Parametric Divergence Mapping: Deriving Evaluation Criteria from Structured Disagreement Between Frontier Models

Siddharth Nigam
Independent Researcher

April 2026

Abstract

Evaluation rubrics for LLM-assisted code artifacts are often written manually, which makes it difficult to tell whether an omitted rule reflects true irrelevance or simply an untested edge case. This paper studies that problem with **Parametric Divergence Mapping (PDM)**, an alternating two-model protocol in which Claude and GPT/Codex edit the same JavaScript measurement script under one narrowly scoped parameter at a time. Each turn must either land one qualifying structural change or return `NULL`; bilateral `NULL/NULL` is the convergence signal. The experiment covered four parameters over 77 total iterations: Measurement Accuracy (P1), Edge Case Handling (P2), Coverage Completeness (P3), and Output Clarity (P4). Three parameters converged: P2 at iteration 18, P3 at iteration 14, and P4 at iteration 10; P1 hit its hard cap at 35 without convergence. The logs also show a repeatable behavioral asymmetry under this prompt template: Claude returned `NULL` 24 times across the full run, whereas GPT/Codex returned `NULL` 3 times, and many Claude `NULL`s were subsequently overturned by Codex structural finds.

The experiment produced three contributions. First, a reproducible procedure for deriving evaluation criteria from constrained model disagreement on a diffable artifact — distinct from multi-agent debate or single-model self-refinement in that it serializes independent model turns across repository commits and treats convergence or persistent overturn as rubric evidence. Second, a meta-methodological refinement that emerged from P4: once a `NULL` is overturned, subsequent `NULL`s are credible only if they explicitly name the failure surface the prior `NULL` missed and probe it directly. This "`NULL` quality" audit is, to our knowledge, novel to cross-model iterative evaluation. Third, two concrete artifacts: a production-ready multi-signal measurement script and a tightened v2 methodology built around pre-registered scopes, visible and hidden fixtures, and explicit acceptance rules.

PDM's contribution is methodological, not metaphysical. It does not produce a ground-truth rubric; it produces a reproducible rubric with documented provenance — every entry traceable to the iteration that added it and the iterations that declined to modify it, *conditional on the researcher-chosen scope*. That distinction is the methodology's honest contribution and its primary limitation.

Keywords: LLM benchmarking, cross-model evaluation, machine-readability, agent experience optimization, rubric derivation, methodological convergence.

1 Introduction

The motivating application in this repository is Syphon's AXO (Agent Experience Optimization) scoring pipeline, which needed empirically grounded dimensions for measuring how well a page supports agent execution rather than a hand-written checklist. That application is concrete, but the underlying methodological problem is broader. Any evaluation rubric for LLM-assisted code artifacts, extraction pipelines, or readiness scores faces the same long-tail difficulty: if a scoring

dimension is missing, the artifact can still emit clean, stable output while being wrong in exactly the cases a downstream consumer cares about.

The growing practice of LLM-driven web crawling, retrieval-augmented generation, and tool-use agents has renewed interest in a question search engines addressed pragmatically twenty years ago: what does it mean for a web page to be machine-readable? Search-engine optimization (SEO) answered this with a mix of heuristic ranking features and observable crawl behavior, but the answer was never codified as a public scoring rubric. With agent-driven consumption replacing keyword-driven retrieval, the same codification gap reappears for AXO.

Ad hoc human specification is poorly matched to that setting. A researcher can specify obvious dimensions up front, but hand-written rubrics do not reliably expose which distinctions survive contact with actual parser behavior, transport failures, schema consumers, and fixture diversity. The P1 trajectory in this repository demonstrates the failure mode directly: a broad "measurement accuracy" objective admitted a long sequence of still-plausible fixes without any shared stopping rule, and the loop reached 35 iterations without bilateral closure.

Cross-model iteration is a plausible alternative because disagreement is informative. When two frontier models receive the same active parameter, the same current script, and the same requirement to either land one qualifying change or return `NULL`, several useful signals become observable. Repeated agreement on `NULL` suggests that the remaining move space violates the current acceptance criteria. Repeated disagreement isolates surfaces where the parameter definition is still under-specified or where one model family is systematically stricter than the other under the given prompt framing. The experiment operationalized that idea by alternating Claude on odd iterations and GPT/Codex on even iterations, storing every artifact and rationale as a changelog in the repository.

The paper's contribution is not an AXO rubric per se. The core contribution is a reproducible procedure for deriving evaluation criteria from structured convergence and divergence on a diffable artifact. AXO is the application case; Parametric Divergence Mapping is the method. A secondary contribution emerged mid-experiment: the `NULL`-quality audit refinement described in Section 6.4. This paper does not claim PDM produces a ground-truth rubric; it claims PDM produces a rubric whose entries each have documented provenance, *conditional on the researcher-chosen scope* — a weaker but falsifiable claim that the remainder of the paper works out carefully.

2 Related Work

The closest methodological precedent for PDM lies outside the LLM literature entirely, so the related work is organized from nearest to furthest.

Paired-annotator corpus labeling with blind adjudication. The tightest conceptual analogue is the paired-annotator pattern in computational linguistics [1]: two annotators label the same items independently, agreement counts, and disagreement is adjudicated. PDM generalizes this from labels to artifact modifications and from human annotators to models. The structural parallels are direct — independent turns, documented rationale, explicit adjudication of disagreement — but two differences matter. First, corpus linguistics treats the label space as fixed and disagreement as noise to resolve. PDM treats *scope* as fixed but the candidate space as open, and treats disagreement as either a `NULL`-breakage signal (one side was wrong) or a boundary signal (the scope does not admit convergence here). Second, corpus linguistics has a human-interpretable kappa or alpha statistic; PDM substitutes bilateral `NULL/NULL` as the stopping criterion, which is

weaker but falsifiable in a different way (the next iteration can overturn it).

