

# Evals That Actually Work

A 5-Stage Framework for Production AI Quality

# "How do you know your AI is good?"

If your answer is "we tested it manually" or "it looked fine in review" — you don't have an eval strategy.

— THE PROBLEM

## Without evals, you're flying blind

### Without Evals

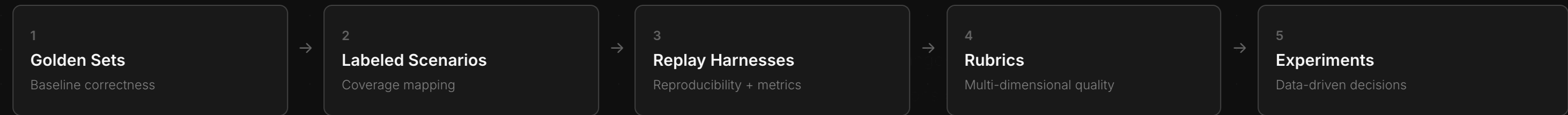
- Guess if changes helped
- Find regressions in production
- Debate quality subjectively
- Ship and hope

### With Evals

- Measure if changes helped
- Catch regressions before shipping
- Compare quality with data
- Ship and know

THE FRAMEWORK

# Five stages, each building on the last



Start at Stage 1. Add stages as your system matures.

Five stages, each building on the last

STAGE 01

# Golden Sets

Golden Sets

*Your first line of defense*

Labeled Scenarios

Predefined scenarios

Replay Harnesses

Replay and drive metrics

Rubrics

Plan and evaluate work

Experiments

Plan and evaluate work

Stage 1: Add stages as your system matures.

---

GOLDEN SETS

## Golden sets define what "correct" looks like

Small (10–20 cases). Fast to run. If these fail, something is fundamentally broken.

GOLDEN\_DATA.YAML

```
- id: "gs-001"
  query: "What is our remote work policy?"

  expected_tools:
    - vector_search

  expected_sources:
    - remote_work_policy.md

  must_contain:
    - "remote"
    - "core hours"

  must_not_contain:
    - "I don't know"
    - "no information"
```

— GOLDEN SETS

Golden sets define what "correct" looks like

— GOLDEN SETS

to run. If these fail, something is fundamentally broken.

Golden sets use four types of checks

CHECK	WHAT IT CATCHES
Tool selection	Agent used the wrong tool
Source citation	Agent cited the wrong document
Content validation	Response is missing key facts
Negative validation	Agent hallucinated or gave up

These are **code evals** — deterministic, binary, no LLM needed.

## — GOLDEN SETS

# Deterministic checks beat probabilistic judges

```
EVALUATOR.PY

# Tool selection
assert "vector_search" in actual_tools      # ✓ or x

# Source citation
assert "refund_policy.md" in response_text  # ✓ or x

# Content validation
assert "30-day" in response_text            # ✓ or x
assert "$500" in response_text              # ✓ or x

# Negative validation
assert "I don't know" not in response_text  # ✓ or x
```

Zero API cost. Zero ambiguity. Run after every commit.



# Stochastic checks beat probabilistic judges

— GOLDEN SETS

## Four rules for golden sets that stay useful

Start small	10–20 quality cases beats 100 sloppy ones
Run on every commit	These are your regression tests
Add from production bugs	Every bug becomes a test case
Never	Change expected output just to make tests pass

5 rules for golden sets that stay useful

STAGE 02

# Labeled Scenarios

run on every commit

*Coverage mapping*

from production bugs

— LABELED SCENARIOS

## Labeled scenarios are golden set cases with tags

SCENARIOS.YAML

```
- id: "sc-m-001"
  query: "What's our refund policy and how many refunds last quarter?"
  expected_tools: ["vector_search", "sql_query"]
  category:      multi_tool
  subcategory:   vector_and_sql
  difficulty:    straightforward
```

The tags don't change how the test runs — they change what the results tell you.

— LABELED SCENARIOS are golden set cases with tags

# Organize by category — empty cells show you where to write tests next

	vector	sql	jira	slack	multi
straightforward	3/3	2/3	2/2	2/2	1/1
ambiguous	1/1	1/2	0/1	1/1	0/1
edge_case	1/1	1/1	1/1	--	--

ge how the test runs — they change what the results tell you.

— LABELED SCENARIOS

# Golden sets and labeled scenarios answer different questions

Golden Sets:

"Does it work?"

→ correctness

Labeled Scenarios:

"Does it work for all types?"

