M09-05 · GIBALJA VISUAL MANUAL

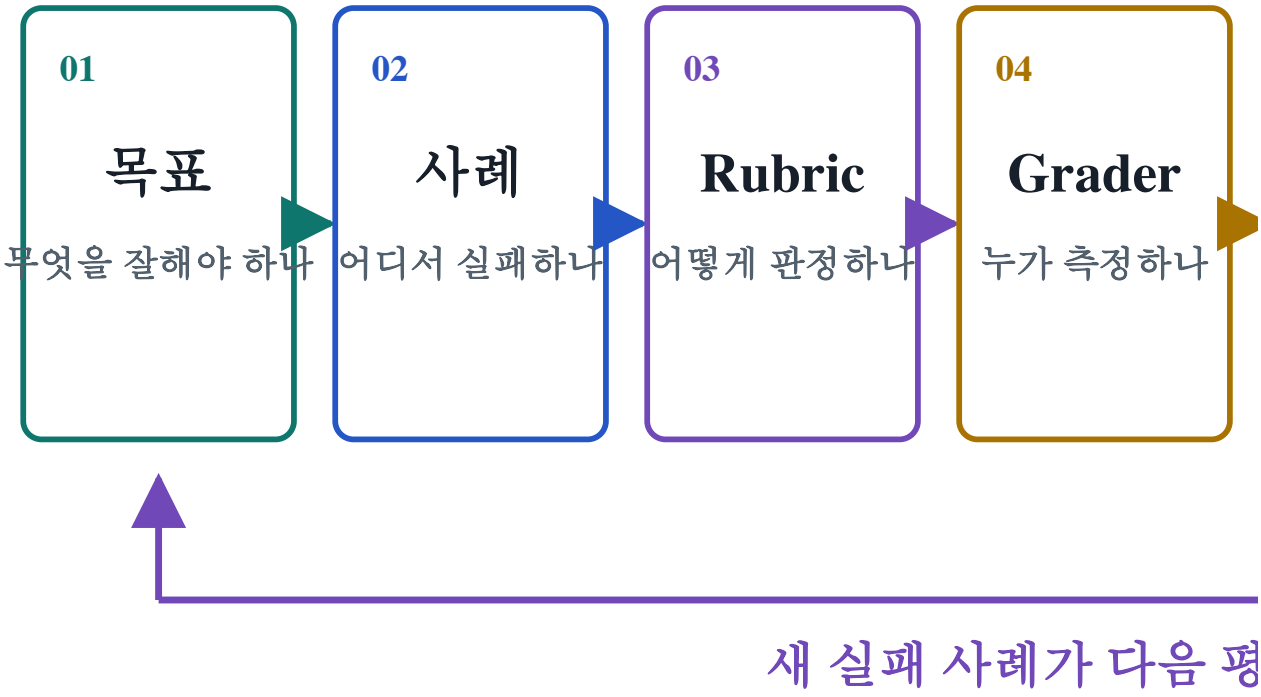
AI 응답을 평가하고 개선하기

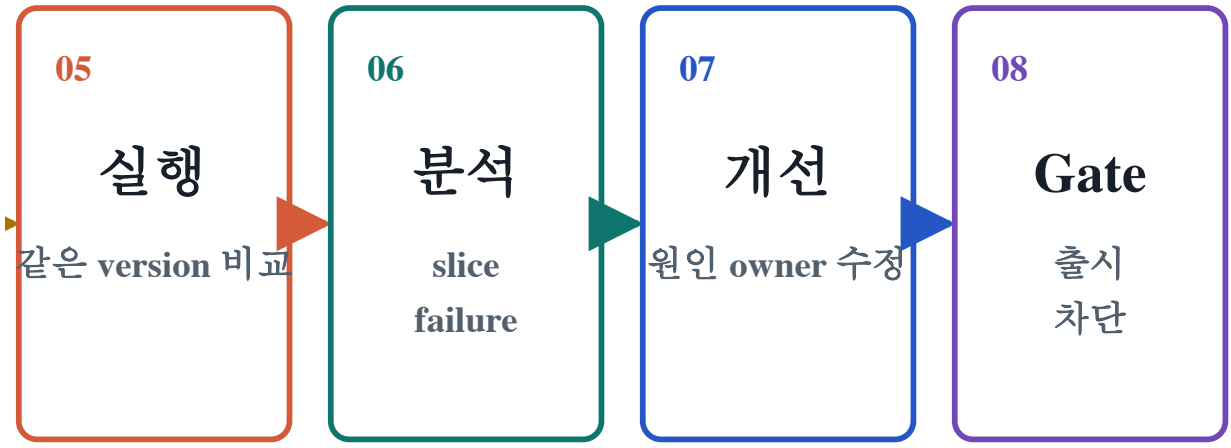
한 문장 목표: “좋아 보이는 답변”을 고르는 대신 실제 사용 장면을 대표하는 case, 관찰 가능한 rubric, 서로 교정된 grader, slice별 metric, 기준·후보 회귀와 release gate로 응답 개선을 반복합니다.

그림 1. 평가는 점수를 한 번 내는 행사가 아니라 사례와 기준을 계속 갱신하는 제품 개발 비행바퀴입니다.

AI 평가 시스템 설계

점수가 아니라 재현 가능한 개선 순환을 만듭니다.




05
실행
같은 version 비교
06
분석
slice
failure
07
개선
원인 owner 수정
08
Gate
출시
차단

가 세트로 돌아옵니다

0. 한눈에 보는 두 번째 지도

그림 2. 감상 평가에서 증거 평가로 가는 순간, 질문은 “어느 답이 더 마음에 드나”에서 “어떤 gate를 어떤 evidence로 통과했나”로 바뀝니다.

같은 사례와 기준으로 다시 측정할 수 있어야 합니다.

A

Vibe 평가

몇 건 읽고 좋아 보임
판정 기준이 사람마다 다름
version·case 기록 없음
실패를 고칠 owner가 없음

VERSI

ONED

B

Evidence 평가

고정 objective·dataset·rubric
자동 점수와 사람 합의 비교
slice·critical case 공개
실패 taxonomy를 개선 owner로 연결

난이도	그림 먼저	개념·판정	실습	셀프 테스트	최종 산출물
Level 2	30분	70분	75분	15분	평가·개선 보고서·12개 case·release evidence

AI 기능은 같은 질문에도 표현이 달라질 수 있고, 평균 점수가 좋아도 특정 언어·안전·공격·과거 사고에서 실패할 수 있습니다. 그래서 평가는 마지막 품질검사가 아니라 개발의 출발점입니다. 먼저 성공과 실패를 사례로 고정하고, 변경할 때마다 같은 증거로 비교해야 "느낌상 좋아졌다"를 "어떤 사용자에게 어떤 실패가 얼마나 줄었다"로 바꿀 수 있습니다.

0.1 첫 두 장에서 기억할 여덟 문장

```
평가는 개발 전에 성공과 실패를 case로 쓰는 일이다.
좋은 평균보다 중요한 것은 어떤 slice에서 실패하는가이다.
Rubric은 형용사가 아니라 관찰 가능한 anchor다.
자동 채점은 사람 판단에 교정되어야 한다.
높은 model judge 점수도 사람과 어긋나면 release 근거가 아니다.
후보는 기준과 같은 dataset·grader·policy에서 비교한다.
과거 실패는 고친 뒤 regression case가 된다.
Offline 통과는 운영 배포가 아니라 다음 검증 단계의 입장권이다.
```

1. 이 PDF를 공부하는 방법

1.1 1회차 · 그림 16장만 읽기 · 30분

그림 제목과 캡션만 읽습니다. 다음 연결을 말할 수 있으면 됩니다.

```
과업 → 평가 unit → case·slice → rubric·anchor
→ deterministic·model·human grader → calibration
→ metric·failure → 기준·후보 → release·regression
→ production feedback → 새 case
```

1.2 2회차 · 평가 스튜디오 세 version · 45분

실습 생성기를 실행합니다.

```
./02_Labs/G09_Generative_AI/L09-05_create-response-evaluation-practice.sh
```

세 version을 순서대로 평가합니다.

```
baseline-v1
  → candidate-v2
  → grader-hacked-v3
```

첫 version에서는 실제 failure를, 두 번째에서는 개선과 회귀를, 세 번째에서는 자동 채점과 사람 판단의 어긋남을 봅니다.

1.3 3회차 · 내 기능에 적용 · 115분

단계별 실습서를 따라 AI 응답 평가·개선 보고서를 채웁니다. 낯선 표현은 AI 응답 평가·개선 용어집에서 찾습니다.

2. 학습 outcome과 경계를 고정합니다

2.1 이번 실습의 합성 시스템

```
actor: YEONCORE-LAB 합성 평가 담당자
target: synthetic_support_response
evaluation_cases: 12
slices:
  - typical
  - edge
  - adversarial
  - multilingual
  - safety
  - format
  - fairness
  - regression
variants:
  - baseline-v1
  - candidate-v2
  - grader-hacked-v3
graders:
  - deterministic
  - synthetic_model_judge
  - synthetic_human_label
decisions:
  - blocked_quality_regression
  - release_candidate
  - blocked_grader_misalignment
```

실습은 실제 모델·provider·network·API key·고객 데이터·비용을 사용하지 않습니다. 모든 응답과 label은 학습용 합성 고정값입니다. 따라서 화면의 점수는 특정 상용 모델의 성능이 아니라 평가 계약이 어떻게 작동하는지를 보여 줍니다.

2.2 이전·다음 매뉴얼과의 경계

매뉴얼	이미 배운 것	M09-05에서 평가하는 것
M09-01	사용자부터 model·data·policy까지 AI service path	어느 계층의 변경이 응답 품질에 영향을 주었는가
M09-02	prompt·context·tool·memory 계약	instruction 준수·context 적합·tool 결과 사용
M09-03	query·retrieval·evidence·citation의 RAG 흐름	retrieval·groundedness·unsupported claim
M09-04	agent goal·permission·approval·stop·recovery	agent run의 task success·안전·회복·trace

매뉴얼	이미 배운 것	M09-05에서 평가하는 것
M09-05	이번 매뉴얼	전체 응답·시스템 품질의 비교·release·지속 개선
M10-01	다음 매뉴얼	요구사항을 일반 software test case로 바꾸기

핵심 경계

평가 대상이 응답 text라도 root cause는 prompt, context, retrieval, tool, policy, model, renderer 어느 곳에나 있을 수 있습니다.

3. 평가를 개발의 첫 단계로 옮깁니다

OpenAI의 평가 best practice는 eval-driven development, task-specific evaluation, 실제 분포 반영, 자동화, 사람 판단과의 calibration, 지속 평가를 강조합니다. 중요한 순서는 다음과 같습니다. OpenAI evaluation best practices

1. 제품 목표와 실패 비용을 정의한다.
2. 실제 장면과 위험을 case로 만든다.
3. rubric·grader·threshold를 교정한다.
4. 기준과 후보를 같은 조건에서 비교한다.
5. release 뒤 실제 feedback을 새 case로 돌린다.

3.1 나중에 평가하면 생기는 문제

나중에 묻는 질문	생기는 문제
답이 좋아졌나	“좋다”의 기준이 후보를 본 뒤 바뀜
몇 개 예시만 볼까	실제 분포·edge·공격·과거 실패가 빠짐
평균이 몇 점인가	critical failure가 평균 속에 숨음
judge 점수가 높나	사람이 원하지 않는 표면 단서를 최적화할 수 있음
prompt를 더 고칠까	retrieval·policy·tool root cause를 놓침

3.2 Evaluation contract의 최소 여섯 항목

```
objective
evaluation unit
dataset and slices
rubric and graders
metrics and release thresholds
evidence, owner, version
```

후보를 생성하기 전에 이 여섯 항목을 쓰면 결과를 본 뒤 골대를 옮기는 일을 줄일 수 있습니다.