Multi-agent debate and consensus. Debate-style setups such as [4] use multiple agents or model instances to improve the quality of an answer through contemporaneous exchange. PDM does not ask two agents to jointly solve one problem instance. Instead, it serializes model interaction across repository commits and treats convergence or persistent overturn as evidence about the rubric for a code artifact. The shared intuition is that disagreement is useful rather than purely noisy; the structural difference is that PDM's units of disagreement are persistent artifact states, not conversational turns.

Iterative self-improvement. Self-Refine [6] and Reflexion [7] improve outputs through repeated critique and revision. The main difference is again the object of optimization and the role of independence. Self-Refine and Reflexion are principally single-model refinement procedures. PDM deliberately alternates model families, stores each turn as a versioned artifact, and treats bilateral `NULL` as the stopping condition. The experiment therefore uses iteration not only to improve a script, but to surface which changes survive adversarially independent re-evaluation.

LLM-as-judge. A large recent literature, especially [10], studies when strong models can approximate human preferences, and bias analyses such as [9] show that evaluator models themselves introduce systematic distortions. PDM is adjacent but distinct. It does not use one model to score another against a fixed rubric. Instead, it derives the rubric dimensions themselves from iterated model behavior under constrained acceptance rules. This shifts the methodological question from "Can a model judge well?" to "Can disagreement and convergence between models reveal which rubric criteria are stable enough to keep?"

Constitutional AI. [2] uses a principles list to guide model behavior and iteratively refines those principles. PDM differs in that its principles list (the scope document) is pre-registered and frozen before iteration begins; revision happens only through pre-committed mid-loop review, a mechanism not invoked in this experiment.

Broad benchmark frameworks. HELM [5] and BIG-bench [8] evaluate models across many pre-specified tasks, scenarios, and metrics. PDM is narrower and more artifact-centric. It does not benchmark a model family across a task suite. It derives parameter-specific evaluation dimensions for one concrete code artifact whose outputs can be diffed, replayed, and checked against visible and hidden fixtures. PDM is a rubric-construction methodology rather than a benchmark.

3 Method: Parametric Divergence Mapping

The method evolved substantially during the experiment. Table 1 summarizes the evolution from the v1 setup used in P1 to the v2 setup used in P3 and P4; the subsections that follow expand its rows.

The progression from P2 to P4 shows a clear tightening trend: reproducible red/green fixtures in P2, scope plus visible/hidden fixture discipline in P3, and explicit consumer-failure pairs plus value-preservation rules in P4. PDM therefore matured during the experiment. The later parameters are not merely additional trials; they are trials under a substantially better-controlled protocol.

Table 1: Design pressures exposed by P1 and the corrections implemented in v2. The v2 protocol was not pre-planned; it was forced by P1's failure modes.

Design pressure exposed by P1	**v2 correction**	**Evidence**
Broad parameter definition without a mechanical stopping rule	P2 narrows qualification to crash prevention or fabrication prevention and disqualifies wording-only changes	`PROMPT_TEMPLATE.md`
No authoritative falsifiability surface	P3 requires visible-fixture diversity tables and precomputed ground truth	`PROMPT_TEMPLATE_P3.md`, `P3_FIXTURES.md`
Scope relitigation across iterations	P3 and P4 introduce binding scope documents with INCLUDE/EXCLUDE lists and hard caps	`P3_SCOPE.md`, `P4_SCOPE.md`
No explicit overfit check	P3 adds hidden holdouts and requires post-change holdout comparison	`P3_SCOPE.md`, `P3_FIXTURES.md`
Conversational state burden across turns	The master Codex prompt externalizes iteration state to repo files and instructs the model to infer the active parameter and iteration from the repository rather than from conversational memory	`prompts/MASTER_CODEX_PROMPT.md`

3.1 Protocol

PDM uses a simple alternating protocol. A single script is held fixed as the current artifact. One parameter is active at a time. Claude acts on odd-numbered iterations, GPT/Codex on even-numbered iterations, and each turn must do exactly one of two things:

1. land one qualifying structural change under the active parameter, or
2. return `NULL` and copy the current script unchanged.

Figure 1 shows the decision flow for a single iteration. This structure was present from the start of the experiment and remained stable across parameters. The convergence signal is bilateral `NULL`: two consecutive `NULL`s from opposite models on the same script, or an unchanged script accompanied by explicit independent validation that no qualifying move remains. P2 converged in the standard consecutive way at iterations 17–18, while P3 and P4 show the stronger "independent validation on unchanged script" form: Claude returned `NULL`, Codex re-audited the unchanged artifact, and also returned `NULL`.

3.2 Parameter Scoping

The repository shows a clear evolution from loose to pre-registered scoping. P1 ran under the original experiment plan, which described the parameter broadly but did not yet provide a binding include/exclude list or an authoritative fixture suite. By P3 and P4, scope was explicitly pre-registered in standalone documents with INCLUDE and EXCLUDE sections, formal qualification criteria, convergence rules, and hard caps.

The scoping transition matters methodologically because it prevents relitigation. `P3_SCOPE.md` locks P1 measurement logic and P2 transport logic, bans browser-dependent or constant-valued signals, and requires every new signal to be measurable, meaningful, falsifiable, non-redundant, and additive. `P4_SCOPE.md` narrows the move space even further to schema consistency, type normalization, null-handling explicitness, precision normalization, and rare field renames, explicitly banning additive changes and semantic changes to signals. The result is that later `NULL`s are interpretable as closure