→ coverage

	100% pass	90% pass	80% pass	70% pass	60% pass		GOLDEN SETS		LABELED SCENARIOS
Standard	3/3	2/3	3/3	3/3	3/3				
Size	1/1	1/2	0/1	1/1			10–20		30–100+
All must pass?							Yes		No
When to run							Every commit		Every release

Golden sets and labeled scenarios answer different questions

STAGE 03      "Does it work?" → correctness  
scenarios: "Does it work for all types?" → coverage

# Replay Harnesses

*Reproducibility + rich metrics*

GOLDEN SETS

10-20

Yes

Every build

LABELLED SCENARIOS

20-100+

No

Every release

---

REPLAY HARNESSES

## Record once. Score anytime.

Capture a real session to a JSON fixture. Evaluate that frozen snapshot whenever you want — immediately, next week, or after a human has annotated ground truth.

```
RECORDER.PY / PLAYER.PY

# Record once (costs tokens)
session = record_session(
    query="What's our refund policy and how many refunds in Q4?",
    session_id="refund-001"
)
# → saves to fixtures/refund-001.json

# Replay forever (costs nothing)
replayed = replay_session("refund-001")
scores   = evaluate_session(replayed)
```

**Record production examples.** Real queries make the best test cases.

REPLAY HARNESSES

Record once. Score anytime.

# Stage 3 introduces ML-grade metrics

Record a session to a JSON fixture. Evaluate that frozen snapshot whenever you want — immediately, next week, or after a human has annotated ground truth.

METRIC	WHAT IT MEASURES
Precision	How many retrieved docs are relevant?
Recall	How many relevant docs were retrieved?
Groundedness	Is the response grounded in sources?
Faithfulness	Does it stay true to sources (no hallucination)?
Tool Accuracy	Did it use the correct tools?

Groundedness and faithfulness require an LLM judge — which we'll cover next.

Record production examples. Real queries make the best test cases



### 3 introduces ML-grade metrics

LLM AS A JUDGE

# Use it right, or don't use it

- What it measures
  - How many retrieved docs are relevant?
  - How many relevant docs were retrieved?
  - Is the response grounded in sources?
  - Does it stay true to sources (no hallucination)?
  - Did it use the correct tools?

Thoroughness require an LLM judge — which we'll cover next.

— LLM AS A JUDGE

## The judge checks claims against sources

```
# Is this claim supported by the source?

claim:  "We offer 30-day refunds"
source: refund_policy.md → "...30-day refund window..."
result: ✓ grounded

claim:  "We offer 60-day refunds"
source: refund_policy.md → "...30-day refund window..."
result: ✗ not grounded (hallucination)
```

The judge makes a binary call per claim, then aggregates into a groundedness score.

— LLM AS A JUDGE

## Calibrate before you ship the judge

CALIBRATION.PY

```
# Step 1: Score 20 examples by hand
human_scores = [4, 3, 5, 2, 5, 4, 3, ...]

# Step 2: Run the LLM judge on the same examples
llm_scores    = [4, 4, 5, 3, 5, 3, 3, ...]

# Step 3: Check correlation
correlation(human_scores, llm_scores)
# If < 0.8, your rubric is broken – fix it before trusting the judge
```

A judge with a bad rubric produces **confident, wrong scores**.

— LLM AS A JUDGE

# LLM judges fail when criteria are vague

<div>one 20 examples by hand</div> <div>✗ VAGUE</div> <div>sc = [4, 3, 5, 2, 5, 4, 3, ...]</div>	<div>✓ SPECIFIC</div>
"Response is helpful"	"User could act on this without follow-up"
"Demonstrates strategic thinking"	"Identifies ≥2 trade-offs with concrete examples"
"Good quality"	"All facts verifiable from cited sources"

If you can't describe what 5/5 looks like in concrete terms, the judge can't score it — and neither can a human.

→ produces confident, wrong scores.

judges fail when criteria are vague

STAGE 04

# Rubrics

*How good, not just whether it passed*

what 5/5 looks like in concrete terms,  
and neither can a human

specifying

User could act on this without follow up?

Identifies 12 trade-offs with concrete examples?

All facts verifiable from cited sources?

— RUBRICS

# Rubrics score across four weighted dimensions

DIMENSION	WEIGHT	QUESTION
Relevance	30%	Does it address the question?
Accuracy	40%	Are the facts correct?
Completeness	20%	Does it fully answer?
Clarity	10%	Is it easy to understand?

Weighted average → single quality score. Track trends. A 5% drop in accuracy is a red flag.

— RUBRICS

## Every score needs an explicit anchor

RUBRICS.YAML

```
accuracy:
  weight: 0.4
  scores:
    5: "All facts correct and verifiable from cited sources"
    3: "Mostly correct with one minor inaccuracy"
    1: "Contains significant errors or misleading information"
    0: "Completely incorrect or fabricated"
```

No anchor = no consistency. A judge that interprets "3" differently each run is useless.