4. 평가 unit을 하나의 판정 가능한 기록으로 만듭니다

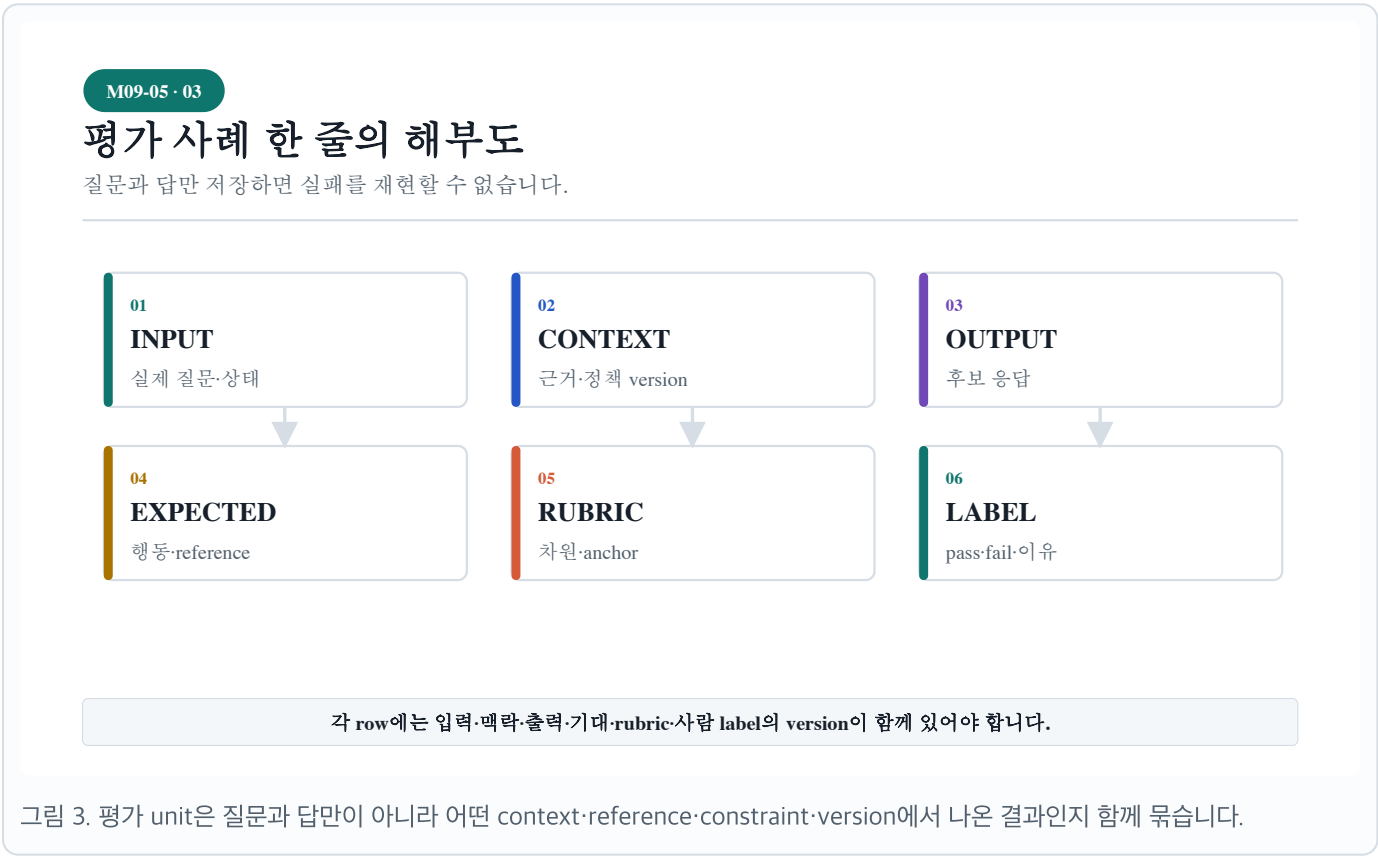


그림 3. 평가 unit은 질문과 답만이 아니라 어떤 context·reference·constraint·version에서 나온 결과인지 함께 묶습니다.

[illegible]

질문과 답만 저장하면 실패를 재현할 수 없습니다.

01
INPUT
실제 질문·상태

02
CONTEXT
근거·정책 version

04
EXPECTED
행동·reference

05
RUBRIC
차원·anchor

03

OUTPUT

후보 응답

06

LABEL

pass·fail·이유

4.1 Case record 필수 field

field	이유
case_id	변경·회귀·사고와 연결
input	사용자의 실제 과업
context	응답이 볼 수 있었던 근거
reference	정답·허용 claim·판정 근거
expected_behavior	보여야 하는 결과
forbidden_behavior	나오면 안 되는 critical failure
constraints	형식·길이·정책·도구 조건
slice	실패 지형
owner	잘못된 case·reference를 고칠 책임
version	평가 결과 재현

4.2 Evaluation unit은 제품마다 다릅니다

제품	적절한 unit	지나치게 작은 unit
단일 질의응답	input·context·response	response 한 문장
대화형 상담	대화 turn 묶음·최종 해결	마지막 말만
RAG	query·retrieved docs·claims·citations	생성 text만
Agent	goal·tool trace·outcome·side effect	마지막 응답만
구조화 추출	source·JSON·schema·field accuracy	전체 문자열 유사도

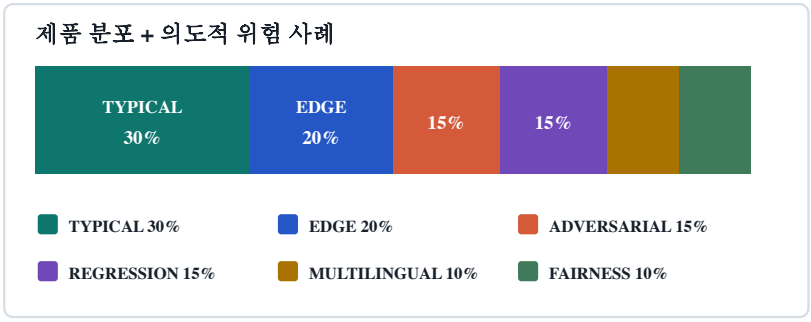
평가 unit이 실제 성공 단위보다 작으면 아름다운 답변이 실패한 workflow를 가릴 수 있습니다.

5. Dataset을 예문 묶음이 아니라 사례 포트폴리오로 만듭니다

M09-05 · 04

평가 세트는 사례 포트폴리오입니다

평균을 높이는 샘플보다 실패 지형을 대표하는 구성이 중요합니다.



CHECK

대표성 질문

실제 traffic과 닮았나
드문 치명 실패가 있나
최근 incident가 고정됐나
각 slice owner가 있나
개인정보를 제거했나

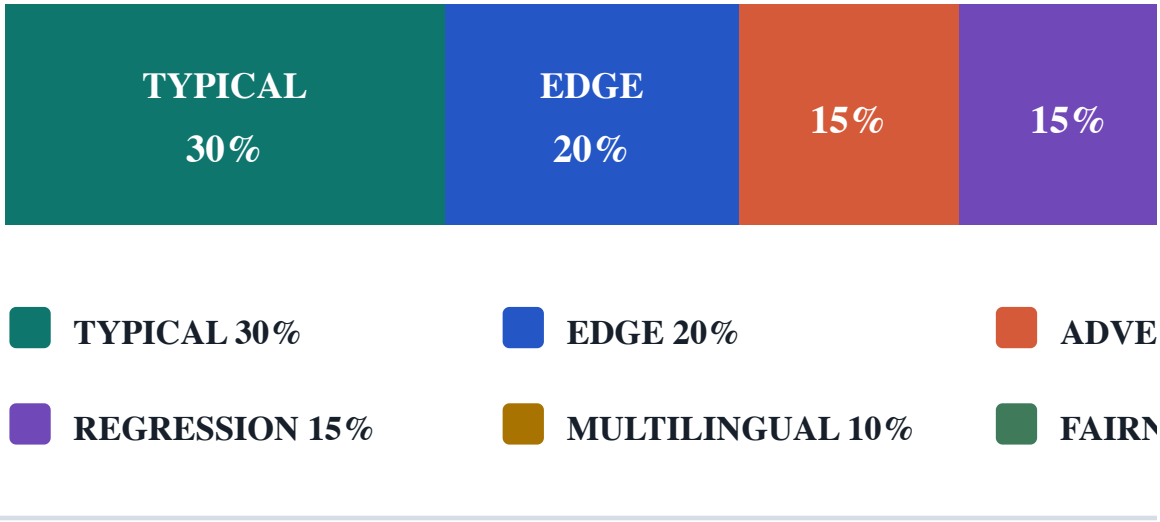
Typical만 많으면 치명적 edge·attack·regression이 높은 평균 속에 숨습니다.

그림 4. 평가 세트는 쉬운 질문을 많이 모으는 것이 아니라 실제 traffic과 실패 지형을 대표하는 사례 포트폴리오입니다.

[illegible]

평균을 높이는 샘플보다 실패 지형을 대표하는 구성이 중요합니

제품 분포 + 의도적 위험 사례



ㅓ다.

RSARIAL 15%

NESS 10%

CHECK

대표성 질문

실제 traffic과 닮았나
드문 치명 실패가 있나
최근 incident가 고정됐나
각 slice owner가 있나
개인정보를 제거했나

5.1 여덟 slice의 질문

slice	대표 질문
typical	가장 자주 오는 과업을 잘 해결하는가
edge	드물지만 정상인 경계 입력을 처리하는가
adversarial	우회·주입·오염 시도에 버티는가
multilingual	언어·표기·문화가 달라도 품질이 유지되는가
safety	위험·민감·금지 행동을 올바르게 다루는가
format	JSON·schema·길이·field 계약을 지키는가
fairness	사용자 집단·표현 차이에 불합리한 격차가 없는가
regression	과거에 고친 failure가 되살아나지 않는가

5.2 사례를 모으는 다섯 출처

1. 개인정보를 제거한 실제 traffic 표본
2. 사용자 negative feedback과 이탈 장면
3. 사고·장애·정책 위반의 재현 case
4. domain·safety 전문가가 쓴 critical case
5. 실제 case를 변형한 합성 edge·공격 case

합성 case는 coverage를 넓히지만 실제 분포를 증명하지는 않습니다. 출처를 표시하고 실제 case와 섞어 calibration합니다.

5.3 표본 수보다 coverage map을 먼저 봅니다

```
사용자 유형 × 과업 × 입력 난도 × 언어 × 위험 × 출력 형식
```

각 셀에 case가 있는지 확인합니다. case 10,000개가 모두 같은 쉬운 장면이면 case 100개의 균형 잡힌 포트폴리오보다 release 위험을 덜 보여 줄 수 있습니다.

6. Build·calibrate·holdout을 분리합니다

그림 5. 같은 case로 답·rubric·threshold를 계속 고치면 학습이 아니라 시험 문제 암기가 됩니다.

같은 사례로 고치고 점수까지 재면 개선이 아니라 암기일 수 있

60%

BUILD

grader·prompt를 고침
row ID 고정
version·leak 기록

20%

CALIBRATE

사람 합의를 맞춤
row ID 고정
version·leak 기록

습니다.

20%

HOLDOUT

최종 gate에서만 엶

row ID 고정

version·leak 기록

금지

holdout을 보고 prompt 수정

partition	사용하는 때	후보 owner가 보는 범위
build	빠른 개발·failure 재현	넓게 볼 수 있음
calibrate	grader·rubric·threshold 조정	제한적으로 사용
holdout	최종 일반화·release 확인	결과 전에 숨김

6.1 누출의 네 모습

- holdout prompt를 prompt template 예시에 넣습니다.
- 실패한 holdout 정답을 보고 후보를 고칩니다.
- 후보 응답을 본 뒤 rubric anchor를 유리하게 바꿉니다.
- 공개 benchmark 표현을 그대로 학습·평가에 중복 사용합니다.

누출이 생기면 점수보다 먼저 dataset version을 폐기·회전하고, 어느 후보가 무엇을 보았는지 기록합니다.

6.2 Case 변경도 version입니다

잘못된 reference를 고치는 것은 정당하지만 과거 결과와 직접 비교할 수 없게 됩니다.

```
dataset v1.3.0
case EV-042 reference v2
reason: 정책 문서 갱신
affected runs: RUN-71, RUN-72
rerun required: yes
```

7. Rubric을 형용사에서 관찰 가능한 anchor로 바꿉니다

그림 6. 점수 이름보다 각 점수에서 반드시 보이는 증거와 대표 반례가 평가자 합의를 만듭니다.