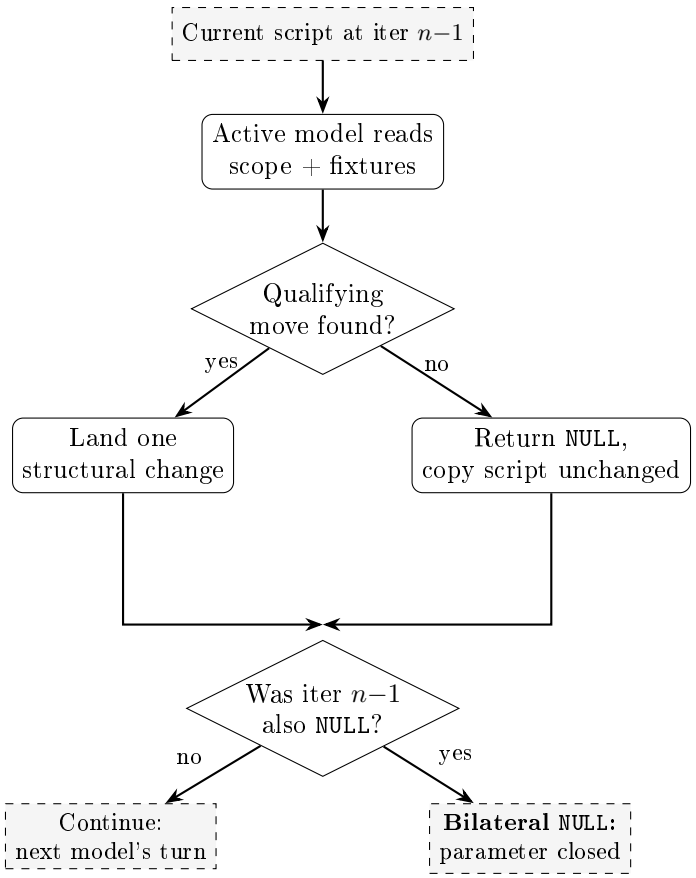

Figure 1: PDM protocol for a single iteration. The bilateral `NULL/NULL` gate is the only convergence condition; every non-`NULL` restarts the clock. Scope documents, fixture definitions, and prior iterations are available to the active model; the partner model's reasoning for the immediately prior turn is visible only through the changelog it wrote.

under a known search space rather than fatigue under an undefined one.

3.3 Acceptance Criteria

By the end of the experiment, the acceptance gate had been distilled into five concrete rules, stated explicitly in the P4 materials:

1. **Reproducible consumer failure.** A non-`NULL` move must name a concrete consumer access pattern that fails before the change and succeeds after it.
2. **Value preservation.** The move must preserve all field values except the representation change it explicitly introduces.
3. **Cross-path coverage.** The fix must apply across every code path that emits the affected field.
4. **Non-bikeshed motivation.** Renames or schema changes must address a real ambiguity or failure, not stylistic preference.
5. **Idempotence.** Reapplying the move must be equivalent to applying it once.

These criteria are explicit in both the reusable template and the pre-registered scope for P4. P2

and P3 implement narrower parameter-specific variants of the same idea: P2 admits only crash prevention or fabricated-measurement prevention with red/green fixture evidence, and P3 admits only additive new signals with falsifiability tables and non-redundancy against the existing schema.

3.4 Fixture Architecture

The v2 framework separates fixtures into two sets. Visible fixtures are shown to both models and define the immediate falsifiability surface. Hidden holdouts are never shown to either model and are used to detect overfitting. For P3, the researcher also computed authoritative ground-truth values in advance, so proposals could be checked against known signal behavior rather than free-form plausibility.

The architecture contains five visible fixtures and two hidden holdouts. The visible set covers product, news, SPA shell, semantic blog, and div-soup landing-page cases. The hidden set covers local business schema and a hybrid SSR-plus-JS page. This design makes two things testable: whether a proposed signal varies at all on the visible set, and whether its apparent novelty survives out-of-distribution examples.

3.5 NULL Semantics — With a Worked Example

The word `NULL` in this protocol is load-bearing. It is not "I could not think of a change" (which would reward laziness) and not "I do not like any candidate" (which would reward taste). It is a claim that no candidate satisfies all qualification criteria given the current state. A `NULL` is falsifiable: the partner model on the next iteration either produces a qualifying change (overturning the `NULL`) or independently returns `NULL` (ratifying it). The partner's response is the adjudicator.

The cleanest worked example in the repository is the P4 iteration 5–6 pair, which is worth walking through in full because it demonstrates both failure modes of the protocol at once.

At iteration 5, Claude returned `NULL` on Parameter 4 (Output Clarity). Its rationale audited cross-fixture leaf types using JavaScript's `typeof` operator and reported "zero drift — every leaf has the same type across all 5 visibles on the success path." Under Claude's audit, the parameter was closed.

At iteration 6, Codex overturned that `NULL`. The overturn identified a failure Claude's audit could not see: `typeof` in JavaScript returns `'number'` for both `2` and `2.1`, so an integer-valued float field is invisible to Claude's surface. But Python's `json.loads` treats `2` as `int` and `2.00` as `float`, so a Python consumer accessing designated float fields `weighted_avg_depth` and `hook_density` got `int` on some fixtures and `float` on others. Codex's fix — a fixed-width float serializer — eliminated the drift at the JSON byte level, which is the surface the consumer actually sees.

Two epistemically distinct outcomes are on display here. Claude's `NULL` was *wrong*: the audit surface did not match the consumer's access surface. Later, at iteration 10, Codex returned `NULL` after re-auditing the iteration 9 artifact, and both models ratified the same state. The log preserves the distinction. A `NULL` that is broken records a methodological finding (use the right audit surface); a `NULL` that is ratified records a scope-internal convergence finding. Section 6.4 analyzes the consequence that forced itself on the protocol: once a `NULL` is broken, the next `NULL` must explicitly name the failure mode it missed and probe that new surface directly.

4 Experimental Setup

4.1 Task

The artifact under refinement was a Node.js script that began as a DOM-depth measurer and ended as a multi-signal AXO measurement tool. The baseline `original-script.js` fetched a URL server-side and emitted only `maximum_depth`, `average_depth`, `element_count`, `timestamp`, and `error`. Through P3 and P4, the script became a 17-key output object with eleven nested signal groups plus the original depth fields.

4.2 Models

Claude acted on odd iterations and GPT/Codex on even iterations throughout the experiment. Later protocol materials standardized the Codex side further: the master Codex prompt instructs Codex to determine the current state from repository files, forbids reading hidden holdouts, and treats the repository rather than the conversation as the source of truth for iteration state. Specific model versions are those offered during April 2026 runs. Section 9 discusses the limits of retrospective replication that follow.

