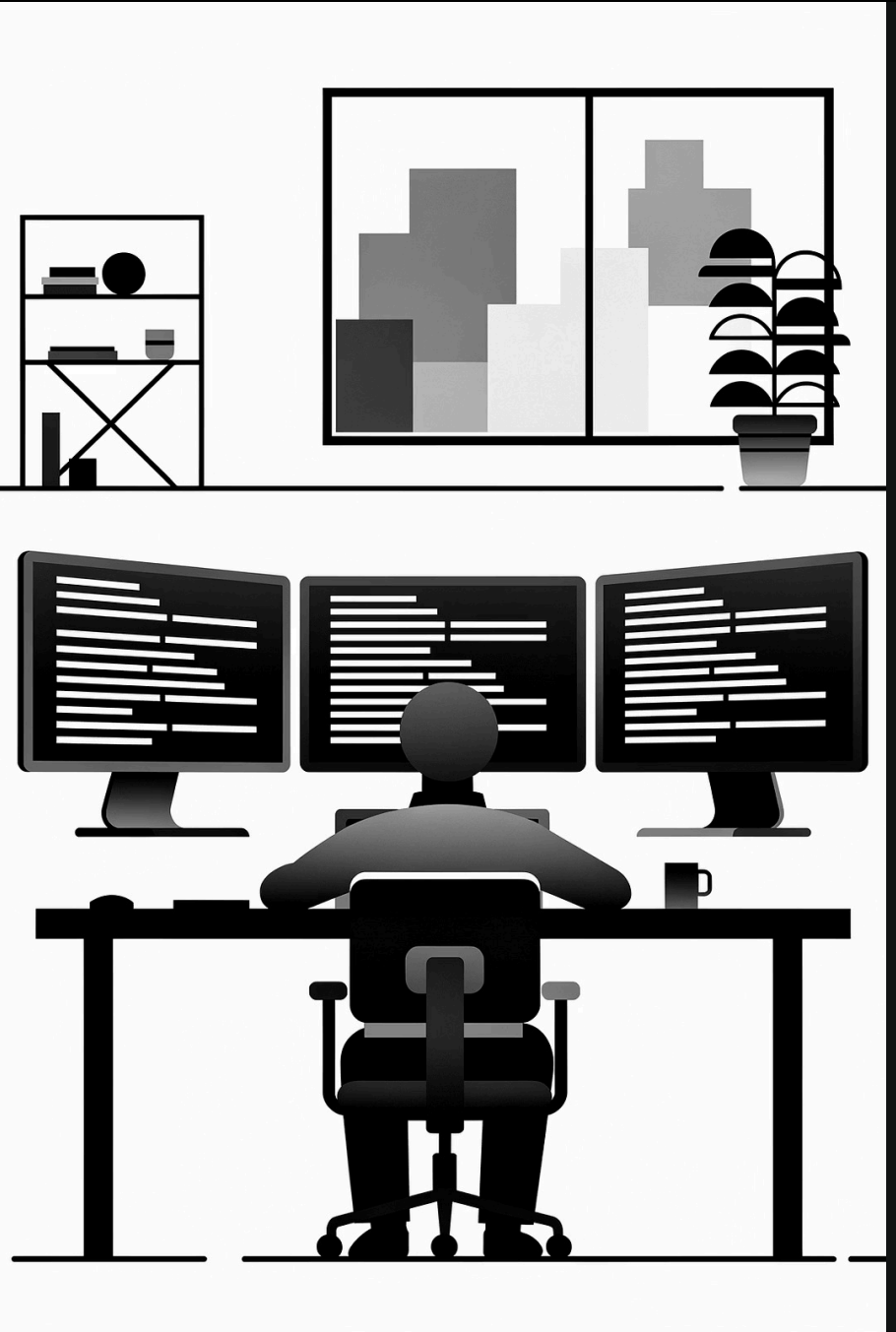


When (& How) to Start Writing Evals

A practical guide to evaluation-driven development for LLM systems

Šimon Podhajský

Head of AI @ Waypoint AI

Personal History

- Yale University — Cognitive Science
- Neuroscience → Data Science → AI
- ex-Pure Storage, SRI, Nano Energies

Waypoint AI

AI platform for escalation management for support teams

- **Chip** = AI support engineer (triage, investigation, postmortems)
- Customers: ClickHouse, Kpler, Volvo
- $3.1M pre-seed (42CAP, Dreamcraft)

Další aktivity

- Co-host of 'AI ta krajta' and 'Data Talk' podcasts
- Talks about LLMs, data, Python