— RUBRICS

score needs an explicit anchor

# Score thresholds tell you what action to take

SCORE	QUALITY	ACTION
4.5–5.0	Excellent	Ship it
3.5–4.4	Good	Minor tweaks
2.5–3.4	Acceptable	Review and improve
1.5–2.4	Poor	Significant work needed
0–1.4	Critical	Stop. Fix now.



the thresholds tell you what action to take

STAGE 05

# Experiments

*Make decisions with data*

QUALITY

Good

Acceptable

Poor

Critical

ACTION

Stop it

Minor tweaks

Review and improve

Significant work needed

Stop, Fix now

— EXPERIMENTS

# Experiments replace intuition with evidence

Variant	Pass %	Rubric	Latency	Cost
baseline	87%	4.1/5	1.2s	\$0.003
gpt-4o	93%	4.5/5	2.1s	\$0.015
new_prompt	91%	4.3/5	1.3s	\$0.003

`new_prompt` gets 91% of gpt-4o quality at 20% of the cost. That's a data-driven decision.

— EXPERIMENTS

# Experiments replace intuition with evidence

## One variable at a time

One change per experiment	Isolate variables to understand impact		
Same test set every time	4.1 / 5.0	Apples to apples	\$0.003 / \$0.010
Track cost	21% / 4.3 / 5.0	6% quality gain at 5× cost may not be worth it	
Version your prompts	Store them as files; commit to git		

of gpt-4o quality at 20% of the cost. That's a data-driven decision.

one variable at a time

WHAT NOT TO DO

# The three eval anti-patterns

we test every time

high cost

your prompts

— ANTI-PATTERN 1

# The Likert trap

"Rate the quality of this response: 1 (poor) to 5 (excellent)"	
What happens	Human A gives "3" for adequate answers. Human B gives "4". Aggregated scores are noise, not signal.
Fix	Define every point on the scale. If you can't write the anchor, you haven't done the work.

— ANTI-PATTERN 2

# Vague criteria

"The response demonstrates strategic thinking"	
Problems	"Strategic thinking" is undefined. Two evaluators will score differently. An LLM judge will hallucinate a definition. You can't tell what improvement looks like.
Fix	Make it falsifiable: "Response identifies ≥2 explicit trade-offs with concrete examples"

— ANTI-PATTERN 3

# Ambiguous ranges

"The response length should be between 300 and 500 tokens"	
Problem	A 450-token response that misses the question passes. A 250-token response that perfectly answers it fails. This criterion measures form, not quality.
Fix	Describe what good looks like: "Answers all parts of the question; does not include unrequested information"

— ANTI-PATTERNS

## Vague ranges

# They all describe form, not quality

“The length should be between 100 and 500 tokens!”

Length, scale labels, and vague terms are **proxies** for quality — not quality itself.

**The rule:** A criterion is done when you can write a concrete example that unambiguously passes and one that unambiguously fails.

If you can't do that, keep writing.



they all describe form, not quality

GOOD PRACTICES

# The opinionated version

ie: ...ion is done when you can write a concrete example that unambiguously passes and one that unambiguously fails.

keep writing

---

GOOD PRACTICES

## Use binary checks where you can

```
EVALUATOR.PY
```

```
# Specific. Deterministic. Fast.  
assert "vector_search" in actual_tools  
assert "refund_policy.md" in response  
assert "I don't know" not in response  
assert "$500 annual stipend" in response
```

Binary checks have zero calibration cost, zero API cost, and produce the same result every run.

Reserve LLM judges for what can't be checked programmatically.

GOOD PRACTICES

checks where you can

# Write rubric anchors before you run any evals

STEP	WHY
1. Write score anchors (0, 3, 5 minimum)	Forces you to define quality
2. Score 20 examples by hand	Creates ground truth
3. Run LLM judge on same examples	Measures calibration
4. Adjust until correlation $\geq 0.8$	Validates the judge
5. Then run at scale	Now you can trust it

— GOOD PRACTICES

# Match the eval to the moment

Every commit	→ Golden sets	(5 min, catch regressions)
Every release	→ Labeled scenarios	(coverage check)
Weekly	→ Replay harnesses	(deep quality analysis)
Before shipping	→ Rubric evals	(multi-dimensional scoring)
On any change	→ Experiment	(validate the hypothesis)

The earlier you catch a regression, the cheaper it is to fix.

GOOD PRACTICES

# Build your eval stack in six steps

- 1

**Write 10–15 golden cases today**  
Tools, sources, content checks, negative validation
- 2

**Label them by query type and difficulty**  
Category, subcategory, complexity
- 3

**Record real production sessions as fixtures**  
Real queries make the best test cases
- 4

**Define rubrics with explicit anchors**  
Every score point needs a concrete description
- 5

**Calibrate your LLM judge against human scores**  
Target correlation  $\geq 0.8$  before running at scale
- 6

**A/B test every meaningful change**  
One variable at a time

## Full stack in six steps

### — THE GOAL

golden cases today

# Make "is this better?" a question you answer with data

m by query type and difficulty

Before: "I think the new prompt is better, it felt more accurate"

After: "The new prompt scored 4.3/5 vs 4.1/5 on accuracy,  
passed 91% of golden cases vs 87%,  
with no change in latency or cost"

That's the difference between guessing and shipping with confidence.

ful change

# Start at Stage 1

---

10–15 golden cases, run after every commit. Add stages as your system matures.