4.3 Fixtures

The P3/P4 fixture suite contained 5 visible fixtures and 2 hidden holdouts. Ground-truth values for the visible suite were researcher-computed and documented in `P3_FIXTURES.md` before P3 iteration 1. The visible set was intentionally constructed to vary on the seeded structural axes, while the hidden set tested whether the accepted signals generalized beyond the fixture patterns the models could see.

4.4 Per-parameter Caps and Scope

Table 2: Per-parameter goals, caps, scope controls, and final artifacts.

P	Goal	Cap	Scope controls	Final artifact
P1	Measurement Accuracy	browser-faithful DOM depth measurement (cap 35)	original v1 framing only	`P-1/iteration-35.js`
P2	Edge Case Handling	clean errors instead of crashes or fabricated measurements (cap 20)	v2 prompt with mandatory reproducible fixture rule	`P-2/iteration-18.js`
P3	Coverage Completeness	additive new AXO-relevant signals (cap 20)	pre-registered scope, visible/hidden fixtures, ground truth	`P-3/iteration-14.js`
P4	Output Clarity	schema consumability without value changes (cap 10)	pre-registered scope, move categories A–E, consumer-failure pairs	`P-4/iteration-10.js`

The P3 and P4 caps were justified explicitly in the scope documents, which cite P1's scope drift as the motivating failure case for tighter control.

5 Results

5.1 P1 Measurement Accuracy

P1 did not converge. The canonical convergence record reports 35 iterations, 14 Claude `NULL`s, 0 GPT `NULL`s, 4 Claude structural changes, and 17 GPT structural changes. The important result is not only that the cap was reached, but how it was reached. GPT/Codex repeatedly found browser-equivalence defects after Claude had declared closure: template opacity at iteration 4, implicit table wrappers at iteration 6, full document-wrapper synthesis at iteration 8, and `parseNoneClosedTags` at iteration 32.

The same convergence record groups P1 fixes into parser configuration, document synthesis, table wrapper synthesis, traversal transparency, and traversal opacity, and leaves three unresolved deferrals: foster parenting, adoption-agency behavior, and the select content model. Those deferrals all require HTML5 tree-construction state unavailable from the parser output, which means P1 was effectively trying to solve browser-faithful DOM reconstruction without an oracle, without fixtures, and without a tight boundary around what counts as "done."

5.2 P2 Edge Case Handling

P2 converged because the failure surface collapsed around one observable bad output: the fabricated synthesized three-element document, reported as `maximum_depth: 1`, `average_depth: 0.7`, `element_count: 3`, `error: null` on inputs that should not have been measured.

The convergence path is best understood as closure of five causal axes of the same symptom:

Table 3: P2 convergence as closure of five causal axes of a single failure symptom.

Failure axis	Representative iterations	Evidence
Transport / HTTP handling	P2-1, P2-2, P2-4, P2-8	stack overflow prevention, no-content rejection, partial-content rejection, catch-chain repair
Auto-parse under claimed HTML	P2-10, P2-11, P2-12	falsy coercion fix and upstream non-string rejection
Declared charset mismatch	P2-14	UTF-16LE mojibake rejected upstream
Pre-parse body shape	P2-15	empty, whitespace-only, and plain-text bodies under `text/html` rejected
Post-parse zero-element markup	P2-16	`<`, comment-only, CDATA, and similar zero-element bodies rejected with doctype exemption

The last two iterations are especially informative. Claude's iteration 17 `NULL` argues that all five axes are now closed and classifies remaining candidates as not-a-fabrication, speculative, or wording-only. Codex iteration 18 then independently rechecks doctype shells, empty HTML shells, SVG/XML with real elements, malformed-doctype junk, BOM-prefixed HTML, and transport failures, and also returns `NULL`. This is a clean example of bilateral closure under a tightened fixture-based rule.

5.3 P3 Coverage Completeness

P3 converged at iteration 14 after accepting 11 new signal groups: `js_dependency`, `structured_data`, `heading_hierarchy`, `semantic_html`, `machine_readable_blocks`, `answer_first`, `link_profile`,

`selector_affordance`, `content_depth`, `text_container_profile`, and `reference_context`. Each signal was required to add a new schema field, vary across the visible fixtures, and clear a non-redundancy test against existing signals.

P3 closes with a strong example of independent validation. Claude iteration 13 rejects `temporal_metadata` as semantically real but non-redundant only in theory, not on the current fixture set. Codex iteration 14 then also returns `NULL`, explicitly stating that "Claude's iter-13 analysis was correct" and rejecting `temporal_metadata`, `section_heading_coverage`, and `article_header_profile` under the same non-redundancy criterion. The experiment log records this as bilateral convergence at 14 with the coverage-completeness space saturated under the current fixture suite.

Section 6.7 examines a pattern that is easy to miss in this summary: more than half the accepted P3 signals carry weak-boolean-redundancy flags. That pattern is the single most important caveat on the P3 result.

5.4 P4 Output Clarity

P4 is the cleanest illustration of the mature method. It produced six non-`NULL` moves and four `NULL` attempts, with bilateral convergence reached at the hard cap of 10 iterations.

Table 4: P4 iteration summary. Categories A–E are schema-consistency (A), type normalization (B), null-handling explicit (C), precision normalization (D), and field rename (E).