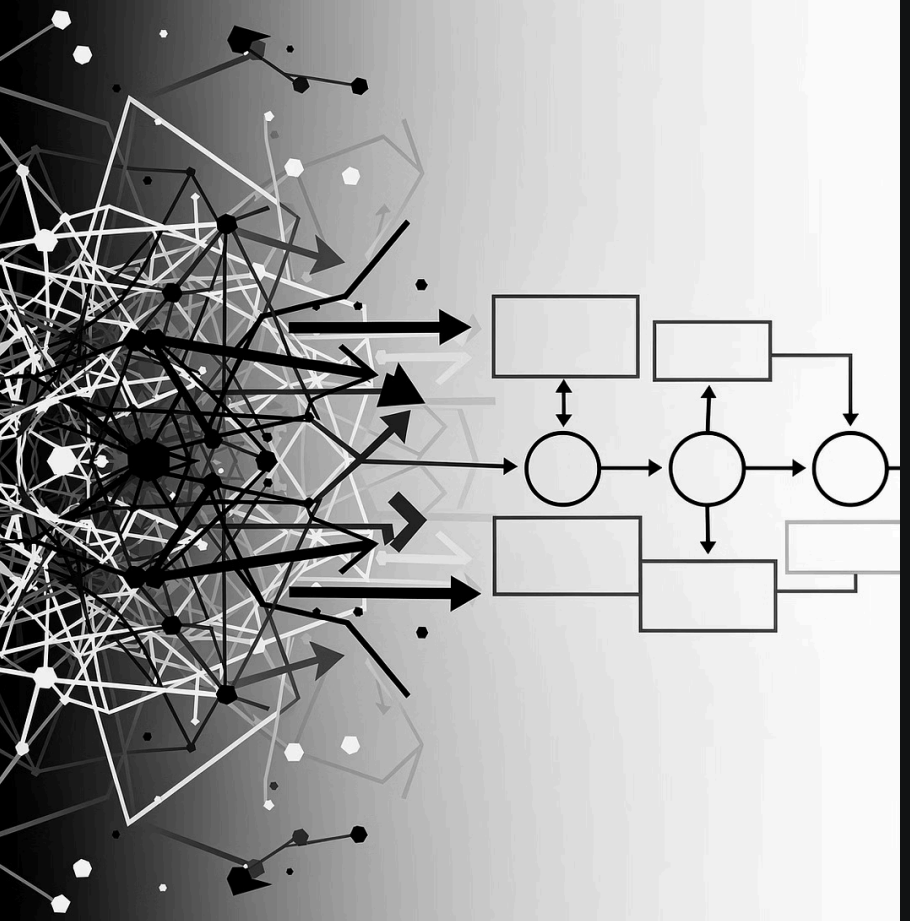
🌐 **simon.podhajsky.net** | 𝕏 **@sim_pod**

What Are Evals?

Automated quality checks for LLM outputs

Example — cover letter generator:

Word count 100-400	Code
Mentions company name	Code
Professional tone	LLM-judge
No hallucinated skills	LLM-judge



The Core Problem

Eval-Driven Development

Build evals *after* discovering patterns

Reactive, informed by real failures

Test-Driven Development

Write tests *before* implementation

Proactive, based on specifications

These are fundamentally different approaches — treating them the same leads to brittle, ineffective evaluations.

Why Not Eval-First?

Unpredictable failure modes

LLMs break in surprising ways you can't anticipate upfront

Vanity metrics trap

Optimizing for imagined benchmarks that don't reflect real user needs

Premature constraints

Narrowing your solution space before understanding the problem landscape

When to Start

The trigger:

