KIRO CLI WORKSHOP · OUROBOROS

루프가 Kiro를 만나면

AWS Kiro와 함께하는 Ouroboros · 철학, 통합, 그리고 한 바퀴

이재규 (JQ Lee)

ZEP Tech Lead · Ouroboros maintainer

github.com/Q00/ouroboros

SPEAKER · 이재규 (JQ LEE)

루프를 만들고, 무대에서도 같은 루프를 이야기합니다

ZEP에서 Tech Lead로 일하며, Ouroboros라는 오픈소스 에이전트 하네스를 메인테인합니다.

- **ZEP Tech Lead**. 프로덕트 엔지니어링과 에이전트 하네스를 함께 맡습니다.
- **Ouroboros maintainer**. github.com/Q00/ouroboros · 인터뷰부터 진화까지 잇는 실행 계약.
- **KT · 삼성 외부 강연**. 서울과기대 Agentic AI 외래교수.

목차

→ 커뮤니티 오픈소스 · MIT · AWS가 공식 지원·보증하는 서비스는 아닙니다.

01 · THE PHILOSOPHY

루프는 반복이 아니라, 정제입니다

자기 꼬리를 삼키는 뱀, 우로보로스.
모든 실행은 평가를 남기고, 모든 평가는 다음 세대의 입력이 됩니다.

WHY OUROBOROS EXISTS

AI 코딩은 출력이 아니라 입력에서 실패합니다

모호한 한 줄 요구를 그대로 실행하면, 모델이 좋아져도 결과는 복권입니다. 그래서 Ouroboros는 실행 전에 소크라테스식 인터뷰로 모호함을 깎습니다.

모호도가 0.2 이하로 떨어져야 요구는 Seed, 불변의 명세가 되고, 그때부터 실행이 허락됩니다.

README · "interview, crystallize, execute, evaluate, evolve"

THE AGENT OS CONTRACT

어떤LLM이 실행해도, 계약은 하나

OS가 시스템콜을 추상화하듯, Ouroboros는 에이전트의 모든 행동을 하나의 실행 계약으로 추상화합니다.

SEED

명세가 곧 계약

인터뷰로 굳힌 목표와 인수 기준. 실행 중엔 아무도 못 바꿉니다.

LEDGER

모든 행동이 장부에

이벤트 소싱. 모든 실행은 기록되고, 재생되고, 감사됩니다.

RUNTIME

실행자는 갈아 끼운다

Claude·Codex·Gemini·Kiro 등 13개 런타임, 같은 워크플로.

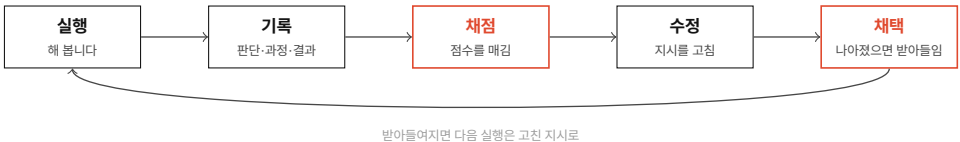
→ 그래서 Kiro는"지원 목록의 한 줄"이 아니라, 이 계약을 실행하는 동등한 커널 백엔드입니다.

README · "a local-first runtime layer that turns non-deterministic agent work into a replayable, observable, policy-bound execution contract"

LOOP ENGINEERING · THE PHILOSOPHY

루프 한 바퀴마다,
조금씩 정제됩니다

자기 꼬리를 삼키는 뱀 우로보로스는 연금술사들에게 갱신의 상징이었습니다. 진정으로 끝나는 것은 없고, 모든 것이 다음 시작의 양분이 됩니다.



— 모든 실행은 평가를 남기고, 모든 평가는 다음 세대의 Seed를 키웁니다. 루프가 도는 이유는 깎기 위해서입니다.
— 약한 모델도 이 루프를 몇 바퀴 돌면 괜찮아집니다. 비싼 모델을 먼저 부르는 대신, 지시를 스스로 다듬게 만듭니다.

FRUGALITY · HOW WE ACTUALLY BUDGET

토큰을 아끼는 게 아니라,
성과당 비용을 잽니다

RpC = pass(0/1) ÷ 쓴 토큰 × 1000. 토큰만 줄이면 성과가 0으로 몰립니다. 그래서 같은 성과를 얼마나 아껴 냈는지를 봅니다.