Iter	Model	Outcome	Category / issue
P4-1	Claude	non-`NULL`	A: catch-block key-set drift
P4-2	GPT/Codex	non-`NULL`	C/B: `heading_hierarchy` null-object
P4-3	Claude	non-`NULL`	D: `average_depth` precision normalization
P4-4	GPT/Codex	non-`NULL`	B: all remaining nested groups object-always
P4-5	Claude	`NULL`	later broken by iter-6
P4-6	GPT/Codex	non-`NULL`	B: Python-visible int/float normalization via fixed-width float serializer
P4-7	Claude	`NULL`	later broken by iter-8
P4-8	GPT/Codex	non-`NULL`	B: serializer collision on sentinel-shaped `url` input
P4-9	Claude	`NULL`	defensible `NULL` after explicit adversarial audit
P4-10	GPT/Codex	`NULL`	bilateral convergence

Two methodological failure modes are documented in the changelogs themselves. First, Claude's iteration-5 `NULL` used JavaScript runtime typing and therefore missed the fact that Python `json.loads` distinguishes `2` from `2.00`; Codex iteration 6 broke that `NULL` with a real consumer failure on `weighted_avg_depth` and `hook_density`. Second, Claude's iteration-7 `NULL` corrected the audit surface to byte-level JSON inspection but still limited the audit to fixture-path inputs; Codex iteration 8 then showed that the serializer itself could corrupt the top-level `url` field on adversarial sentinel-shaped input. By iteration 10, both models agreed that the remaining space had collapsed from ambiguity to preference. Codex's convergence rationale states: "What remains is not ambiguity in the emitted JSON. It is preference. That is exactly where P4 is supposed to stop."

6 Behavioral Findings

6.1 Observed NULL Asymmetry Under This Prompt Template

The strongest behavioral result is the `NULL` asymmetry. Counting directly from the canonical iteration table yields 24 Claude `NULL`s and 3 GPT/Codex `NULL`s across the four parameters.

Table 5: Observed NULL counts by parameter and model under the prompt template described in this paper. Model-family attributions are deliberately weak; see discussion below.

Parameter	**Claude** `NULL`s	**GPT/Codex** `NULL`s
P1 Measurement Accuracy	14	0
P2 Edge Case Handling	5	1
P3 Coverage Completeness	2	1
P4 Output Clarity	3	1

The logs show a consistent directional difference in behavior. Claude tends to ask whether the remaining move is truly within scope and sufficiently structural to justify continuation. Codex tends to ask whether any definitional or consumer-visible defect still survives. The repository supports this as behavior, not mechanism: P1's convergence record explicitly summarizes Claude as optimizing "scope control and practical impact thresholds" and GPT as optimizing "definitional correctness regardless of implementation cost." P4's broken `NULL`s show the same asymmetry under a much tighter parameter: Claude stopped once the pre-registered issue list looked closed, whereas Codex continued to search for consumer-visible type corruption and serializer edge cases.

Three possible causes, not cleanly separable on one model pair. It is tempting to read this as a statement about Claude and Codex as model families. The evidence on $n = 1$ pair does not support that strong a claim. At least three mechanisms could produce this pattern, and the current data cannot cleanly separate them:

1. **Genuine model-family difference in conservatism.** Claude and Codex may differ in how much scope-tightness they apply to ambiguous structural judgments. If this is the dominant cause, the asymmetry should replicate with the same model families under different prompt templates.

2. **Prompt-template framing effects.** The `NULL` condition was defined through iteration in prompt materials that one researcher wrote. The phrasing that made `NULL` feel like a legitimate, falsifiable answer (rather than an admission of failure) may land differently with different models. If this is the dominant cause, the asymmetry should attenuate or flip under a template rewritten by someone else.

3. **Position effects.** Claude occupied odd iterations, which often meant following a non-`NULL` structural move where the obvious next change had already been made. Codex occupied even iterations, which often meant following a `NULL` with fresh surface to probe. This confound is structural, not cosmetic: it could generate the entire observed asymmetry even with identical model behavior.

The section title and prior framing have therefore been kept deliberately weak. What this experiment establishes is *a consistent NULL asymmetry under the specific prompt template and iteration schedule used.* Generalizing from that observation to a statement about model families requires the cross-pair replication described in Section 11.

6.2 The Claude-NULL / Codex-Structural-Find Pattern

This pattern recurs across parameters:

The pattern does not imply that Claude was simply "too conservative" in every case. P2 iteration 6

Table 6: Recurring pattern of Claude NULL followed by Codex structural find. Model attributions carry the same caveat as Section 6.1.

Parameter	Claude `NULL`	Codex overturn
P1	iter 3	iter 4 finds template opacity
P1	iter 5	iter 6 finds implicit table wrappers
P1	iter 7	iter 8 synthesizes HTML/HEAD/BODY wrappers
P1	iter 31	iter 32 enables `parseNoneClosedTags`
P2	iter 5	iter 6 improves descriptive HTTP status text within the parameter definition
P2	iter 7	iter 8 fixes catch-branch ordering bug
P2	iter 13	iter 14 finds charset-mismatch fabrication
P4	iter 5	iter 6 finds Python-visible int/float drift
P4	iter 7	iter 8 finds serializer collision on adversarial input

is a good counterexample: the change did not alter success-versus-failure control flow, but it still fell within the stated parameter definition because P2 still allowed "descriptive message" quality at that point. The important empirical point is narrower: the model in the second position (Codex, under this schedule) was more willing to keep searching after the model in the first position (Claude) judged the surface saturated.

6.3 Convergent Follow-Through

Not all cross-model interaction took the form of overturning a prior `NULL`. Several of the most productive sequences involved the second model extending the exact failure class just exposed by the first.

In P2, Codex iteration 10 found falsy-coercion fabrication on headerless bodies, and Claude iteration 11 generalized that same auto-parse failure class to claimed-HTML responses. Codex iteration 14 then found the encoding axis of the fabricated-three-element bug, and Claude iteration 15 extended the same class to empty, whitespace-only, and plain-text `text/html` bodies. Codex iteration 16 extended that one layer deeper again, from pre-parse body shape to post-parse zero-element markup.

At convergence, the same pattern appeared in a stricter form. Claude iteration 13 in P3 returned `NULL` with a detailed rejection of `temporal_metadata`. Codex iteration 14 did not merely fail to find a new move; it explicitly validated the same rejection and adopted the same non-redundancy conclusion.

