Evaluating AI causal coding – prompts, models and the metrics that lied

14 Jul 2026

Evaluating AI-assisted causal coding: prompts, models, and the metrics that lied. A day-long systematic evaluation, conducted by an AI agent under human direction, of the prompts and models behind Causal Map's AI coding.

Abstract

This working paper reports a systematic evaluation of the prompts and language models used for AI-assisted causal coding in the Causal Map app, carried out in one day by an AI agent (Claude) working under human direction, with every finding measured against real texts rather than assumed. Thirteen prompt versions and six models were tested on three contrasting texts (a disfluent interview, a formal strategy document, and a loneliness interview from the agents4qual corpus), producing roughly two thousand coded links, each judged for evidential quality by a second model against a rubric, with a 60-link human audit calibrating the judge.

Three findings stand out. First, the most consequential intervention was not technical: an apparently sensible rule banning "restatement" links (where cause and effect sound alike) was silently suppressing real causal content, because an activity that brings something about is a real claim, however weakly worded. Replacing the ban with a positive instruction, say WHAT happened and to WHOM, recovered 42% more good links from the same model at the same error rate. No metric could see this problem, because every metric counted bad links that got in and none counted good links kept out. Second, model generations split cleanly into productive coders and precise coders, and the best pipeline uses both: a productive model over-codes, then a careful model prunes, keeping about 95% of the good links while removing about 70% of the errors. The pruner must be the more careful model; a model pruning its own output leaves twice the errors. Third, and methodologically central: six times over, a metric we invented to measure quality was convincingly wrong before it was right, and seven separate silent-failure modes in the evaluation harness each made a model look worse than it was. Counting caught the harness bugs, and counting would also have shipped the worst prompt. Only reading the actual links, by hand, caught that.

The evaluation also produced a layered prompt architecture separating the minimalist coding method (see Minimalist coding for causal mapping) from its optional extensions and from case-specific briefing, with the dependencies between extensions made explicit and machine-checked.

Introduction

Causal Map's AI coding asks a language model to read a text and return causal links: one cause, one effect, one verbatim quote as evidence. The coding stance is the minimalist one described in Minimalist coding for causal mapping: we record what sources claimed, with provenance, and defer judgement about the world.

The quality of that coding depends on a prompt and a model, and until this evaluation both were chosen mostly on experience and inspection. This paper reports what happened when we measured instead. It is a sister paper to the agents4qual submission on AI-led thematic analysis, and like that paper it is also a "show your work" account: the working notes, prompt versions, harness code and judged outputs are in the `causal-map-extension` repository (`docs/plans/ai-coding-prompts/` and `docs/plans/ai-coding-prompt-experiments`), so the work can be reconstructed or continued at any point.

The headline numbers matter less than the method lessons. We set out to find the best prompt and the best model. What we mostly found was how easily an evaluation of this kind deceives its own evaluator, and what discipline it takes to notice.

Method

The harness

The evaluation harness reuses production's own machinery rather than a parallel implementation: the app's chunking (4,000 characters, snapped to sentence boundaries), its prompt builder, its response parsers, and its quote matcher. This matters because it makes normally invisible data loss visible: for instance, links whose quote cannot be located in the text are silently dropped in production, and the harness counts them.

Two coding modes were tested throughout: "holistic" (the model returns a Mermaid diagram first, which is parsed into links) and "normal" (the model returns JSON per chunk). A prompt containing a `====` separator becomes a multi-iteration prompt: the first section codes, later sections see the whole conversation and can add, fix or delete links, exactly as in production. The harness allows a different model per iteration, which is what makes cross-model pipelines testable.

The judge

Every coded link was judged by a second model (gemini-3.1-pro) against a rubric asking: is the claim supported by the quote; is it supported by the surrounding passage; is the quote verbatim; are the labels sound; is the direction right; is the failure convention applied correctly. Verdicts are good, flawed or wrong.

The judge was calibrated by a 60-link human audit, scored cold with the judge's verdicts withheld. Agreement on usability was only 63%, and the judge ran about 20 points optimistic. But the judge beat the human on the subtle house conventions, catching an inverted direction the human missed. Neither is a gold standard; judge numbers are treated as a relative signal between prompts, never as absolute quality.

The judge's rubric deliberately encodes the same coding doctrine as the prompt. It has to: a judge that scores a real activity-to-outcome link as wrong will push every prompt revision towards over-pruning. That feedback loop, prompt and judge reinforcing each other's mistake, produced the most instructive error of the evaluation (below).

The texts

Three texts of about 12,000 characters each, chosen to fail differently: a disfluent spoken interview about a policy-engagement programme; a formal strategy document about the same programme; and a loneliness interview from the agents4qual corpus, coded with a fully generic prompt carrying no case briefing at all. Conclusions drawn on one text repeatedly failed to survive the others, so every recommendation below rests on all three.