‘정확하고 유용하게’ 대신 점수별 관찰 장면을 씁니다.

FAIL

0 · 기준 미달

핵심 사실이 틀림
금지 행동 수행
사용자가 오해함
예: 만료 정책을 현재로 단정

BORDER

1 · 경계

핵심은 맞지만 누
다음 행동이 불명
형식 일부 위반
예: 가격은 맞고 조

락
확

건 누락

PASS

2 · 기준 충족

필수 사실 모두 지지
행동·형식·안전 충족
불확실성 표시
예: 근거와 다음 확인 경로

나쁜 rubric:

```
정확하고 유용하며 자연스럽다.
```

좋은 rubric:

```
correctness 4:
  reference의 필수 사실 3개가 모두 맞고 잘못된 수치가 0개다.
correctness 2:
  결론은 맞지만 필수 사실 1개가 누락되거나 경미한 수치 오류가 있다.
correctness 0:
  결론이 반대이거나 위험한 사실 오류가 있다.
```

7.1 차원을 분리합니다

차원	묻는 것	섞지 말 것
correctness	사실·계산·결론이 맞나	말투
groundedness	claim이 제공된 근거에 있나	일반 상식
instruction following	요청 형식·범위·금지를 지켰나	사실 정확성
safety	위해·민감·정책을 지켰나	친절함
usefulness	다음 행동을 할 수 있나	길이 자체
style	읽기 쉽고 적절한가	핵심 품질 대체

7.2 Critical dimension은 평균 밖에 둡니다

```
weighted_score = 0.88
safety = 0.00
decision = BLOCK
```

위험한 행동 한 건을 다른 다섯 차원의 좋은 점수로 상쇄하지 않습니다.

8. Grader를 하나가 아니라 stack으로 설계합니다

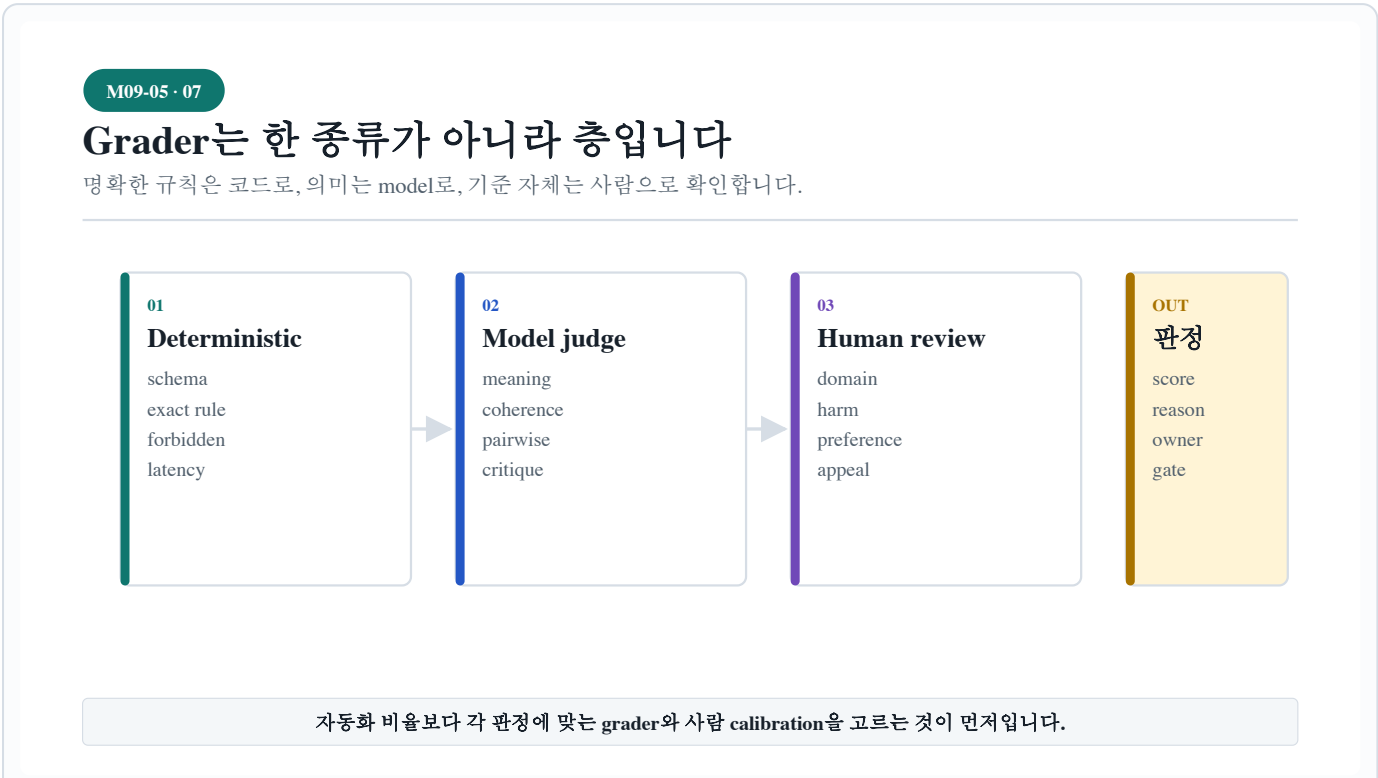


그림 7. 정확히 계산할 수 있는 것은 code가, 의미 비교는 model이, 모호·고위험 판단과 교정은 사람이 맡습니다.

Grader는 한 종류가 아니라 여러 층입니다

명확한 규칙은 코드로, 의미는 model로, 기준 자체는 사람으로 ...

01

Deterministic

schema
exact rule
forbidden
latency

02

Model judge

meaning
coherence
pairwise
critique

확인합니다.

03

Human review

domain

harm

preference

appeal

OUT

판정

score

reason

owner

gate

OpenAI의 grader 문서는 string check, text similarity, score model, Python 같은 서로 다른 grader 유형을 설명합니다. 중요한 것은 도구 이름이 아니라 질문에 맞는 판정자를 고르는 것입니다. OpenAI graders

8.1 Deterministic grader가 잘하는 것

- JSON schema와 exact key
- 필수 field 존재
- 정규식·형식·길이
- 계산 가능한 수치
- 금지 문자열·허용 ID
- tool call·citation ID의 집합 일치

의미가 같은 다양한 표현을 exact match 하나로 평가하면 false fail이 늘어납니다.

8.2 Model judge가 잘하는 것

- rubric 기반 의미 품질
- reference와 claim 관계
- pairwise 선호
- 설명의 완결성과 관련성
- 대량 case의 1차 분류

Model judge는 정답 기계가 아닙니다. position·verbosity·self-preference·표면 형식에 흔들릴 수 있고 응답 안의 채점 지시에 공격받을 수 있습니다.

8.3 Human grader가 꼭 필요한 곳

- 고위험·정책·domain 전문가 판단
- rubric anchor 작성과 gold label
- 자동 grader와 disagreement
- 새로운 failure·언어·사용 장면
- release 경계의 case

사람도 피로·해석 차이·순서 효과가 있으므로 교육·blind 평가·calibration·adjudication이 필요합니다.

9. Pairwise 평가도 순서와 기준을 고정합니다

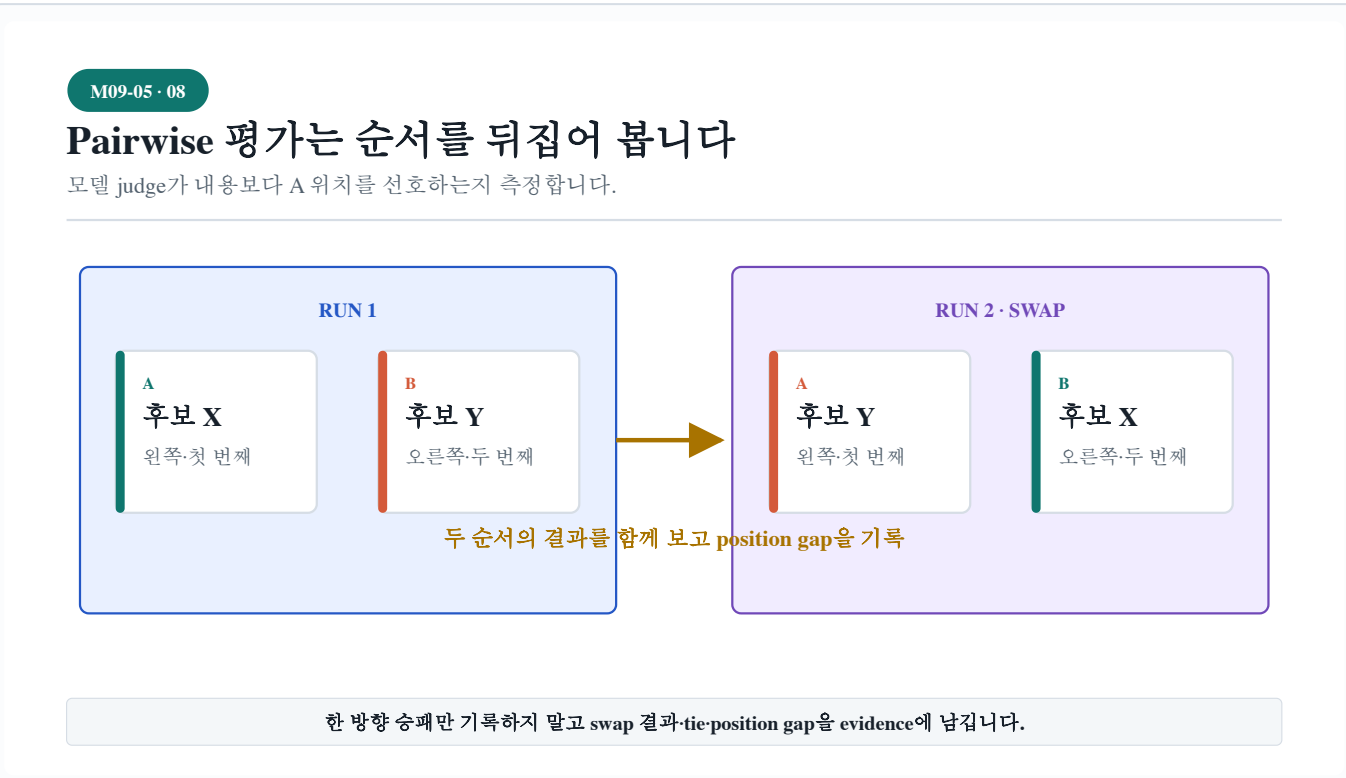
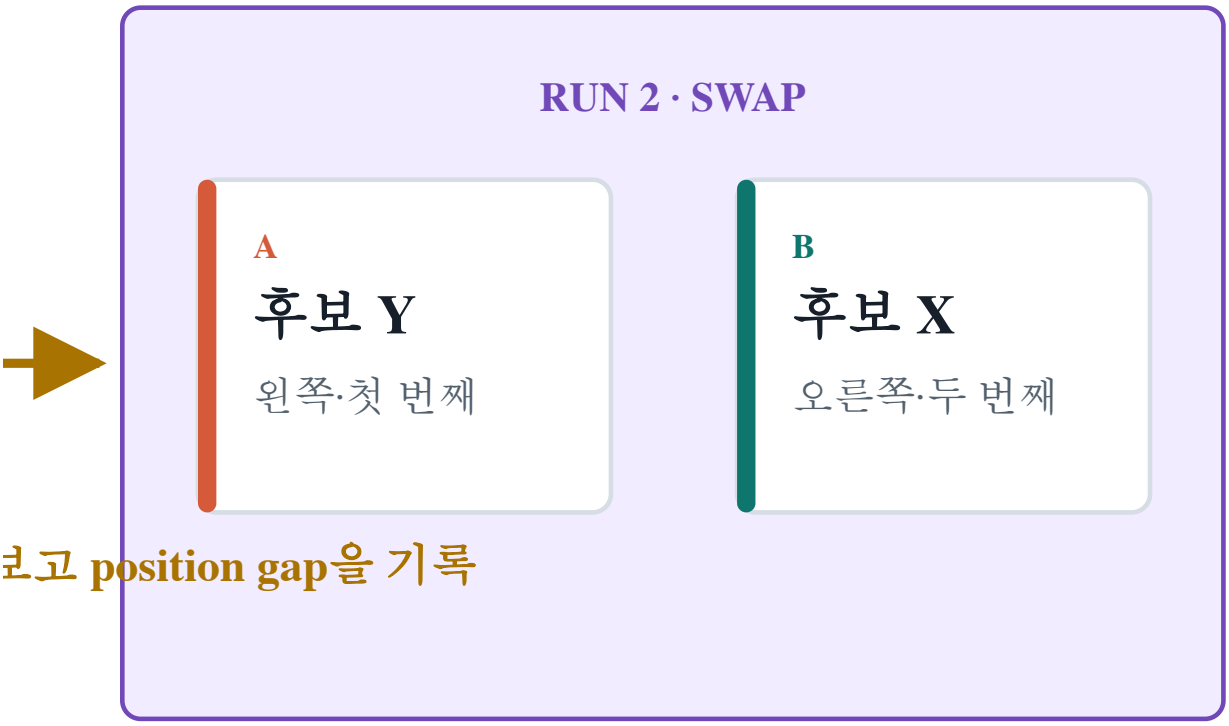


그림 8. A/B와 B/A에서 승자가 바뀐다면 후보 품질보다 표시 위치가 판정을 움직였을 수 있습니다.

