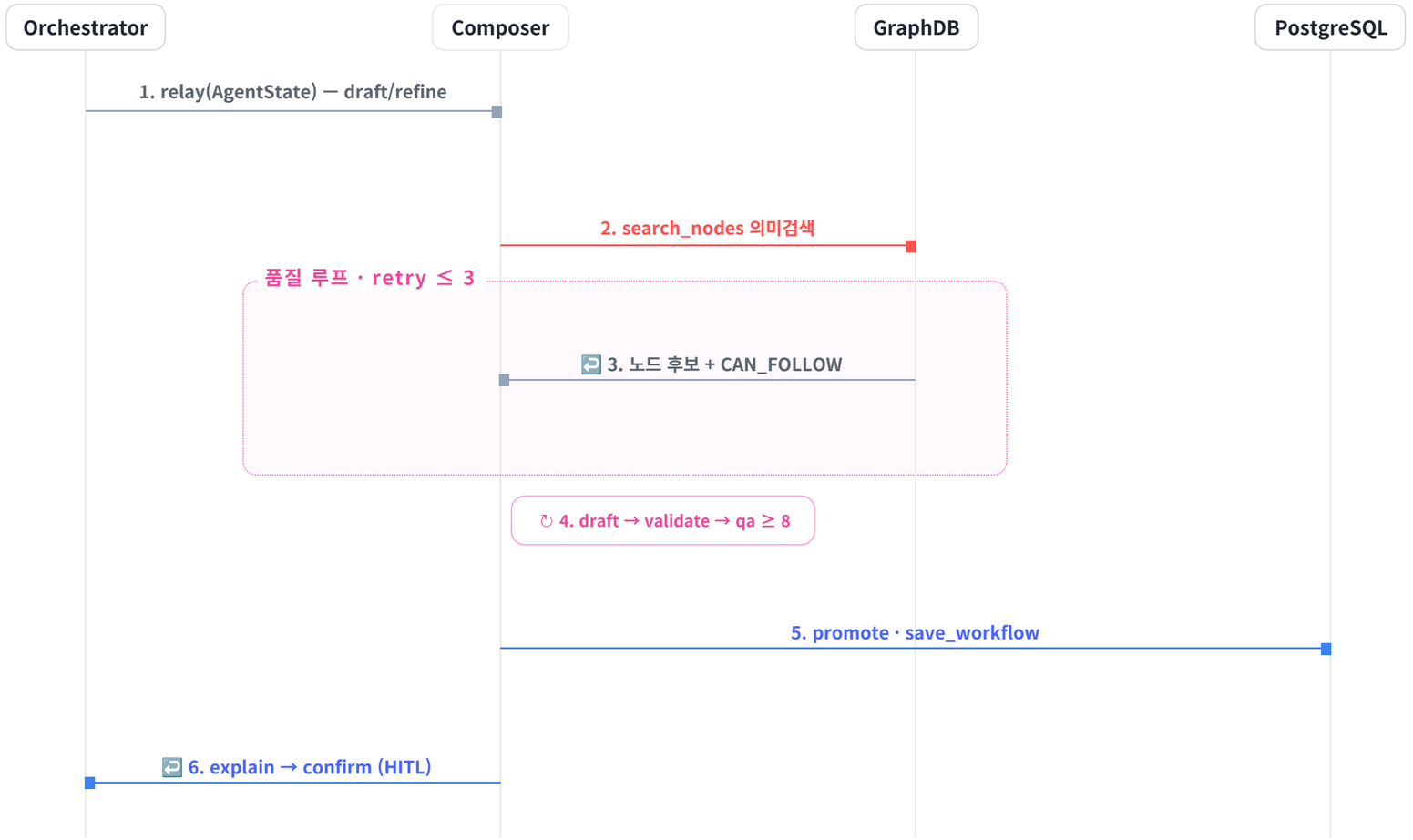
qa ≥ 8

retry ≤ 3

confirm HITL

COMPOSER FIXED DAG SEQUENCE

검색 → 초안 → 검증 → 확정 저장



💡 **Composer** 검색 → 초안 → 검증 → QA를 거쳐 score ≥ 8을 통과해야 확정하고, 확정 워크플로우만 PostgreSQL에 저장한다.

Skills Builder — SOP 추출 시퀀스

SOP 문서를 wizard 3단계로 personal 스킬 DRAFT로 만든다

2단계 분리

메타 → 상세 분할 추출

JSON 잘림(ctx 8192 초과)을 분리로 해소

결정적 스켈레톤

코드가 BINDS 결정

SOP→스켈레톤 조립,
COMPOSER.md·정밀 매핑 ADR-0028

이중 저장

- GCS · SKILL.md
- PostgreSQL DRAFT
- GCS · COMPOSER.md

SOP WIZARD SEQUENCE

추출 → 검토 → 확정 · HITL 2지점

💡 **Skills Builder** 추출 → 검토 → 확정의 HITL wizard로, 사용자가 편집한 결과만 DRAFT 스킬로 저장한다. 게시 시 NodeDefinition으로 승격된다.

Q&A

감사합니다.