Independent arrival at the same `NULL` across models is the protocol's strongest convergence signal. It is epistemically distinct from a ratification where the second model simply returns `NULL`: in independent arrival, the two models' reasoning chains, constructed without access to each other, converge on the same rejection criterion. Two of the three bilateral convergences in this experiment (P3 iteration 14, P4 iteration 10) closed this way.

6.4 Meta-Methodology Emergent from P4: The NULL Quality Audit

P4 produced the clearest methodological lesson in the repository: *once a `NULL` is broken, the next `NULL` has to name the failure mode it missed and then audit that surface directly.* This is not a decoration on the protocol — it is a concrete refinement of what counts as a credible `NULL`, and it is, to our knowledge, novel to cross-model iterative evaluation methodologies.

Claude's iteration-7 changelog makes the rule explicit. It begins by correcting iteration 5's methodology, stating that the earlier `NULL` used the wrong surface because JavaScript `typeof` collapses integer and float values into the same runtime type. Claude's iteration-9 changelog repeats the pattern at a higher bar: it opens by distinguishing itself from iterations 5 and 7 and then documents eight concrete audit surfaces, including adversarial serializer probes that the earlier `NULL`s had not run.

Stated formally: a credible `NULL` at iteration n must (a) enumerate the failure modes that broke the most recent `NULL` in the same parameter, and (b) demonstrate that it has probed each one on the current artifact. `NULL`s that do not satisfy this are structurally the same as the `NULL`s that were already overturned. Under the unaugmented protocol, this was implicit; P4's broken-`NULL` sequence forced it to become explicit.

The generalization beyond this experiment is straightforward. Any iterative convergence protocol where the stopping criterion depends on one side declaring closure is vulnerable to bad-closure declarations that pattern-match to good ones. The `NULL`-quality audit is a local adversarial check: the closure must survive the surface that the last closure-claim failed to survive. This is cheap to enforce, visible in changelogs after the fact, and directly falsifiable.

6.5 NULL That Does Not Hold

P4 iterations 5 and 7 are the clearest examples of methodologically flawed `NULL`s. They have already been walked through in Section 3.5 and summarized above; the diagnostic point is worth stating separately.

Iteration 5 concluded that no live P4 move remained because it diffed success-path leaf runtime types in JavaScript and found no drift. Codex iteration 6 showed that this missed the relevant consumer surface entirely: Python consumers treat whole-number JSON tokens as `int`, so designated float fields still drifted across fixtures.

Iteration 7 corrected the type-audit surface but still dismissed serializer fragility as hypothetical refactoring rather than a live consumer failure. Codex iteration 8 then converted that fragility into a reproducing counterexample by passing `__AXO_FLOAT__2.00__` as the URL and showing that the output corrupted `"url"` from string to float. The lesson is straightforward: a `NULL` that only inspects the fixture surface can still fail when the parameter is defined over downstream consumability rather than fixture values.

6.6 Model-Dependence of Findings

A question PDM's methodology raises but cannot settle on the evidence of one model pair: are the rubric entries that emerge from PDM model-dependent? If the experiment were re-run with GPT-4 in place of Codex, or Gemini in place of Claude, would the convergence points change?

On the evidence of this experiment, the answer is plausibly yes at the margin and plausibly no at the core. The six weak-boolean-redundancy signals in P3 (analyzed in detail in the next subsection) are the marginal cases: each was accepted on a decomposition argument that a different model pair might have rejected. The P2 failure-class axes are the core cases: both models independently recognized the fabricated-three-element pattern and closed it consistently across transport, encoding, and body-shape axes. The core is more likely to replicate; the margin less so. This is a falsifiable prediction and motivates the cross-pair replication listed under future work (Section 11).

6.7 Weak-Boolean-Redundancy: When Convergence Accepts Weakly-Distinguishing Signals

Six of the eleven accepted P3 signals carry explicit weak-boolean-redundancy caveats in the fixture notes: `semantic_html`, `machine_readable_blocks`, `link_profile`, `selector_affordance`, `content_depth`, and `text_container_profile`. That is more than half of P3's accepted output. The pattern is significant enough to analyze separately, because it is the weakest part of the P3 result and the one that would change most under a larger or more diverse fixture set.

What the pattern looks like. A signal is flagged weak-boolean-redundant when its derived boolean projection (e.g., `semantic_html.present`, `machine_readable_blocks.present`) takes the same value across visible fixtures as a previously-accepted signal's boolean projection. For example, by iteration 4, `semantic_html.present` was true on V1/V2/V4 and false on V3/V5 — identical to the pattern of `heading_hierarchy.valid`. The derived boolean adds no new partition over the visible set.

How these signals cleared the bar anyway. In each case, the signal was accepted because of sub-field or holdout behavior that broke the redundancy at a level below the derived boolean. `semantic_html.semantic_element_count` distinguishes V1 (8) from V4 (11) where all prior signals collapsed. `machine_readable_blocks.address_count` exercises a measurement dimension that is flat at zero on all visibles but non-zero on holdout H1. `selector_affordance`'s boolean is recoverable from prior signals on visibles, but both holdouts break the prior-signal prediction in opposite directions. The acceptance logic, in other words, relied on out-of-distribution behavior to rescue signals that the in-distribution test declared redundant.

Why this is the right place to be skeptical. Two different reviewers could read the pattern two different ways. The charitable reading is that the protocol correctly identified signals that carry non-redundant information in a dimension the visible set does not exercise, and the holdouts are doing exactly the job they were designed to do. The less charitable reading is that the acceptance criterion is too permissive once the boolean projection is redundant, and accepted signals are disproportionately sub-field decompositions of patterns the existing schema already captures. Both readings are defensible on the current evidence. The `reference_context` case at iteration 12 is the sharpest version of the tension: Claude's own analysis showed that its derived `mode` category was fully recoverable from prior signals on all seven fixtures, and it was accepted on sub-field decomposition grounds anyway.