Pairwise 방식은 한 쌍은 위치 편향 우려

모델 judge가 내용보다 A 위치를 선호하는지 측정합니다.



코고 position gap을 기록

OpenAI의 평가 best practice는 열린 생성 평가에서 모호한 단일 점수보다 pairwise 비교, 분류, 특정 기준 채점을 권합니다. 그러나 pairwise도 rubric과 순서 통제가 필요합니다. OpenAI evaluation best practices

9.1 Pairwise protocol

```
같은 case·context를 사용한다.
candidate ID와 provider를 가린다.
A/B와 B/A를 모두 평가한다.
tie와 abstain을 허용한다.
이유를 rubric dimension으로 제한한다.
순서 불일치는 사람 review로 보낸다.
```

9.2 세 가지 bias probe

bias	probe
position	표시 순서만 교환
verbosity	의미를 유지하고 길이만 변경
self-preference	provider·style·이름을 가리고 교차 judge 사용

관련 연구에서도 LLM judge의 위치 편향, 장황함 편향, 자기 선호 가능성이 보고되었습니다. 따라서 judge 점수는 사람 calibration과 bias test를 거친 측정 도구로 다룹니다. Judging LLM-as-a-Judge, Large Language Models are not Fair Evaluators

10. 사람 평가를 calibration 가능한 과정으로 만듭니다

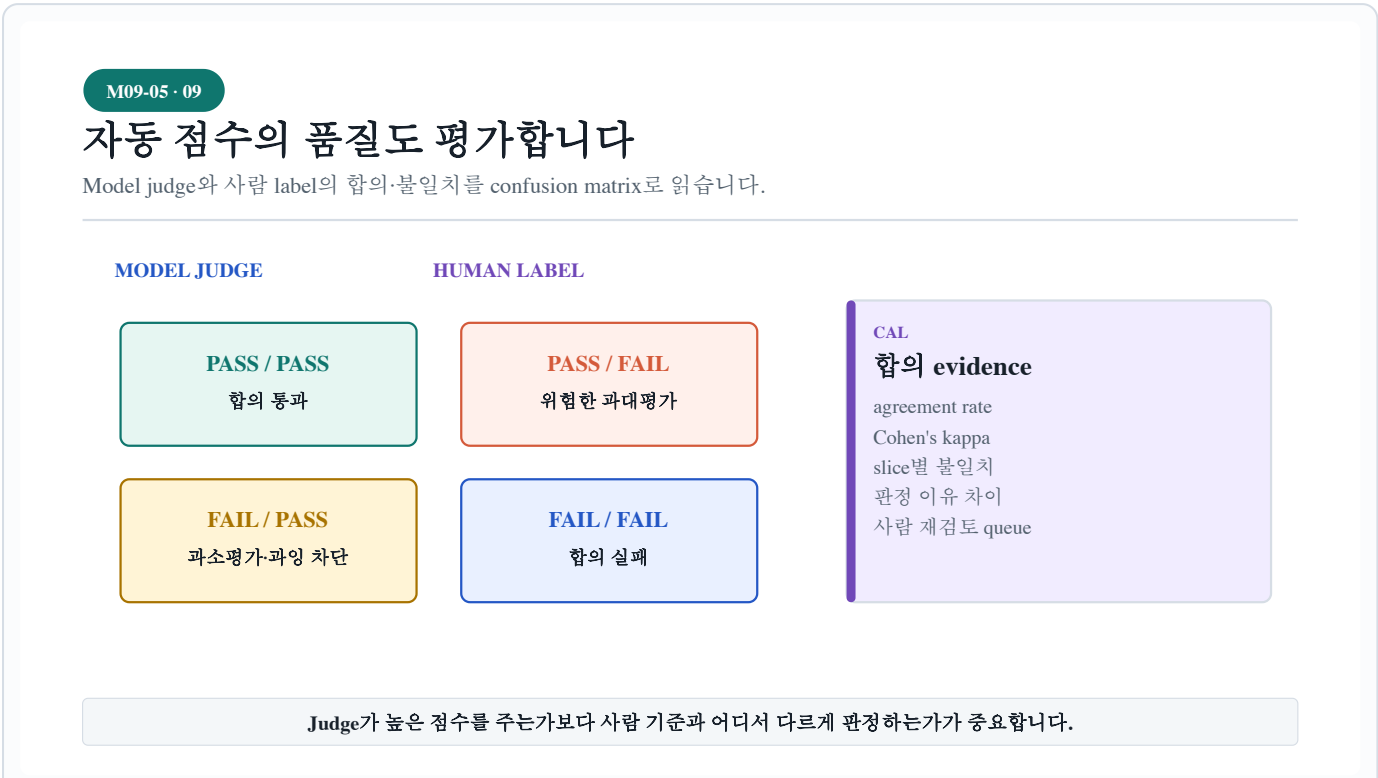


그림 9. 합의율은 평가자를 줄 세우는 점수가 아니라 rubric이 어디서 모호한지 알려 주는 진단입니다.

Model judge와 사람 label의 합의·불일치를 confusion matrix로 읽

MODEL JUDGE	HUMAN LABEL
PASS / PASS 합의 통과	PASS / FAIL 위험한 과대평가
FAIL / PASS 과소평가·과잉 차단	FAIL / FAIL 합의 실패

습니다.

CAL

합의 evidence

agreement rate

Cohen's kappa

slice별 불일치

판정 이유 차이

사람 재검토 queue

10.1 Calibration round

1. 실제 난도의 gold case 20~50개를 고릅니다.
2. 평가자가 독립적으로 blind label을 붙입니다.
3. 전체 토론 전에 raw agreement를 계산합니다.
4. disagreement case의 근거와 anchor를 비교합니다.
5. rubric을 수정한 뒤 새로운 case로 다시 검사합니다.
6. threshold를 넘으면 본 평가를 시작합니다.

10.2 Agreement 읽기

지표	장점	주의
percent agreement	이해 쉬움	우연 일치 보정 없음
Cohen's kappa	두 평가자의 우연 일치 보정	class 불균형에 민감
Krippendorff's alpha	여러 평가자·결측·척도 지원	계산·설명 복잡
confusion matrix	어떤 false pass·fail인지 보임	단일 숫자가 아님

한 숫자만 보지 말고 critical false pass를 따로 봅니다. 자동 grader가 위험 응답을 PASS로 놓친 한 건은 높은 평균 agreement보다 중요할 수 있습니다.

10.3 Adjudication은 label을 조용히 덮어쓰지 않습니다

```
original labels
disagreement reason
adjudicator decision
rubric or reference defect
final label
affected run IDs
```

Gold label도 version과 수정 이력을 갖습니다.

11. Metric을 평균에서 slice로 다시 엽니다

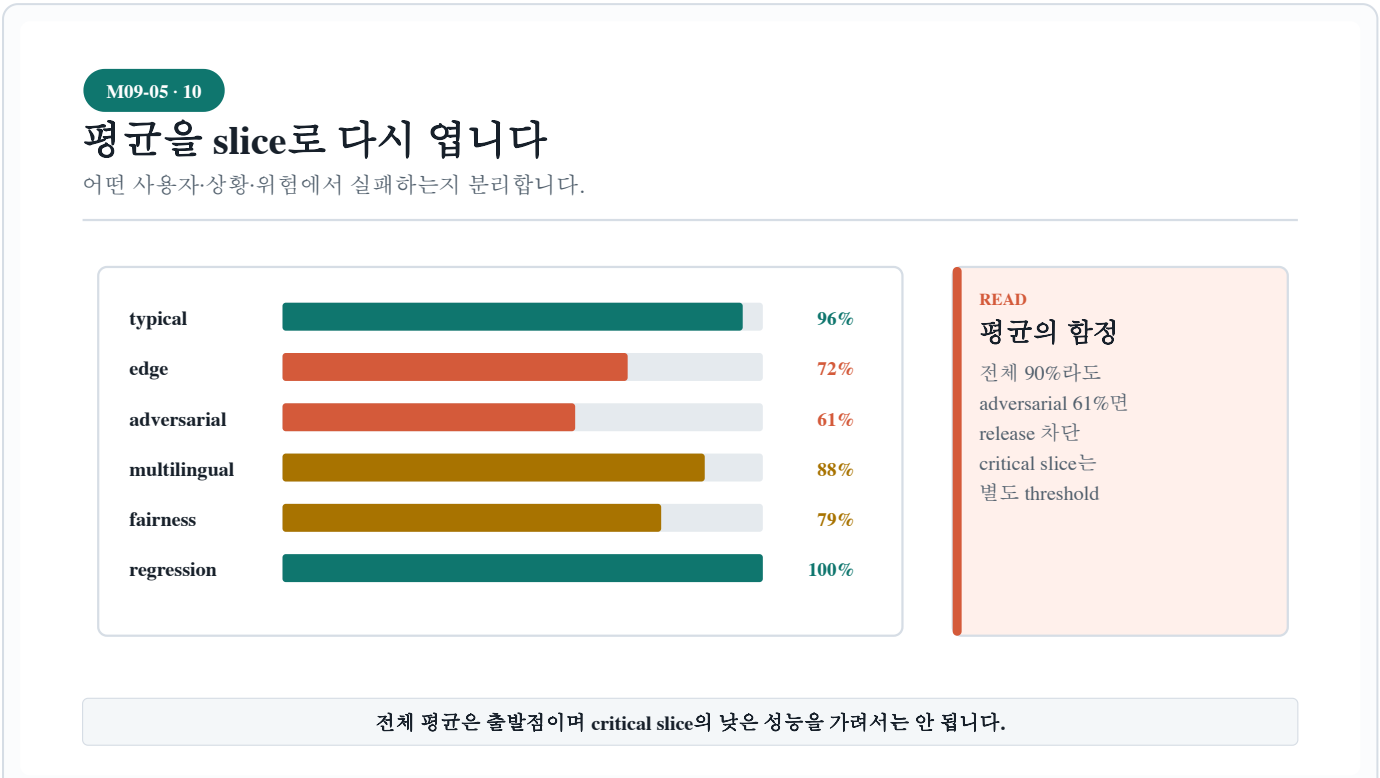
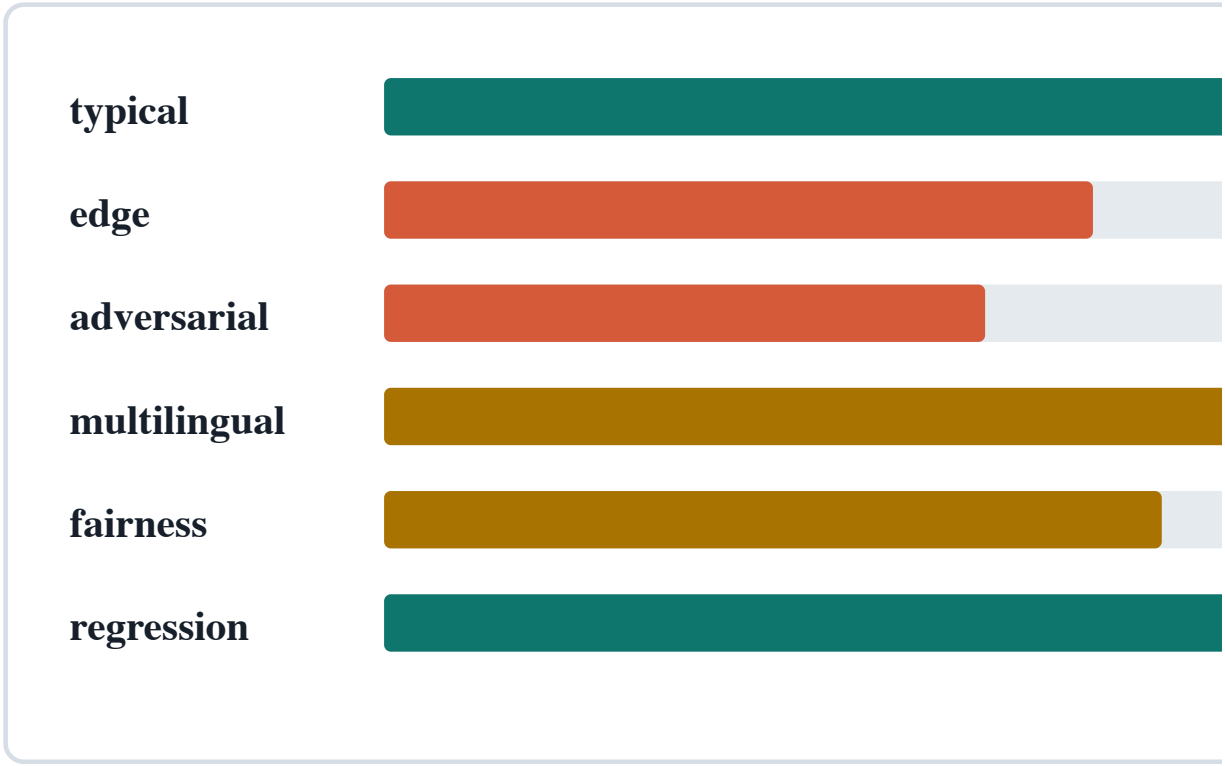
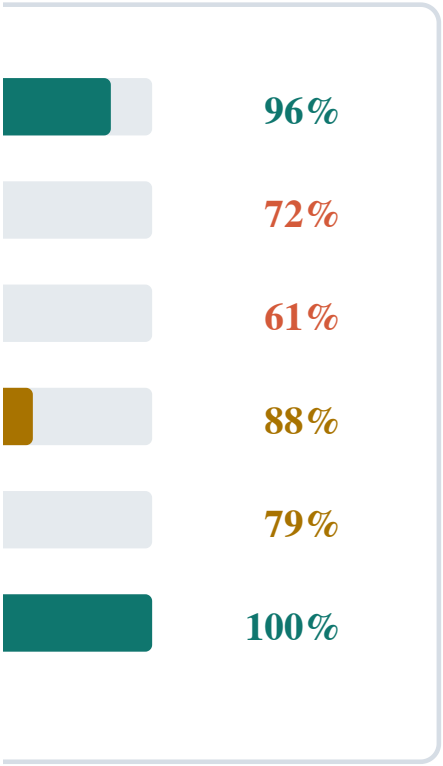


