AI Evals Cheatsheet

Find out whether your AI product works, not whether the model is smart.
Running example: a support agent that cancels orders.

biku.dev
biku.dev/blog/ai-evals-cheatsheet

Capture Log full traces → **Look** Sample and read them → **Annotate** Note what went wrong → **Count** Group failures, rank them → **Automate** Pass/fail check per failure → **Validate** Agree with humans? → **Run & repeat** CI, monitoring, guardrails

1 Why evals aren't unit tests

- **Inputs are unbounded.** Free text, not buttons and dropdowns. You can't list every case up front.
- **The model is probabilistic.** Same or reworded input, different output.
- **Failures are silent.** No stack trace, just a confident, well-written, wrong answer.
- **So evals are a loop,** not a one-off. Simulate before launch, then update from real users.

2 Benchmarks vs product evals

	MODEL BENCHMARKS	PRODUCT EVALS
Tests	General ability	Your task, data, tools, users
Examples	MMLU-Pro, SWE-bench, Arena	"Was the order actually cancelled?"
Use for	Shortlisting models	Deciding what ships

A model that ranks low on benchmarks can win on your task. Benchmarks filter; your evals decide.

3 Traces: log everything the model saw

- ☐ User input and system prompt
- ☐ Retrieved context
- ☐ Every tool call **and its result**
- ☐ Model replies, including sub-agents
- ☐ Latency, tokens, model and prompt version

Store them, and build a viewer that makes one trace readable in under a minute. If reading traces is painful, nobody will do it.

4 Which traces to read

METHOD	FINDS
Random sample	Unknown unknowns. Start here.
Clustering	Themes across similar traces
Outliers	Many turns, high latency, lots of tool calls
User signals	Thumbs down, retries, escalations

Use more signals as you learn the data, but always keep some random sampling.

5 Synthetic data, carefully

- **Generate inputs only,** never whole traces. Run them through the real system.
- **Start from scenarios:** cancel before shipping, after shipping, unclear order, angry customer.
- **For RAG,** generate a question from a document, then check retrieval finds that document.

TOO CLEAN	LIKE REAL USERS
"Cancel order #42."	"pls cancel the blue one"

6 Error analysis: annotate, group, count

Annotate in plain words. One note per trace:

```
trace 0187
cancel_order tool returned an error,
but the agent told the user
"Your order is cancelled."
```

Group notes into failure categories (AI can draft them; you refine), then **count**:

FAILURE CATEGORY	TRACES
False confirmation: tool failed	12
Missed handoff to a human	7
False confirmation: tool never called	5
Wrong order picked	4

Fix the biggest count first. Split a category that hides two causes.

7 Should this get an eval?

- ☐ You've seen it in traces, or can reproduce it
- ☐ You'll iterate on it; the fix isn't obvious
- ☐ You can say exactly what pass and fail mean
- ☐ It matters to users or the business

Obvious bug with an obvious fix? Just fix it. Evals cost time to build and maintain.

8 Pick the cheapest grader that works

GRADER	USE FOR	EXAMPLE
Code check	Objective rules. Fast, free, deterministic.	JSON valid? Cited doc exists? `cancel_order` succeeded?
Classifier / decision model	Closed questions at scale. Cheap, low latency.	"Did the user ask for a human?"
LLM judge	Judgement and nuance	"Was the reply polite, given how upset the user was?"
Human	Ground truth. Doesn't scale.	Labels to validate the others

Always binary: PASS / FAIL. Nobody can act on 3.9 vs 4.2 out of 5.

9 LLM judge prompt template

```
You are checking one failure: whether a
support agent claimed an order was cancelled
when it wasn't.
PASS if the agent only confirms a
cancellation after cancel_order succeeded,
or tells the user it failed.
FAIL if it confirms a cancellation the tool
calls don't support.
Examples: {3-5 labelled traces from the
train set}
Trace: {trace}
Give one sentence of reasoning, then PASS or
FAIL.
```

One judge per failure. Write criteria the way a domain expert would explain them to a new hire.

10 Trust the judge before you use it

Label traces yourself (AI can propose labels; you accept or fix), then split:

Train	Dev ~40%	Test, held out

- **Train:** examples inside the judge prompt.
- **Dev:** compare judge vs human. Tune the prompt until they agree.
- **Test:** check once at the end. Worse than dev means you overfit.

	HUMAN: PASS	HUMAN: FAIL
Judge: pass	✓	Missed failure
Judge: fail	False alarm	✓

11 Where evals run

CI/CD	On every prompt, model or code change, against a fixed dataset
Monitoring	On sampled production traces, to catch new failures
Guardrails	In the request path, blocking bad output. Must be fast.

12 Traps

- **Generic metrics** like "helpfulness 4.2/5". Not actionable.
- **Synthetic data that's too clean.** Real users are messy.
- **A judge you never checked** against human labels.
- **Agents rewriting prompts unreviewed.** They game the evals. Review, simplify, test on unseen cases.
- **Evals for their own sake.** Every eval should change a decision.

13 What evals buy you

- Prompt and model changes backed by evidence.
- **Move tasks to smaller, cheaper models:** set a quality bar, keep it, cut the cost.
- A clear "good enough to ship" line.