Implications for fixture sizing. The weak-boolean-redundancy pattern is an empirical argument for larger fixture sets in future PDM runs. Seven fixtures is enough to make every P3 signal falsifiable, but not enough to make every acceptance robust to adding new fixture categories. A production rubric derivation should run PDM on a fixture set of 20 or more with holdouts proportionally larger — not because the current signals would be falsified, but because a larger set would either retire the weak-boolean-redundancy flags or harden the decomposition arguments into genuinely orthogonal distinctions. This is cheaper to do than it is to argue about in the abstract.

7 What PDM Produces and Does Not Produce

7.1 What PDM Produces

PDM produces a rubric whose entries each have a documented provenance: the iteration that added the entry, the iterations that declined to modify or remove it, and the rejection reasons for nearby candidates that did not make the cut. Every entry answers the question "why is this here, and not something else?" with a specific log reference.

This is a weaker claim than "PDM produces a correct rubric." No mechanism in PDM validates the rubric against external ground truth. The rubric is correct *relative to the scope document*, which the researcher wrote. A badly-scoped parameter produces a badly-grounded rubric (P1 is the experiment's own negative example).

The honest way to state PDM's contribution, which should be preferred over the looser phrasing used in casual summaries: PDM shifts accountability from "the researcher chose these signals" to "the researcher chose this scope, and these signals are what two models independently converge on under it." The scope choice remains researcher-driven; the signal-choice step within that scope is what PDM externalizes. That is a real contribution — it is strictly more auditable than unconstrained rubric authorship — but it is not the stronger claim that PDM externalizes rubric construction end-to-end.

7.2 What PDM Does Not Produce

PDM does not produce signal weights. Coverage Completeness (P3) converged on eleven signals; the experiment does not claim those eleven signals are equally important for scoring a page's machine-readability. Weight derivation requires a different instrument: either labeled data (pages rated as agent-usable or agent-hostile) or a downstream pipeline whose success is measurable (task-completion rates). This paper addresses only criterion derivation, not weighting.

PDM does not produce per-signal thresholds. P3 added `js_dependency.dependent` as a boolean signal, but "what value of `script_count` should flip the boolean" was set by the proposing model's internal judgment, not by convergence. A principled threshold derivation would run a second-order PDM loop on the threshold function parameters, which this paper does not attempt.

PDM does not produce generalization guarantees. The eleven signals came from seven fixtures. Fixtures were representative by construction but not statistical. A signal converged-on in PDM may fail on an unseen fixture class (the weak-boolean-redundancy flags in Section 6.7 are explicit acknowledgments of this risk).

8 Discussion

8.1 What Bilateral NULL/NULL Actually Signals

Bilateral `NULL` in PDM is not a claim of absolute truth. It is a claim that, under the current parameter definition, current acceptance criteria, and current fixture architecture, two independently prompted frontier models both failed to identify a qualifying move. P2-18 says this directly: remaining candidates were "not crashes, not fabrications, already clean errors, or speculative defensive checks." P4-10 states the same idea even more sharply: "What remains is not ambiguity in the emitted JSON. It is preference." In other words, bilateral `NULL` marks procedural closure, not metaphysical completeness.

8.2 Why P1 Did Not Converge

P1 failed for reasons that later protocol documents address directly, even if the repository does not package them as a single five-item postmortem.

First, P1 lacked an authoritative ground-truth oracle. The later P3 fixture suite explicitly says that ground-truth values had to be computed before iteration 1 because otherwise models "cannot be held to the falsifiability rule."

Second, P1's notion of a qualifying structural move was too vague. The original plan defined `NULL` generically as the absence of a structural improvement, but later v2 materials had to narrow that substantially. `PROMPT_TEMPLATE.md` explicitly records that P2 dropped a "correct error classification" clause because it was undefined and invited cosmetic or message-quality moves.

Third, P1's scope was too broad. Both `P3_SCOPE.md` and `P4_SCOPE.md` cite P1 scope drift to 35 iterations without convergence as the reason for binding INCLUDE/EXCLUDE lists and hard caps.

Fourth, P1 had no falsifiability discipline. P2 adds mandatory red/green fixture evidence, P3 requires visible-fixture diversity tables, and P4 requires a concrete consumer-failure pair. None of that structure existed for P1.

Fifth, the later protocol also reduced the burden of accumulated conversational state by externalizing iteration context into repository files and a reusable master prompt. The repository does not isolate "accumulated context bias" as a separately quantified causal factor, but it does show that later iterations were deliberately redesigned to depend less on free-form conversation state and more on repo-state reconstruction.

Taken together, P1 was an under-instrumented search for browser-faithful DOM equivalence. P2–P4 were controlled experiments.

8.3 Generalizability

PDM is not a general substitute for human evaluation of open-ended generation. It depends on a specific kind of task: the artifact must be diffable, its outputs must be replayable, and at least part of the failure surface must be observable via fixtures, downstream consumer accesses, or both. That is why the method fits a measurement script well and would fit other deterministic or semi-deterministic code-quality tools, but would be much less convincing for unconstrained creative generation.

8.4 Cost

The direct cost is easy to count: $35+18+14+10=77$ iterations, each of which required one model inference plus human operator work to save artifacts, run checks, and update logs. The repository does not track token counts, wall-clock latencies, or operator minutes separately, so precise cost numbers cannot be reported.[1] The method is therefore inexpensive relative to large benchmark construction, but not free: it trades broad annotation expense for iterative operator supervision on one artifact.

[1] Author-reconstructed estimate, to be replaced with measured values in a subsequent revision: approximately [X *operator-hours*] across the 77 iterations and on the order of [Y *total tokens*] of model inference. These numbers are deliberately left as placeholders rather than guessed.

8.5 No External Validity Check

Nothing in PDM validates that pages scoring well under the derived rubric are, in fact, more machine-readable than pages scoring poorly. This paper does not perform a downstream pipeline evaluation. The AXO framework is planned to run exactly this evaluation in a subsequent phase ("Part 2" of the project, which integrates the P4-final script into Syphon's scoring pipeline and measures agent task-completion rates against pages rated by the rubric), but that phase is outside this paper's scope. Bilateral `NULL` proves scope-internal stability; it does not prove external utility. Any claim PDM makes about rubric quality is contingent on the eventual Part 2 evaluation.