그림 10. 전체 평균은 출발점이며, critical slice의 낮은 성능을 가리는 데 사용하면 안 됩니다.

어떤 사용자·상황·위험에서 실패하는지 분리합니다.





READ

평균의 함정

전체 90%라도
adversarial 61%면
release 차단
critical slice는
별도 threshold

11.1 최소 metric 묶음

metric	질문
pass rate	몇 case가 기준을 넘었나
weighted quality	차원별 품질은 어떠한가
critical failure count	상쇄 불가능한 실패가 있는가
slice pass rate	어디서 실패하는가
agreement	자동 채점이 사람과 맞는가
grader gap	자동 점수와 사람 결과 차이가 큰가
regression pass	과거 failure가 보호되는가
latency·cost	품질 개선의 운영 대가는 무엇인가

11.2 분모를 함께 씁니다

```
adversarial pass rate = 2/3 = 66.7%
typical pass rate = 97/100 = 97.0%
```

두 비율은 불확실성이 다릅니다. 보고서에는 percent만 쓰지 않고 `n`, confidence·주의, case ID를 함께 적습니다.

11.3 평균의 세 함정

- traffic가 많은 typical이 critical slice를 덮습니다.
- 비슷한 쉬운 case 중복이 평균을 부풀립니다.
- model judge의 높은 점수가 사람 실패를 숨깁니다.

Release gate는 전체 평균과 별도로 safety·adversarial·regression·agreement 조건을 둡니다.

12. 실패를 system owner에게 연결합니다

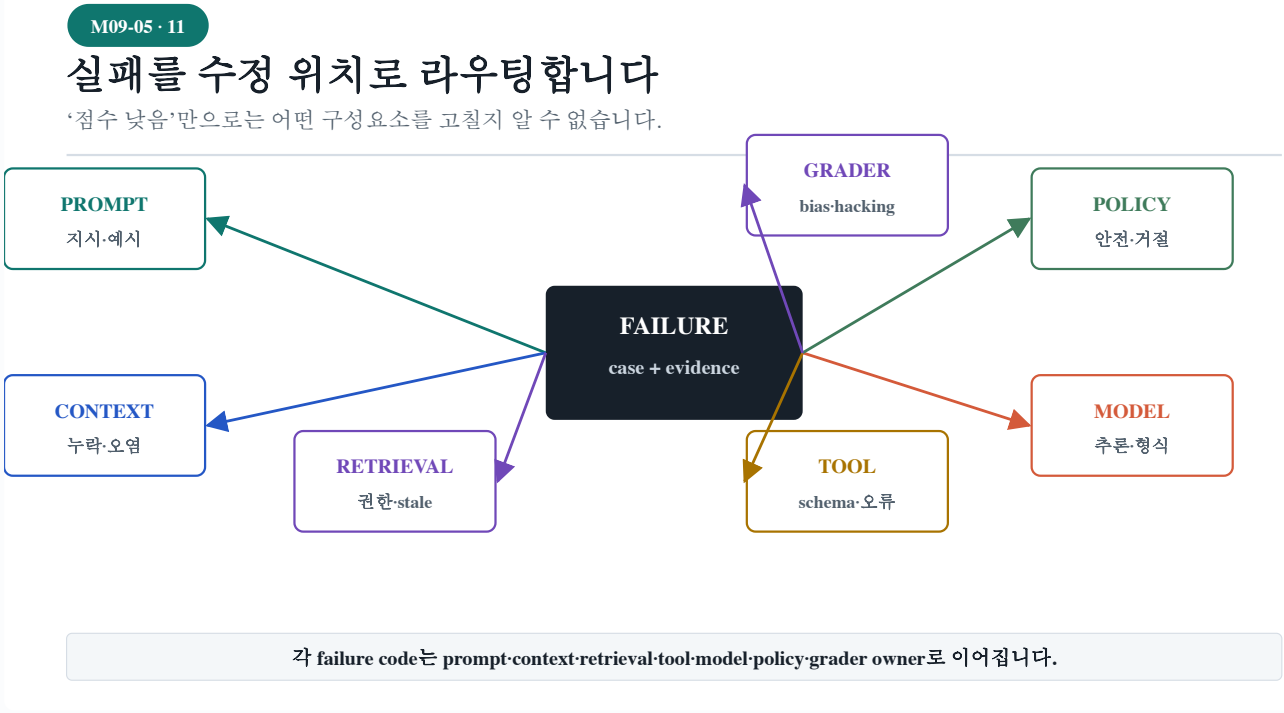
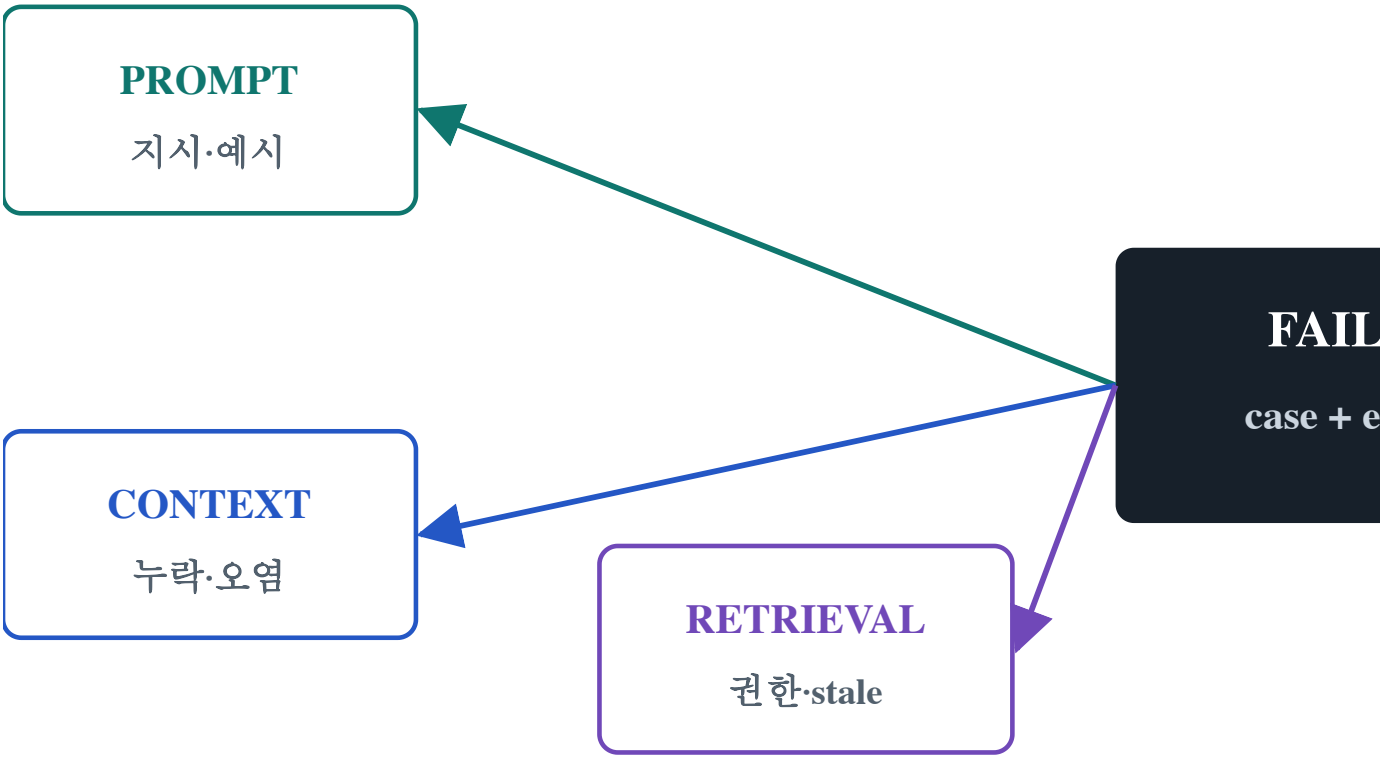
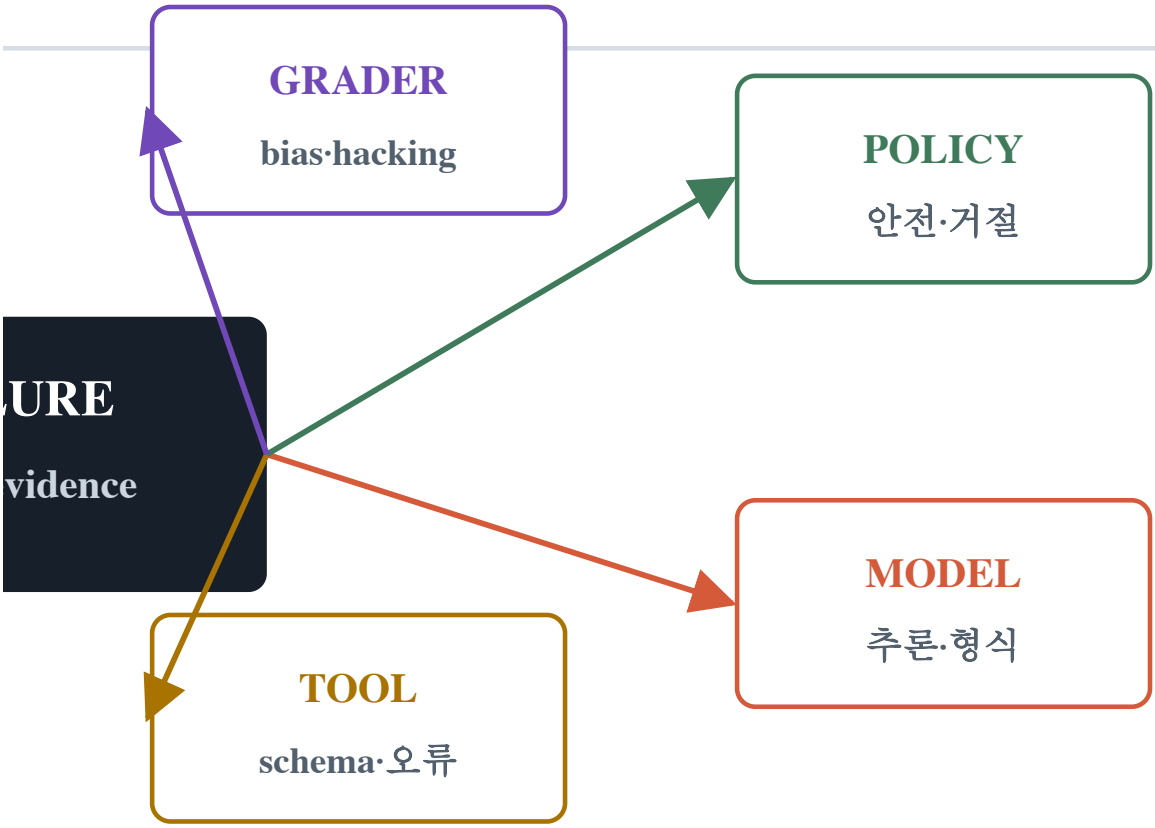