The prompt: banning restatement suppressed real content

The prompt went through thirteen versions. Most of the improvements were ordinary: a quote rule (quotes must be at least two sentences and contain cause, effect and the link between them, which fixed fragment-quotes outright); a causal-claim gate (most sentences are not causal claims; a list is not a chain; a description is not a claim; where the text gives a purpose, the purpose is the effect).

The consequential change was different in kind. Versions 4 to 6 attacked a real failure mode, links like "IPR staff set up a webpage → webpage set up", by banning restatement: if the two ends of a link sound alike, do not code it. Precision improved. Every metric approved.

The ban was wrong. The human director spotted what the metrics could not: an activity that brings something about is a real causal link, and the heart of this kind of coding. "Sue delivered coaching → Bob received coaching" is weak, but it is not false: Sue did something, and something happened to Bob. The fix is a better formulation ("→ Bob's skills in writing bids improved"), not deletion. "Rising unemployment made me lose my job" sounds tautological and is a perfectly good claim: a general state bringing about a particular event. Only genuinely empty links are dead: the identical label at both ends, or an effect that is the cause with the actor stripped out and nothing added.

Version 8 rebuilt the rule around a positive doctrine:

> A causal link is: something one thing did, or some state it was in, made something happen to something ELSE, or kept it as it was. Say WHAT happened, and to WHOM. Name the actor as the subject.

On the same model (gemini-2.5-flash), same texts, same judge, the effect was large: 105 good links became 149, a 42% gain, with the error rate unchanged. The ban had been suppressing real causal content, and nothing in the measurement system could see it, because every metric counted bad links that got in and none counted good links kept out.

There is a sting in this story. Version 8 initially appeared to make the careful model (gemini-3.1-pro) worse, and for several hours the working conclusion was that different models need different prompts. The real cause was a one-line editing bug: the new prompt still contained the old "no restatements" bullet further down, so the prompt contradicted itself. The productive model ignored the contradiction; the careful model obeyed the stricter branch. Removing that single stale line restored the careful model completely. The general lesson: when a prompt changes a rule, search the whole prompt for the old rule, because long prompts restate their rules in several places and a stale copy silently reverses the change for exactly the models that read most carefully.

The prompt architecture: method, extensions, case

That editing bug was possible because every prompt version was a single 300-line file with everything welded together: case briefing, coding method, label conventions, codebook, failure coding. A change to a core rule had to be made everywhere it was restated, and once it was not.

The prompt is now composed from layers, mirroring the structure of the method itself:

- a **generic core**: what a causal claim is, the say-what-happened doctrine, the quote rule, with deliberately generic examples;
- optional **extensions**: hierarchical `General; specific` labels, `~` opposites, despite/failure coding, causal packages, extra columns;
- an optional **codebook**, guided (prefer these labels, invent where none fits) or strict (closed list; a claim that will not fit is not coded), and independent of hierarchy, since a codebook can be flat or many levels deep, open or closed, in any combination;
- a **case briefing** (project background, abbreviations), and optionally hand-written case-specific worked examples.

The extensions are not orthogonal, and pretending otherwise produced wrong instructions. Two couplings have to be managed. First, dependence: despite coding uses the `~` opposite marker throughout, so it cannot load without opposites (the reverse is fine). Second, combination semantics: when two extensions are both on, the prompt must say how they compose. With a codebook and hierarchical labels together, the codebook entries fill the general part of each label; with a codebook alone, they are the whole label. With opposites and a codebook, `~Employment` is legal even though only `Employment` is listed. These pairwise rules live in a declarative table in the composer and are injected only when both parts are on, always attached to the later-loading layer so that no file ever references a layer that might be absent.

One negative result: a hand-written layer of case-specific worked examples, added to test whether domain-matched counterexamples help weaker models, produced no measurable gain. The mechanism is kept; the claim is not made.

Models: productive coders and precise coders

Six models were compared on identical prompts, all links judged (single runs; the noise band is about ±10 percentage points):

model	links	good%	wrong %	character
gemini-2.5-flash	250	70%	16%	productive: finds the most real content, at a cost
gemini-3.1-pro	57	81%	2%	precise: misses roughly 60% of the real content
gemini-3.5-flash	70	84%	1%	precise, at a quarter of 3.1-pro's price
gemini-3-flash-preview	154	58%	18%	ruled out: volume without quality
claude-sonnet-4-6	108	35%	29%	ruled out
claude-haiku-4-5	79	27%	36%	ruled out

(The first three rows are from the final three-text run; the last three from the earlier two-text model comparison. The Claude results were checked three ways, including a human audit, since a Gemini model was judging a rival.)

The generational pattern is consistent: newer Gemini models are markedly more conservative and more accurate. gemini-3.5-flash was the surprise, behaving like a cheap version of the flagship: on the loneliness interview it coded 23 links and every one was judged good. But its volume is a third of gemini-2.5-flash's, so precision and coverage remain different tools. Importantly, the precise models are not merely declining junk: of the links the productive model finds and the precise one does not, over half are judged good.