> **"I've seen this failure before."**

Not when exploring

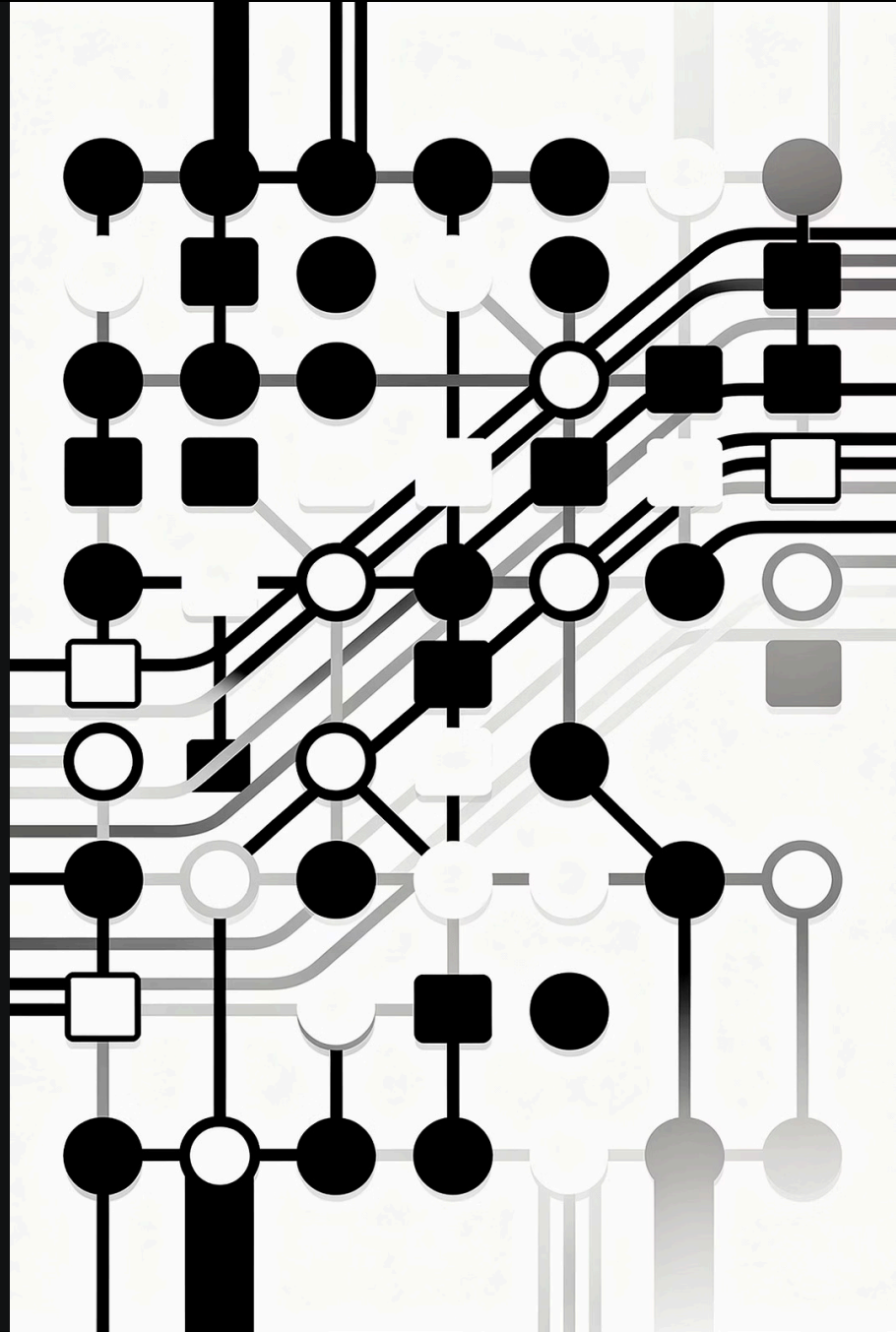
Discovery phase requires freedom

Not when guessing

Hypothetical failures mislead

When patterns repeat

Empirical evidence guides evals



Grounded Theory Approach

METHODOLOGY

Systematic method from Glaser & Strauss (1967) for discovering patterns in qualitative data

01

Open Coding

Document individual failures as they occur — raw observations without interpretation