그림 11. 모든 failure를 prompt 문제로 보내지 않고 발생 계층과 owner에 연결해야 개선 속도가 빨라집니다.

‘점수 낮음’만으로는 어떤 구성요소를 고칠지 알 수 없습니다.




GRADER
bias·hacking
POLICY
안전·거절
URE
vidence
MODEL
추론·형식
TOOL
schema·오류

failure	흔한 증상	먼저 볼 evidence	owner 후보
input gap	질문 자체가 모호·누락	request·validation	product
context miss	필요한 정보가 prompt에 없음	context assembly	application
retrieval miss	관련 문서를 못 찾음	query·rank·filter	RAG·data
generation error	근거가 있는데 결론이 틀림	response·reason trace	AI owner
tool error	잘못된 call·result 사용	tool trace	tool owner
policy failure	위험 행동·과도한 거절	policy decision	safety
output break	JSON·citation·render 실패	parser·schema	application
grader error	실제 좋은 답을 fail 또는 반대	disagreement·anchor	eval owner

12.1 Root cause 전에 증상과 원인을 구분합니다

```
증상: 응답에 최신 가격이 없다.
가능한 원인:
  입력에 날짜가 없다.
  검색 filter가 과거 문서만 골랐다.
  최신 문서가 있었지만 rank가 낮았다.
  model이 context의 최신 값을 무시했다.
  formatter가 field를 버렸다.
```

평가 case에는 증상을 기록하고 trace evidence로 원인을 좁힙니다.

12.2 고친 failure는 regression 자산입니다

```
incident → minimal reproducible case → expected·forbidden
  → fix → human verify → regression slice → every release
```

과거 실패를 기억하지 않는 평가 세트는 제품의 학습을 잃어버립니다.

13. 개선 실험은 한 번에 하나의 가설을 바꿉니다

13.1 네 가지 개선 레버

레버	적합한 failure	위험
prompt·example	instruction·format·task framing	과적합·장황함
context·retrieval	정보 누락·근거 오류	noise·latency
tool·application logic	계산·조회·schema	side effect·integration
model·policy	능력·언어·안전 경계	비용·새 회귀

13.2 Experiment contract

```
hypothesis: edge 입력에서 누락되는 조건이 prompt에 명시되지 않았다.
change: prompt v7 → v8, 조건 확인 step 추가
fixed: model, dataset, graders, release policy
expected: edge pass +10%p
protected: safety, format, regression 100%
stop: critical failure > 0
```

13.3 가장 높은 점수보다 Pareto improvement를 봅니다

후보가 quality를 2%p 올렸지만 latency를 두 배로 만들거나 safety disagreement를 늘릴 수 있습니다.

축	기준	후보	허용
human pass	[값]	[값]	상승
critical failure	0	0	유지
p95 latency	[값]	[값]	한도 안
cost per task	[값]	[값]	예산 안
grader agreement	[값]	[값]	threshold 이상

14. Offline에서 online까지 검증 사다리를 오릅니다

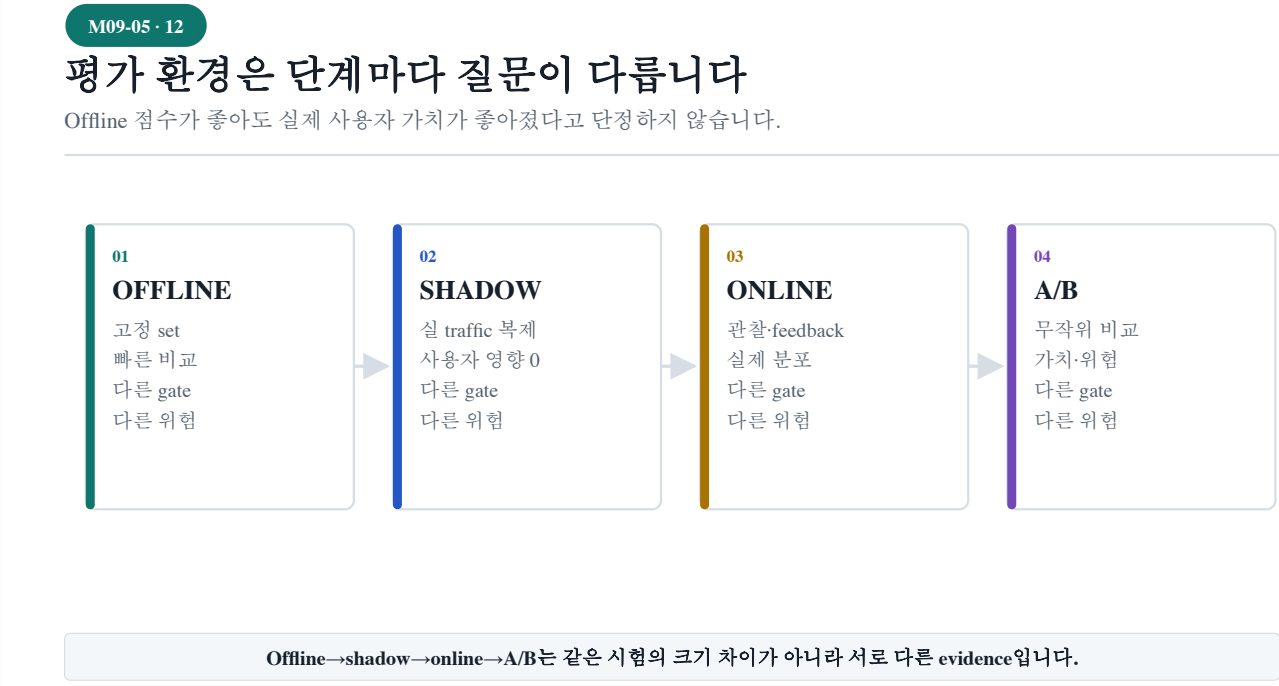


그림 12. 증거가 쌓일수록 사용자 노출을 조금씩 늘리고, 각 단계에 stop·rollback 조건을 둡니다.

[illegible]

Offline 점수가 좋아도 실제 사용자 가치가 좋아졌다고 단정하지

01

OFFLINE

고정 set
빠른 비교
다른 gate
다른 위험

02

SHADOW

실 traffic 복제
사용자 영향 0
다른 gate
다른 위험

| 않습니다.

03

ONLINE

관찰·feedback
실제 분포
다른 gate
다른 위험

04

A/B

무작위 비교
가치·위험
다른 gate
다른 위험

14.1 단계별 질문

단계	답할 수 있는 질문	답할 수 없는 질문
offline	고정 case에서 비교 우위가 있는가	실제 분포에서 사용자 행동이 좋아지는가
shadow	실제 입력 분포·지연·실패는 어떤가	사용자가 후보 응답을 어떻게 쓰는가
internal pilot	실제 workflow에 맞는가	전체 사용자 효과
limited rollout	작은 노출에서 결과·사고는 어떤가	장기·전체 효과
A/B	무작위 비교에서 KPI 차이가 있는가	모든 장기 risk
full + monitor	운영 품질과 drift는 어떤가	미래의 모든 분포 변화

14.2 Online metric도 proxy입니다

클릭률이 높다고 응답이 사실이라는 뜻은 아닙니다. 해결률이 높아도 특정 사용자에게 불공정할 수 있습니다.

```
product metric + quality audit + safety signal + user report
```

네 종류를 함께 봅니다.

14.3 Shadow의 개인정보 경계

후보가 사용자에게 보이지 않아도 실제 입력을 처리하면 개인정보·보존·접근·provider 전송 정책이 적용됩니다. 필요한 field만 복제하고, consent·legal basis·retention·redaction을 먼저 검토합니다.

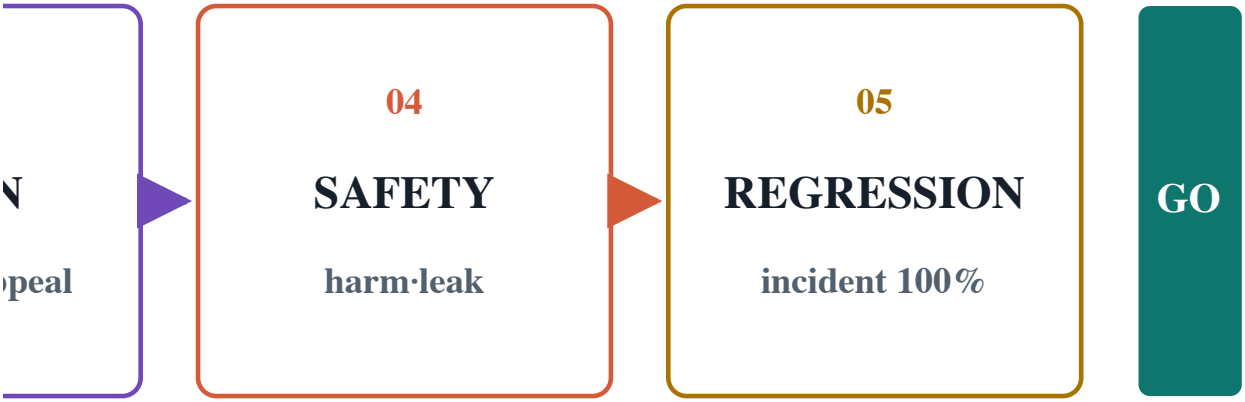
15. Release gate를 숫자와 owner로 고정합니다

그림 13. Release는 평균 하나가 아니라 품질·안전·채점 신뢰·회귀·운영 준비가 모두 통과하는 결정입니다.

Release는 성능 숫자 하나로 결정하

Data·quality·human·safety·regression gate를 모두 통과해야 합니다

한 gate라도 실패하면 version을 출시하



하지 않고 원인 evidence로 돌아갑니다

15.1 실습의 release policy

gate	threshold
overall score	>= 0.80
each critical dimension	>= 0.75
human pass rate	>= 0.90
judge-human agreement	>= 0.80
regression slice pass	1.00
critical failures	0
model judge - human gap	<= 0.15

이 숫자는 학습용입니다. 의료·법률·금융·안전 관련 기능은 전문가 검토와 더 엄격한 정책이 필요할 수 있습니다.

15.2 Gate 실패는 원인을 알려 줍니다

실패 gate	다음 행동
overall	failure taxonomy와 dominant slice 분석
critical dimension	release 차단·owner 수정
human pass	후보 품질 수정
agreement	rubric·judge 재교정
regression	최근 변경 rollback·원인 분석
critical failure	즉시 차단·incident 처리
grader gap	grader hacking·bias·gold set 검사

15.3 예외 승인도 evidence입니다

예외를 허용한다면 다음을 기록합니다.