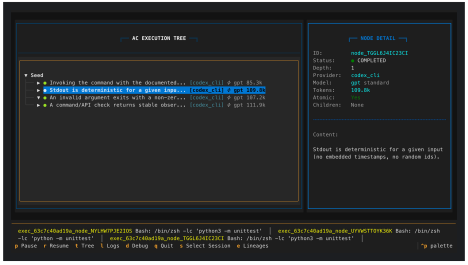
— **승격은 한 단계씩만**. 작고 쉬운 작업은 저렴한 모델로 먼저 보내고, 실패해야만 한 단계 위로 올립니다.
— **근거를 잃으면 즉시 FAIL**. 싼 모델로 내렸다가 근거 없는 답이 나오면 통과가 아닙니다.
— **기록이 라우터가 됩니다**. 작업마다 남는 모델·비용·통과 이력이 다음 실행의 라우팅 근거가 됩니다. Kiro도 예외 없이 같은 장부에 남습니다.

→ "Token maxing is dead. Say hello to frugality." 그 원칙은 런타임이 바뀌어도 그대로입니다.

RECORDS · OUROBOROS DASHBOARD

AC마다 남는 모델·토큰 기록

AC(acceptance criteria) 하나가 끝날 때마다, 어떤provider·모델로 몇 토큰을 써서 통과했는지 바로 기록됩니다.



— 오른쪽 NODE DETAIL의 Provider·Model·Tokens가 RpC(성과당 비용) 계산의 재료입니다.

— 같은 유형의 AC가 또 나오면, 이 기록을 근거로 저렴한 모델부터 시도합니다.

→ Kiro도 예외 없이 같은 장부 위에서 돕니다.

FRUGALITY · RECORDS BECOME A ROUTER

기록이 쌓이면,
다음 모델 선택의 근거가 됩니다

어떤 작업을 어떤 모델로 풀었는지 남긴 기록이, 다음 실행에서 모델을 고르는 근거가 됩니다.

— 제일 비싼 모델을 기본값으로 켜 두지 않습니다. 비싼 모델은 실패한 다음에만 부릅니다.

— 판단 근거는 감이 아니라 이력입니다. 같은 유형의 작업을 작은 모델이 통과시킨 기록이 있으면, 다음에도 거기서 시작합니다.

→ 쉽게 시작하고, 실패할 때만 한 단계 올립니다.

02 · THE INTEGRATION

Kiro는 어떻게 들어가 있나

추측이 아니라 실제로 확인합니다.
이슈 번호, 되는 것과 안 되는 것.

GITHUB.COM/Q00/OUROBOROS · VERIFIED ISSUES

Kiro 지원은 커뮤니티가 가져왔습니다

#98 · 2026-03-08

OctopusGarden

루프를 주고받기 시작한 첫 영감

#303 · 2026-04-02

Kiro 어댑터 착륙

커뮤니티 기여자의PR로 LLM·러너 어댑터가 들어옴

#606 · 2026-05-04

런타임 백엔드 승격

한 번의 반려를 거쳐, 하루 만에 v0.34.0으로 릴리스

#1916/#1917 · 2026-08-07

원이 닫히다

run → eval → evolve를 한 번의 ooo run으로 잇는 RFC

→ 메인테이너가 넣은 게 아니라, Kiro를 쓰던 사용자가 어댑터를 써서 보냈습니다. 그게 이 프로젝트가 커지는 방식입니다.

THE ADAPTER · WHAT IT ACTUALLY DOES

어댑터는 껍데기가 아니라 번역 계층입니다

Ouroboros가 하는 말을 Kiro가 알아듣는 말로 옮깁니다. 모델 이름, 허용 범위, 화면에 찍히는 로그까지 전부 다시 씁니다.