Pipelines: over-produce, then prune

The best results came from two-pass pipelines where a second iteration reviews the first pass's links. Three regularities:

- **Cleanup after a productive model works.** The prune keeps about 95% of the good links while removing about 70% of the wrong ones. The errors that survive a first pass are dominated by evidence failures (the quote does not show the claim), which is exactly what a review pass can check.
- **The pruner must be a different, more careful model.** A productive model pruning its own output leaves twice the

errors of a careful model pruning the same output, for the same number of good links kept. A careful coder's rare mistakes are the hard, plausible-looking ones that a pruner cannot distinguish from real links; a productive coder's mistakes are the obvious ones a careful pruner removes cleanly.
- **Padding after a conservative model fails.** Asking a precise model's second pass to add missed links buys recall by lowering the bar: error rates rise. Over-produce and prune; do not under-produce and pad.

The recommended configurations, on the composed prompt across all three texts:

use	pipeline	good links	wrong links
coverage	gemini-2.5-flash codes, gemini-3.1-pro prunes	153	12
coverage, cheaper	gemini-2.5-flash codes, gemini-3.5-flash prunes	139	9
precision	gemini-3.5-flash alone	59	1

Whether three times the good links is worth twelve times the wrong ones is a judgement about the project, not something the data settles.

The metrics lied: six times, convincingly

This is the finding we would most want a reader to take away. Six times, a measurement invented to assess quality was wrong before it was right, and each time it was convincing.

1. **Structural metrics.** An early prompt version improved every countable structural property of the map: fewer disconnected components, longer causal chains. It achieved this by inventing links, 52% of which were wrong, quoting fragments that verified perfectly and evidenced nothing. It was nearly shipped on its numbers.
2. **The soft judge.** The LLM judge ran about 20 points optimistic against a blinded human audit, agreeing with the human on only 63% of links.
3. **An invented rule, measured against itself.** One version escalated a real convention (labels must not contain causal words) into a stricter invented one (labels must not contain directional words), built a checker to enforce it, and reported a labelling crisis. The house convention deliberately puts direction in the label. The crisis was an artefact of measuring against our own invention.
4. **A hardcoded English word list** in a measurement tool, for a product that codes texts in many languages. Withdrawn on principle, and it did not work anyway.
5. **A lexical tautology detector** (word overlap between cause and effect) scored a pure restatement as acceptable, because word overlap cannot see meaning.
6. **The restatement ban itself**, the central story above: the error was not in a measurement but in the objective, and it was invisible to every metric because nothing counted what was kept out.

Alongside these, the harness itself corrupted its own results in seven distinct silent ways before being trusted: reading only the first part of multi-part model responses (truncating every response from thinking models); silently reusing an expired authentication token; output files overwriting each other; rate-limited chunks silently dropped (making a model look conservative); truncated response bodies returned with a success status; fenced JSON rejected as invalid (making one model look like it produced nothing at all); and dropped connections killing whole sweeps. Every single one made a model look worse than it was, and every one produced numbers that looked plausible.

The discipline that emerged, stated as rules: verify the harness before believing any result (check the failure counters, not just the outputs); hand-read a real sample before shipping any conclusion; count what is kept out, not only what gets in; treat any single-run difference under ten points as noise, whichever way it flatters; and when a measurement and a human disagree, suspect the measurement first, but check the human too.

Limitations and open questions

Every number here is a single run; the noise band is wide, and several near-ties (including whether the layered prompt is marginally weaker than its monolithic ancestor) are undecidable at this resolution. The judge is a model with known softness, calibrated once. The texts are three; the languages are one. Nothing yet measures under-coding directly, that is, taking the claims a prompt declines and checking how many were real; this blind spot enabled the worst error above and remains open. And the most productive coding model tested is scheduled for retirement with no tested successor matching its recall.

Reflection

The substantive outputs of this day were a doctrine, an architecture and a pipeline. But the transferable finding is about evaluation itself. An AI agent measuring AI coding builds its own instruments, and those instruments fail silently and flatteringly. The failures were not exotic: stale tokens, truncated responses, optimistic judges, objectives that optimise what is countable. What caught them, each time, was either a human reading the actual output and trusting their unease, or a deliberate habit of auditing the measurement before believing it. The counting was indispensable and the counting was insufficient. That is not a paradox; it is a division of labour.

Provenance

Working notes, all thirteen prompt versions, the composer, the harness, the judge and the judged link sets are in the `causal-map-extension` repository under `docs/plans/ai-coding-prompts/` and `scripts/`. The evaluation was directed by Steve Powell and conducted by Claude (Opus 4.5/4.8) on 2026-07-14. This paper was drafted by the AI and edited by the authors.