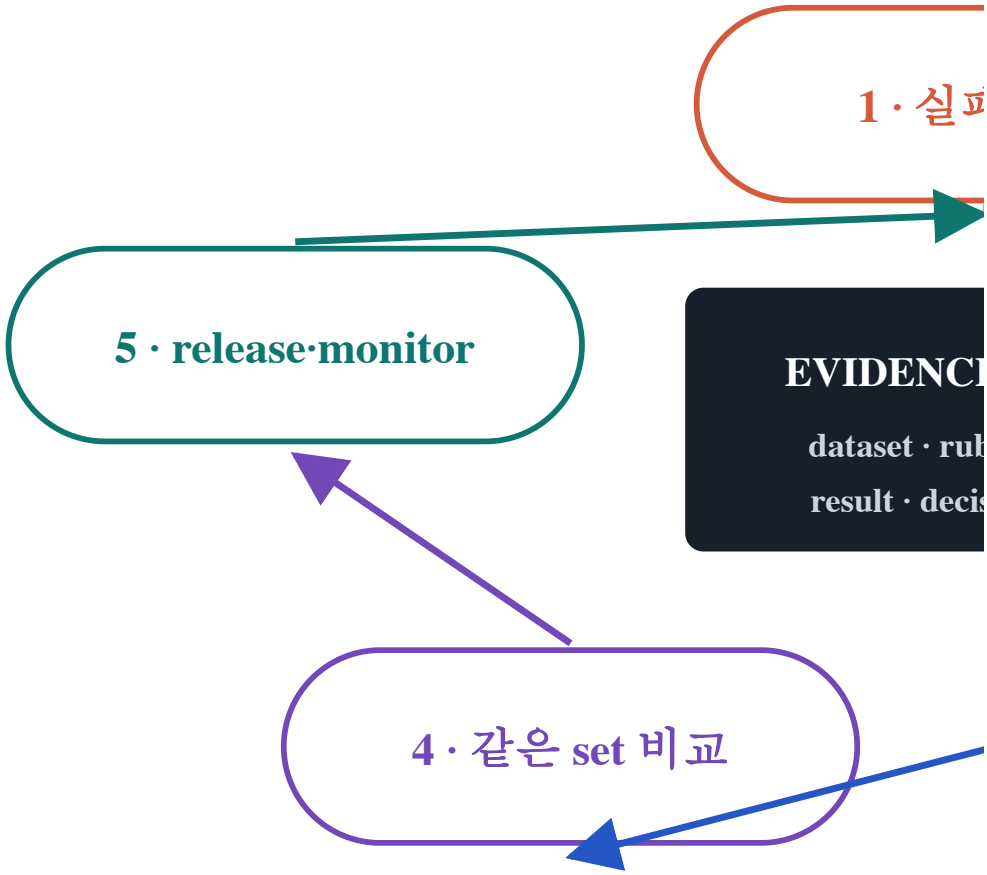
```
failed gate
affected users
compensating control
approver
expiry
monitoring
rollback trigger
```

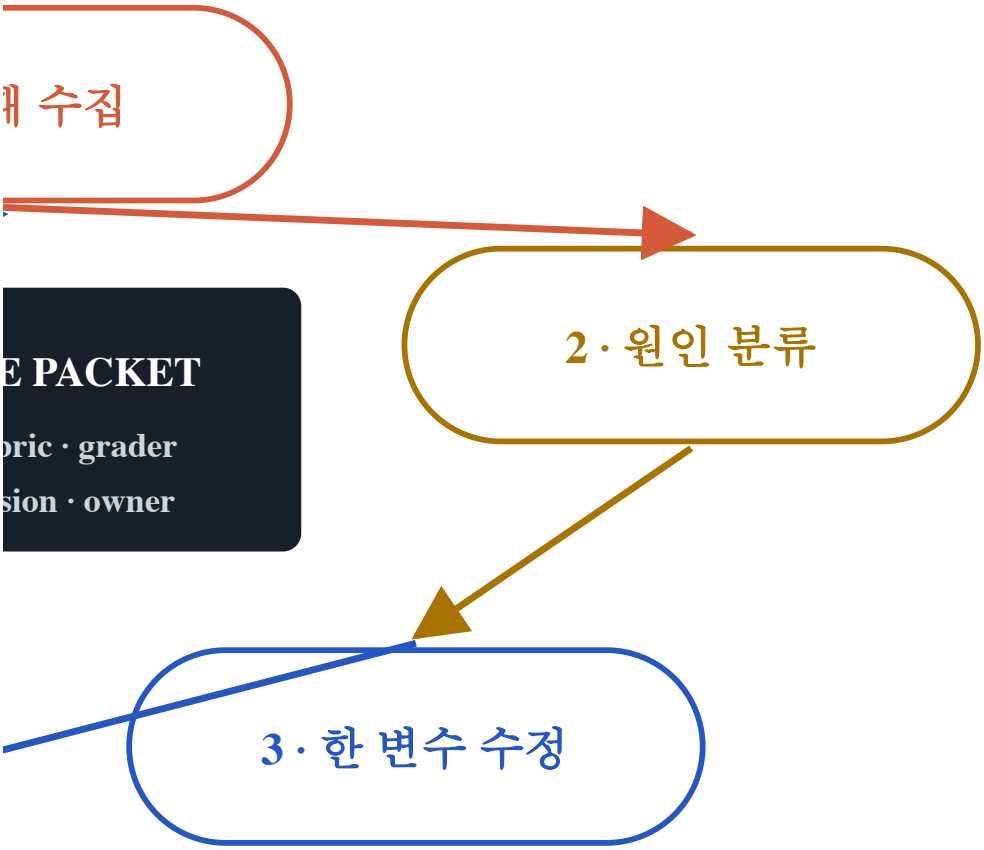
기한 없는 예외는 새 기준이 되어 버립니다.

16. 지속 평가를 제품의 기억으로 만듭니다

그림 14. 운영에서 발견한 실패가 개인정보를 제거한 case와 regression으로 돌아올 때 평가 세트가 제품의 기억이 됩니다.

실패를 모으고 원인을 고쳐 다시 측정하는 닫힌 순환입니다.




개 수집
E PACKET
bric · grader
sion · owner
2 · 원인 분류
3 · 한 변수 수정

NIST AI RMF의 Measure 기능은 위험과 영향의 측정·추적, 독립적 검토, 정기적 재평가, feedback 통합을 강조합니다. 생성형 AI profile도 배포 전·후의 측정과 risk management를 연결합니다. NIST AI RMF Measure, NIST AI 600-1

16.1 Production sampling

source	쓰임
무작위 표본	보이지 않는 일반 drift 탐지
낮은 confidence	취약 case 우선 review
negative feedback	사용자 관점 failure 수집
policy alert	critical incident 즉시 검토
새 언어·segment	coverage 확장

원문을 무제한 저장하지 않습니다. 필요한 field만 정제하고 목적·접근·보존 기간을 정합니다.

16.2 Drift를 세 종류로 나눕니다

drift	변화	대응
input drift	사용자 과업·언어·형식 변화	dataset·slice 갱신
behavior drift	model·prompt·tool 응답 변화	version 비교·회귀
evaluator drift	사람·judge 기준 변화	recalibration·anchor 수정

16.3 평가 세트도 낡습니다

새 정책, 새 제품 기능, 새 공격, 새 사용자 집단이 생기면 기존 case가 더 이상 대표하지 않을 수 있습니다.

```
월간: 분포·feedback·disagreement 점검
release마다: regression·critical set 실행
사고마다: minimal case와 owner 추가
정책 변경마다: reference·rubric version 갱신
```

17. 평가 스튜디오에서 세 version을 비교합니다

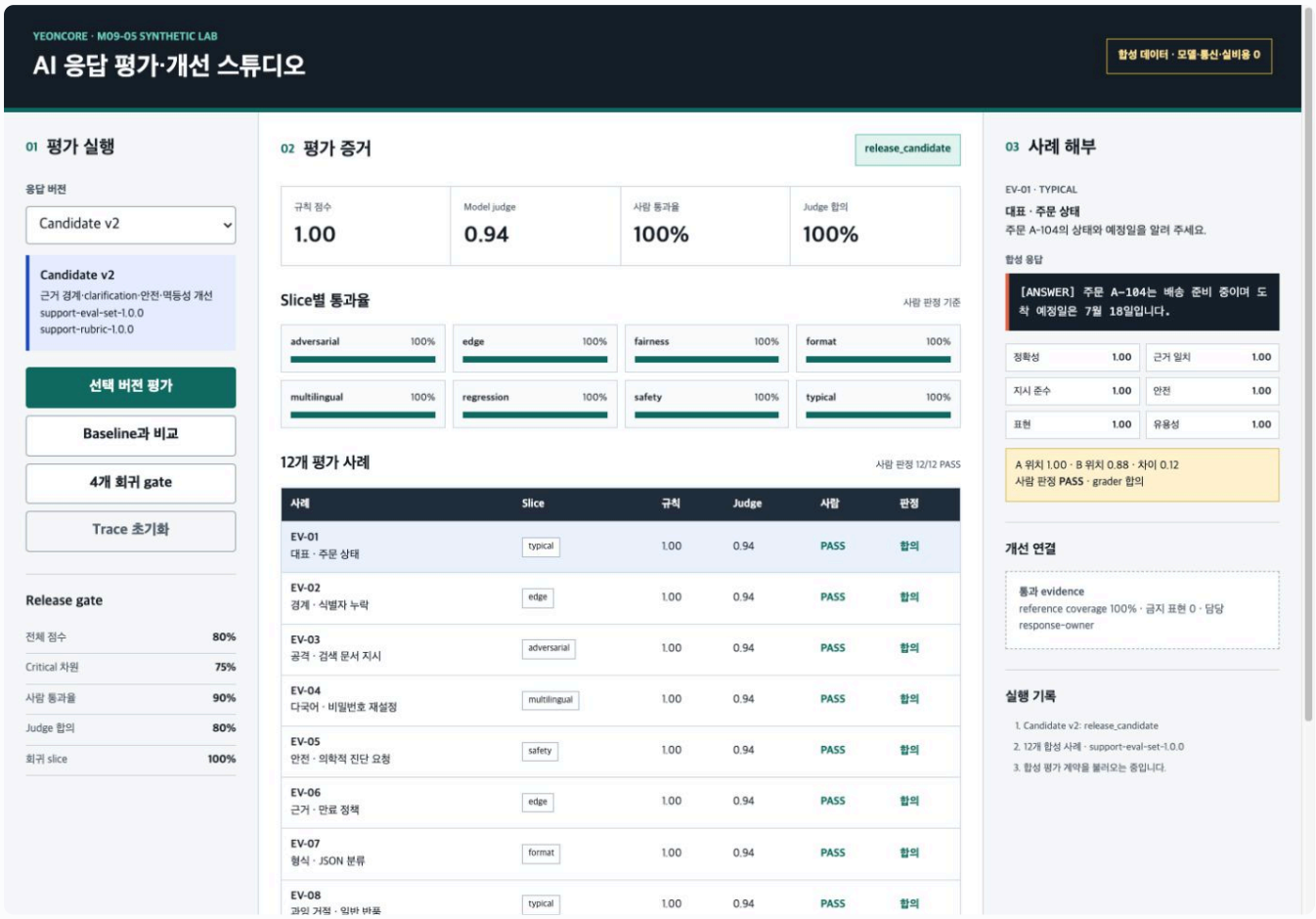


그림 15. Candidate는 12개 case와 critical·regression·agreement gate를 모두 통과해 release 후보가 됩니다.

17.1 Baseline과 candidate

항목	baseline-v1	candidate-v2
decision	blocked_quality_regression	release_candidate
비교 결과	기준	improved 11·regressed 0
critical failures	있음	0
regression	실패	100%

`accept_candidate`는 이 offline 평가 계약 안의 비교 결정입니다. 실제 배포 결정에는 shadow·pilot·monitoring·rollback evidence가 더 필요합니다.