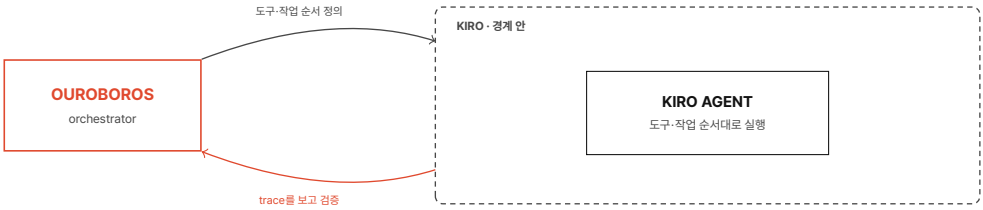
— **모델 이름을 맞춥니다**. Ouroboros가 부르는 이름과 Kiro가 알아듣는 이름이 달라서, 어댑터가 중간에서 표기를 맞춰 줍니다.
— **허용 범위를 옮깁니다**. "무엇까지 허용할지"를 Kiro가 이해하는 표현으로 바꿔서 넘깁니다.
— **화면을 정리합니다**. Kiro가 찍는 원시 로그에는 진행 표시나 색상 코드가 섞여 있는데, 어댑터가 읽을 수 있는 텍스트만 남깁니다.

→ 번역이 안 되는 부분은 숨기지 않고, 안 되는 채로 그대로 남깁니다.

CAPABILITY · WHAT GOES IN, WHAT STAYS OUTSIDE

Kiro 안에 넣는 것과, 바깥에서 지키는 것

Kiro 경계 안에는 도구와 작업 순서만 넘겨줍니다. 통과 여부는 항상 경계 바깥, Ouroboros가 정합니다.



— **안으로 들어가는 것**. Ouroboros가 정의한 도구 목록과 작업 순서를 Kiro 에이전트에게 넘겨, 그 안에서 실행하게 합니다.
— **바깥에 남는 것**. 어떤 판단을 했고 어떤 도구를 불렀는지 남긴trace는 경계 밖으로 나와, Ouroboros가 인수 기준과 대조합니다.
— **안에서 스스로 보고한 결과를 그대로 믿지 않습니다**. 통과 여부는 Kiro 안이 아니라, 바깥에서trace를 본Ouroboros가 정합니다.

→ 경계 밖에서 확인할 수 없는 일은, 할 수 있다고 선언하지 않습니다.

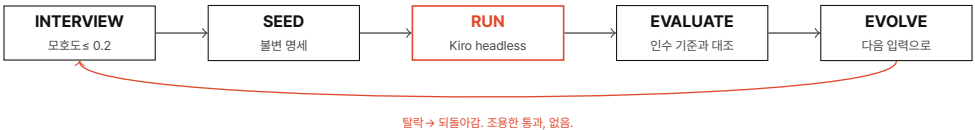
03 · THE MEETING

둘이 만나면,
한 바퀴가 이렇게 돕니다

인터뷰부터 진화까지.
Kiro 세션 안에서, 그리고 터미널에서.

ONE REVOLUTION · INSIDE A KIRO SESSION

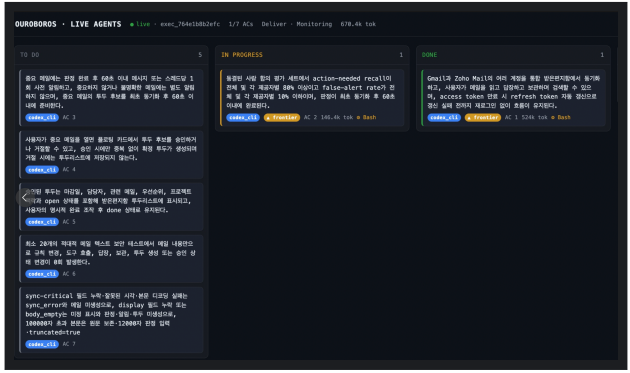
한 줄 요구가, 검증된 코드가 되는 한 바퀴



- Kiro 세션 안: `ooo interview` → `ooo seed` → `ooo run`. Claude Code·Codex에서 쓰던 명령 그대로입니다.
- Kiro는 **채점 답안지를 받지 않습니다.** 검증 명령은 의도적으로 숨겨지고, 채점은 워커가 아니라 orchestrator가 합니다. 실패를 스스로 보고해서 지울 수 없습니다.

LIVE · KIRO SESSIONS ON THE BOARD

도는 동안, 화면은 이렇게 보입니다



실제 실행 화면 · TO DO · IN PROGRESS · DONE

KIRO × OUROBOROS · WHAT COMES OUT

한 바퀴가 끝나면,
손에 남는 건 네 가지입니다

— **인수 기준을 통과한 코드**. Kiro가 구현하고, orchestrator가 채점을 마친 것만 통과합니다.
— **seed_<id>.yaml**. 불변 명세. 같은 시드를 Claude·Codex에서 다시 돌려도 같은 계약으로 실행됩니다. Kiro에 잠기지 않습니다.
— **이벤트 장부**. Kiro의 모든 행동이 `~/.ouroboros/ouroboros.db`에 남아 재생·감사가 됩니다.
— **AC 트리와 칸반**. 어느 기준이 통과했고 어디서 탈락했는지, 도는 동안 실시간으로 보입니다.