02

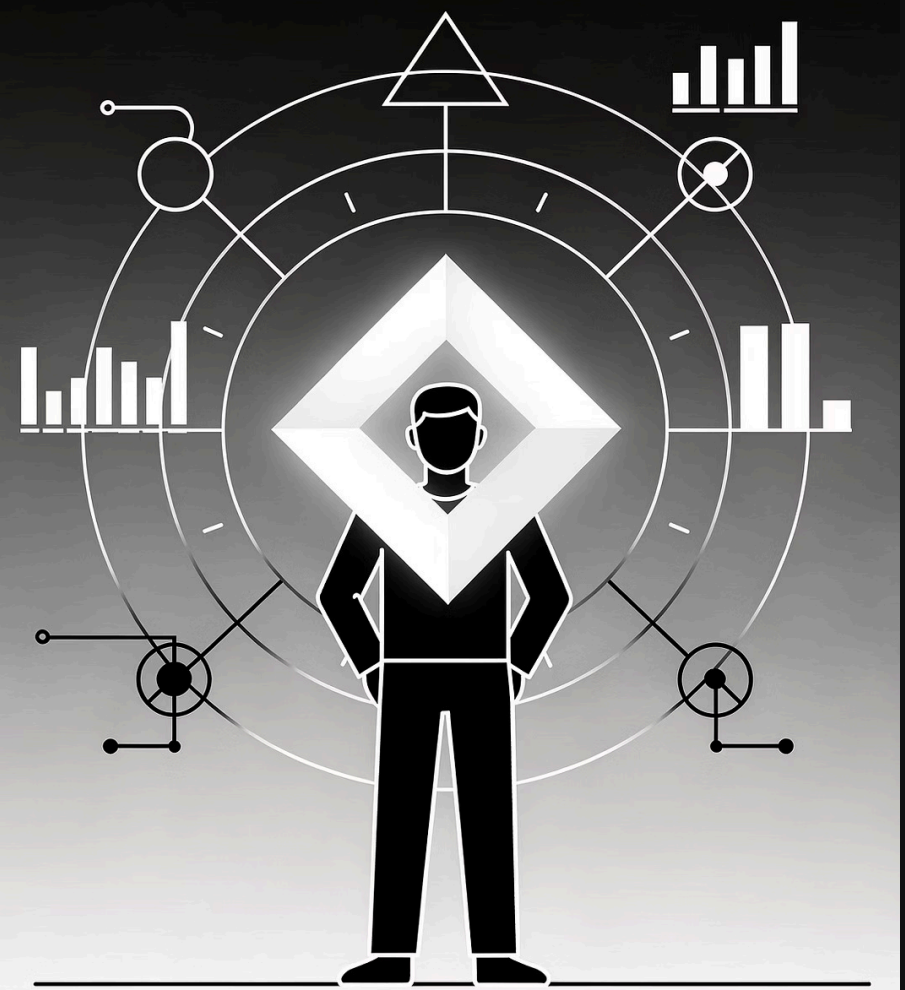
Axial Coding

Synthesize notes into 5-6 meaningful categories that capture recurring themes

03

Theoretical Saturation

Stop collecting when new examples fit existing patterns — no novel failure modes



The Benevolent Dictator

One domain expert, consistent judgment, is crucial for effective evaluation analysis.

Knows your users

Understands what "good" actually means in context, translating user needs into tangible evaluation criteria.

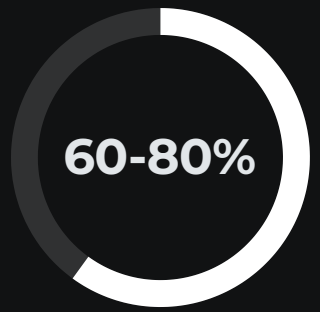
Maintains consistency

The same person judges throughout the analysis phase, ensuring coherence in pattern recognition.

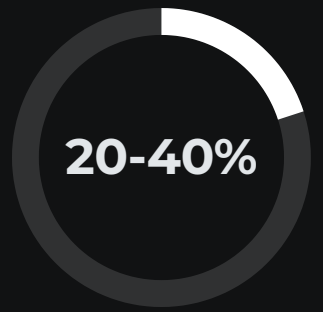
Stops at saturation

Recognizes when failure patterns repeat, avoiding wasted effort on novel but infrequent issues.

The 60-80% Rule



Error analysis
Manual review and pattern identification



Eval code
Writing actual evaluation logic

Investment sequence

1. Manual review sessions first
2. Automation after clarity emerges
3. ~30 min/week ongoing maintenance

Evals = Living PRDs

Your evaluation suite should function as evolving product requirements — codified expectations that grow with your understanding.

Level 1: Code-based

What it checks: Format, length, keywords

When to run: Fast, cheap — every code change

Example: Response contains required sections, stays under token limit

Level 2: LLM-as-judge

What it checks: Tone, relevance, helpfulness

When to run: Slower, more expensive — samples only

Example: Response maintains professional tone, addresses user intent

The Exception

> Eval-first *does* work for hard constraints

"Never mention competitors"

Brand safety requirement

"Always include disclaimer"

Legal compliance mandate

Why this works: Binary outcomes, knowable requirements, narrow scope — no exploration needed

Practical Sequencing

Phase	Focus	Evaluation Strategy
Exploration	"What's possible?"	None — observe and learn
Iteration	"What keeps breaking?"	Grounded analysis of failures
Hardening	"Prevent regressions"	Level 1 + Level 2 evals
Scaling	"Real user outcomes"	A/B tests with live traffic

The Mantra

Discover

not imagine

> **Write evaluators for errors you discover, not errors you ~~imagine~~.**

Sources & Further Reading

- Hamel Husain — LLM evaluation frameworks
- Shreya Shankar — ML testing patterns
- Glaser & Strauss (1967) — *The Discovery of Grounded Theory*