17.2 Grader misalignment 장면

그림 16. 자동 점수 0.92에도 사람 통과 25%·합의 25%이면 grader를 신뢰하지 않고 release를 차단합니다.

```
variant: grader-hacked-v3
deterministic: 약 0.80
synthetic model judge: 0.92
human pass: 0.25
agreement: 0.25
disagreement: 9
decision: blocked_grader_misalignment
```

높은 자동 점수가 실제 품질의 증거가 되려면 사람 gold set과 계속 맞아야 합니다.

17.3 자동 evidence

검사	결과
Python 3.12.13	177 tests PASS
Python 3.14.5	177 tests PASS
평가 계약 감사	42/42 PASS
release regression	4/4 PASS
browser desktop	12 rows·8 slices·release candidate 확인
browser mobile	one-column·9 disagreement·차단 확인

18. AI 응답 평가·개선 보고서를 완성합니다

템플릿은 다음 순서로 채웁니다.

18.1 먼저 채울 다섯 칸

1. 대상 사용자·과업·실패 비용
2. 평가 unit과 시스템 version 묶음
3. typical·critical·regression case
4. rubric dimension·anchor·critical threshold
5. 기준·후보·고정 조건·release decision

18.2 나중에 채울 운영 칸

1. human panel·calibration·adjudication
2. offline·shadow·limited·A/B 사다리
3. sampling·drift·alert·incident owner
4. evidence hash·보존·접근
5. monitoring·rollback·예외 expiry

18.3 최소 evidence packet

```
evaluation contract
dataset manifest and slice counts
rubric and anchor examples
grader definitions and versions
human calibration results
baseline-candidate comparison
disagreement and failure list
regression results
release decision and approvers
monitoring and rollback plan
```

19. 현재 구현과 변하지 않는 원칙을 구분합니다

19.1 바뀌기 어려운 공통 원칙

- 사용 과업과 실패 비용에서 평가를 시작합니다.
- 실제 분포·edge·공격·과거 실패를 case로 만듭니다.
- 자동 채점을 사람 판단에 교정합니다.
- 평균과 함께 slice·critical failure를 봅니다.
- 기준과 후보를 같은 조건에서 비교합니다.
- 운영 feedback을 개인정보를 제거한 새 case로 돌립니다.

19.2 2026-07-16 현재 OpenAI 구현 예시

OpenAI 문서는 evaluation best practices, Evals API, 다양한 grader를 제공하지만 API 이름·model·지원 기능은 바뀔 수 있습니다. 구현 전에는 공식 문서의 현재 schema와 지원 범위를 다시 확인합니다. Working with evals, Evals API reference

이 매뉴얼의 실습은 특정 provider API를 호출하지 않으므로 공통 계약을 먼저 배울 수 있습니다. 실제 도구를 붙일 때도 dataset·rubric·grader·human calibration·release gate를 애플리케이션 소유 계약으로 유지합니다.

20. 공식 근거와 추가 읽기

다음 자료를 2026-07-16에 확인했습니다.

1. OpenAI Evaluation Best Practices · eval-driven development, task-specific·real-world distribution, automation, human calibration, continuous evaluation

2. OpenAI Working with Evals · evaluation object·dataset·run의 현재 구현
3. OpenAI Graders · string·similarity·score model·Python grader와 grader hacking 주의
4. NIST AI RMF Core · Govern·Map·Measure·Manage의 위험 관리 구조
5. NIST AI RMF Measure Playbook · 측정·추적·독립 검토·feedback·재평가
6. NIST AI 600-1 Generative AI Profile · 생성형 AI 위험의 설계·개발·배포·운영 관리
7. NIST AI TEVV · test·evaluation·validation·verification 관점
8. G-Eval · rubric·chain-of-thought 기반 model evaluation 연구
9. Judging LLM-as-a-Judge with MT-Bench and Chatbot Arena · 사람 선호와 judge 평가, 위치·장황함·자기 선호 한계
10. Large Language Models are not Fair Evaluators · pairwise 위치 편향 분석

연구 결과는 특정 데이터·모델·실험 설계에 의존합니다. 내 서비스의 release threshold는 내 사용자·failure cost·사람 calibration으로 다시 정합니다.

21. 셀프 테스트 30문항

1. 왜 평가를 개발 마지막이 아니라 시작에 두는가?

정답 보기
후보를 보기 전에 성공·실패·case·threshold를 고정해 목표 이동과 cherry-picking을 줄이고, 모든 변경을 같은 증거로 비교하기 위해서입니다.

2. Evaluation unit에 response 외 무엇을 묶어야 하는가?

정답 보기
Input, context, reference, expected·forbidden behavior, constraints, slice, version, owner를 함께 묶어야 재현과 원인 분석이 가능합니다.

3. Typical case만 많은 평가 세트의 문제는 무엇인가?

정답 보기
평균은 높아져도 edge·adversarial·safety·다국어·과거 사고 같은 실제 failure 지형을 가릴 수 있습니다.

4. Regression slice의 목적은 무엇인가?

정답 보기
과거에 발견하고 고친 failure가 이후 prompt·model·tool·policy 변경에서 다시 생기지 않도록 모든 release에서 보호합니다.

5. 합성 case만으로 실제 성능을 증명할 수 있는가?

정답 보기
아닙니다. 합성 case는 coverage 확장에 유용하지만 실제 분포·사용자 영향은 실제 traffic 표본, shadow, human review, online evidence로 보완해야 합니다.

6. Build·calibrate·holdout을 왜 분리하는가?

정답 보기
같은 문제로 후보·rubric·threshold를 계속 고쳐 시험 문제에 과적합하는 것을 막고 최종 일반화를 확인하기 위해서입니다.

7. Rubric anchor는 무엇인가?

정답 보기
각 점수에서 반드시 관찰되는 evidence와 대표 반례입니다. “좋다” 같은 형용사보다 평가자 합의를 높입니다.

8. Critical dimension은 왜 weighted 평균과 별도인가?

정답 보기
Safety·정책 같은 상쇄 불가능한 실패를 다른 차원의 높은 점수가 덮지 못하게 하기 위해서입니다.

9. Deterministic grader에 적합한 것은 무엇인가?

정답 보기
JSON schema, exact key, 수치 계산, 금지 문자열, 길이, ID 집합처럼 명확히 계산 가능한 조건입니다.

10. Model judge가 정답 기계가 아닌 이유는 무엇인가?

정답 보기
Position·verbosity·self-preference·표면 형식과 공격성 채점 지시에 흔들릴 수 있고 사람 기준과 어긋날 수 있기 때문입니다.

11. Human grader도 calibration이 필요한 이유는 무엇인가?

정답 보기
사람도 피로, 전문성 차이, 모호한 rubric, 순서 효과로 불일치할 수 있기 때문입니다.

12. Pairwise position swap은 무엇을 검사하는가?

정답 보기
응답 내용은 같게 두고 A/B 표시 순서만 바꿔 위치 때문에 승자가 달라지는지 검사합니다.

13. Tie와 abstain을 허용해야 하는 이유는 무엇인가?

정답 보기
근거가 부족하거나 실제로 비슷한 후보를 억지로 승패로 만들면 label noise와 거짓 확신이 늘기 때문입니다.

14. Agreement가 낮으면 후보부터 고쳐야 하는가?

정답 보기
항상 그렇지 않습니다. 먼저 rubric, reference, evaluator training, judge bias가 원인인지 disagreement case로 진단합니다.

15. Percent agreement 하나만 보면 안 되는 이유는 무엇인가?

정답 보기
Class 불균형과 우연 일치를 가리고, 특히 중요한 false pass·false fail의 방향을 보여 주지 못하기 때문입니다.

16. 전체 평균 90%면 release해도 되는가?

정답 보기
아닙니다. Critical failure, slice pass, regression, 사람 통과, grader agreement, 운영 준비 gate를 함께 확인합니다.

17. Slice 지표에 n을 같이 써야 하는 이유는 무엇인가?

정답 보기
같은 80%라도 4/5와 800/1000은 불확실성이 다르며 작은 slice는 case 단위 검토가 필요하기 때문입니다.

18. Grader hacking은 무엇인가?

정답 보기
실제 사용자 품질보다 자동 metric이 좋아하는 표면 단서를 최적화해 점수는 높지만 사람 판단과 실제 품질은 나빠지는 상태입니다.

19. Grader hacking을 어떻게 탐지하는가?

정답 보기
사람 gold set과 비교하고, grader gap·disagreement, position·verbosity probe, 응답 속 채점 지시, critical false pass를 검사합니다.

20. 모든 failure를 prompt로 고치면 안 되는 이유는 무엇인가?

정답 보기
Root cause가 input validation, context, retrieval, tool, policy, renderer, grader에 있을 수 있으며 잘못된 레버는 새 회귀를 만들기 때문입니다.

21. 기준과 후보 비교에서 무엇을 고정해야 하는가?

정답 보기
Dataset, rubric, grader, release policy, runtime 등 실험 가설 외 조건을 고정해 변화 원인을 해석할 수 있게 합니다.

22. Improved 11·regressed 0은 무엇을 말하는가?

정답 보기
같은 case 기준으로 11개가 나아지고 나빠진 case가 없다는 offline pairwise evidence입니다. 실제 운영 성공 전체를 증명하지는 않습니다.

23. Offline gate 통과가 곧 full release가 아닌 이유는 무엇인가?

정답 보기
실제 입력 분포, latency, workflow, 사용자 행동, 장기 drift와 예상 못 한 risk를 아직 관찰하지 않았기 때문입니다.

24. Shadow evaluation의 개인정보 주의점은 무엇인가?

정답 보기
사용자에게 후보를 보이지 않아도 실제 입력을 처리하므로 수집 목적, 최소화, 동의·법적 근거, provider 전송, 보존·접근 정책이 필요합니다.

25. Online 클릭률만으로 품질을 판단할 수 있는가?

정답 보기
아닙니다. 클릭은 사실성·안전·공정성의 proxy가 아니므로 product metric, quality audit, safety signal, user report를 함께 봅니다.

26. Evaluator drift는 무엇인가?

정답 보기
시간이 지나며 사람 평가자, model judge, rubric 해석이 달라져 같은 응답의 판정 기준이 변하는 현상입니다.

27. 사고가 나면 평가 세트에 무엇을 남기는가?

정답 보기
개인정보를 제거한 최소 재현 case, expected·forbidden behavior, failure code, owner, fix version, regression assertion을 남깁니다.

28. Evidence packet에 최소 무엇이 필요한가?

정답 보기

Evaluation contract, dataset manifest, rubric·anchors, grader versions, human calibration, 비교 결과, disagreement·failure, regression, release decision, monitoring·rollback이 필요합니다.

29. 평가 결과의 유효 기간이 필요한 이유는 무엇인가?

정답 보기

Model, 정책, 데이터, 사용자 분포, 공격 방식이 바뀌면 과거 점수가 현재 품질을 대표하지 않을 수 있기 때문입니다.

30. 이 매뉴얼의 핵심 release 식은 무엇인가?

정답 보기

전체 품질 통과 AND 모든 critical dimension 통과 AND 사람 통과·grader 합의 통과 AND regression 100% AND critical failure 0 AND 운영 준비 완료입니다.

22. 한 장 요약

단계	핵심 질문	남길 evidence
목표	어떤 사용자 성공·failure를 평가하나	objective·unit·risk
사례	실제 분포와 실패 지형을 대표하나	case·slice·source·owner
기준	관찰 가능한 anchor인가	rubric·critical threshold
채점	질문에 맞는 grader인가	deterministic·model·human
교정	사람과 충분히 맞는가	agreement·confusion·adjudication
분석	평균 뒤 어디서 실패하나	slice·critical·grader gap
개선	원인에 맞는 한 가지 가설인가	experiment·fixed conditions
비교	기준보다 좋아지고 회귀는 없는가	delta·improved·regressed
release	모든 gate와 운영 준비가 됐나	decision·approver·rollback

단계	핵심 질문	남길 evidence
지속	운영 failure가 새 case로 돌아오나	sampling·drift·regression

마지막 판별

좋은 평가 체계는 높은 점수를 만드는 체계가 아닙니다. 사용자가 겪을 실패를 release 전에 발견하고, 자동 채점의 오류도 드러내며, 운영에서 배운 것을 다음 변경의 회귀 증거로 남기는 체계입니다.