→ 이 관측 화면들은 Kiro를 모릅니다. 장부만 읽습니다. 그래서 런타임을 바꿔도 화면은 그대로입니다.

SEED · THE IMMUTABLE SPEC

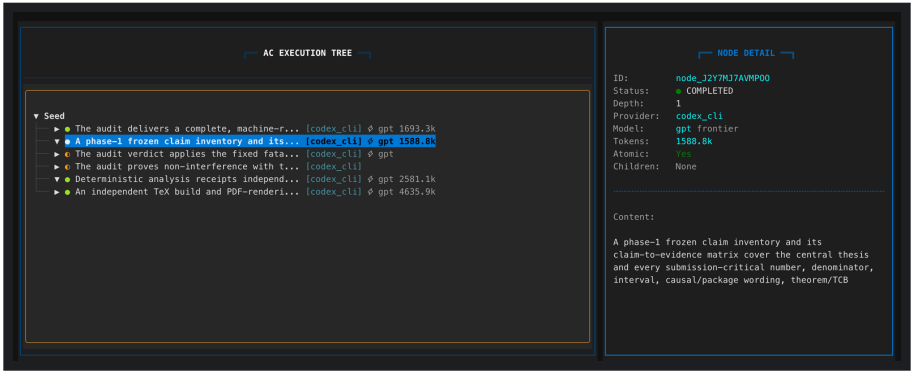
시드 하나가, 이렇게 고정됩니다

```
goal: Create hello_auto.py with hello_auto() returning exactly 'hello from ooo auto'.
  Create tests/test_hello_auto.py importing hello_auto and asserting that exact value.
  Verification command is uv run pytest tests/test_hello_auto.py.
task_type: code
brownfield_context:
  project_type: greenfield
  context_references: []
  existing_patterns: []
  existing_dependencies: []
constraints:
- Keep scope to a reversible local MVP, preserve existing project patterns, and avoid
  new dependencies unless explicit acceptance criteria require them
- 'Non-goal: Do not perform credential handling, billing, production deployment, legal
  or medical judgment, security-sensitive authority choices, or ambiguous external
  side effects'
acceptance_criteria:
- Create `hello_auto.py` and `tests/test_hello_auto.py` so `hello_auto() -> str` returns
  exactly `hello from ooo auto`, the test imports `hello_auto` and asserts that exact
  value, and the exact command `uv run pytest tests/test_hello_auto.py` passes.
- A command/API check returns stable observable output or artifacts proving the original
  requirement for All public API symbols importable from the documented module path.
- A command/API check returns stable observable output or artifacts proving the original
  requirement for Unit tests cover every public function/method's primary success
  path.
- A command/API check returns stable observable output or artifacts proving the original
  requirement for `ruff check` and the project's type-check command exit 0.
- A command/API check returns stable observable output or artifacts proving the original
  requirement for Completion requires an observable check that demonstrates the requested
  behavior and a negative or edge-path check where the implementation surface supports
  one.
```

실제 seed_<id>.yaml · goal · constraints · acceptance_criteria

AC TREE · NODE-LEVEL DETAIL

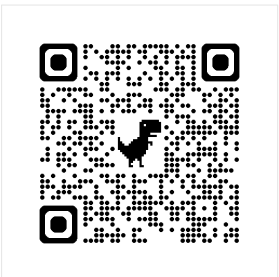
기준 하나마다, 통과 여부가 남습니다



실제 실행 화면 · AC EXECUTION TREE · NODE DETAIL

OUROBOROS · TRY IT YOURSELF

지금, 여기서 시작할 수 있습니다



ouroboros.page

github.com/Q00/ouroboros · MIT LICENSE

생태계 활성화를 위해 스타를 눌러주세요

TAKEAWAY

루프는 끝나지 않습니다.
오늘, 무대 하나가
더 늘었을 뿐입니다

```
$ pipx install 'ouroboros-ai[mcp]'
$ ouroboros setup --runtime kiro
```

OUROBOROS

github.com/Q00/ouroboros

MIT LICENSE