9 Threats to Validity

The previous section frames PDM's honest limits. This section isolates threats that do not fit cleanly under "limitations" because they are risks to the *interpretation* of the results rather than caveats on scope. A reviewer in a methodological venue will ask about each of these; the protocol's credibility depends on naming them directly.

Prompt-template confound. Every rule a model operates under — what counts as a qualifying move, what `NULL` means, what fixtures to consult — was written by one researcher. The prompt materials were themselves iterated during the experiment (Table 1). As a result, statements framed as being about model behavior (e.g., Section 6.1's `NULL` asymmetry) are more precisely statements about *model behavior under this specific template*. A second researcher writing the prompt might produce different asymmetries, different convergence points, or different move-category taxonomies. This is not fixable by post-hoc analysis; it is fixable only by replication with prompt variation.

Operator as hidden third judge. The protocol is described as a two-model alternation. In practice, one human operator ran the loop: saving artifacts, verifying that changelogs cited specific fixtures and code paths, and writing the convergence record. When a model produced a non-`NULL` move that looked marginal, the operator's implicit adjudication was the forward-progress signal. No inter-rater reliability measurement exists for this role because the operator was a single person. The appendices preserve the raw model outputs, which makes operator influence reconstructible in principle, but the paper does not attempt to quantify it.

Version irreproducibility. "Claude" and "GPT/Codex" refer to the specific model versions offered during April 2026 runs. Neither service offers retrospective version pinning. Replication attempts will necessarily use newer model versions and may reach different convergence points on the same scope and fixtures. This is a threat to the *output* reproducibility of the experiment, not to the *protocol* reproducibility. The paper's primary claim is about the protocol; the specific convergence points, including which signals P3 accepted and where P4 closed, are second-order findings whose replication is expected to be imperfect.

Fixture leakage. The P3-6 `answer_first` signal used an `informativeSentencePattern` regex that Claude itself later flagged as mixing generic patterns with fixture-specific content words. The signal was accepted anyway. This is a positive flag for the paper's transparency, but it is also a case where the acceptance rule did not catch a leakage risk. A tighter protocol would add an explicit fixture-independence check to the qualification criteria — any proposed signal whose measurement code mentions fixture-specific tokens should require additional holdout evidence before acceptance.

Small-n behavioral claims. Section 6.1 reports 24 vs. 3 `NULL`s on one model pair with 77 total iterations. This is a strong directional signal for the specific configuration, but the denominator is too small to support confidence intervals on rate differences, and there is no natural null distribution to test against. Behavioral claims in this paper are qualitative descriptions of the observed run, not statistical generalizations. Future runs should be designed with this in mind: either a larger iteration budget per parameter, or multiple independent runs with the same scope.

Confirmation bias in the mid-experiment protocol rewrite. The v1→v2 transition happened after P1 failed. The designer of v2 had seen P1's failure modes, which means v2 is selected to address those failure modes. P2–P4 therefore test v2 on tasks where v1 was already known to be inadequate. This is a standard limitation of any iteratively-designed methodology and is not secretly fixable, but a clean future run would apply v2 to a fresh task from iteration 1 without any v1 baseline.

10 Limitations

The threats above cover the interpretive risks. This section lists the scope and coverage constraints that remain even under the most favorable interpretation of the results.

- The experiment used two model families, not a larger jury. The observed asymmetry may not replicate with different pairings.
- Iteration caps were pragmatic choices justified in scope documents, not statistically optimized stopping rules.
- The category system for P4 moves (A–E) was introduced only when the output-clarity parameter began, not from the start of the full experiment.
- The dominant P2 bug class, the fabricated synthesized three-element document, is specific to this script and parser stack. The methodology generalizes better than the bug class does.
- The repository does not include a standalone P1 postmortem file quantifying every proposed root cause. Some discussion in the "Why P1 did not converge" subsection therefore reconstructs protocol lessons from later documents rather than quoting a single canonical postmortem.
- Seven fixtures is a small sample. Weak-boolean-redundancy flags in P3 (Section 6.7) are explicit acknowledgments that adding fixtures could force re-classification. A production rubric derivation would run PDM on a fixture set of 20 or more, with holdouts proportionally larger.

11 Future Work

The paper's contribution is a methodology and one worked application. Three follow-on directions are most valuable, in rough order of priority.

Cross-model-pair replication. Replicate the protocol with different model pairings (GPT-4 + Gemini, Llama + Mistral) on the same scope documents and fixtures. Convergence points that replicate across pairs are strong candidates for model-independent rubric entries. Convergence points that differ reveal model-pair-dependent artifacts, which is itself useful information. This is the single highest-value follow-up: it directly addresses the prompt-template confound and the

small-n behavioral claim from Section 9, and it is cheap relative to running PDM from scratch on a new artifact.

Weight derivation via downstream evaluation. The eleven signals from P3 need weights before they form a scoring function. Weight derivation via PDM would require a parameter like "P5 — Signal Weighting," with moves of the form "increase weight of `structured_data` relative to `selector_affordance` by factor X" and a convergence condition tied to a held-out task-completion evaluation. This is the bridge to the AXO scoring pipeline described in Part 2.

Downstream integration and validation. Integrate `p4-final.js` as the production AXO measurement script and test whether the derived dimensions improve end-to-end scoring behavior in Syphon. This is the external validity check PDM itself cannot provide and the only evidence that the converged rubric measures what it was supposed to measure. The other extensions listed in earlier drafts (three-model PDM, automated mid-loop review, formalization of divergence signals, cross-domain PDM applications) are interesting but secondary to these three.

12 Conclusion

The repository provides a complete worked example of deriving evaluation criteria rather than assuming them. Across 77 alternating iterations, PDM produced one non-converged parameter, three converged parameters, a production-ready measurement script, and a visibly tighter experimental framework than the one it started with.

The central empirical result is not that two frontier models eventually agreed. It is that their pattern of disagreement and closure exposed which rubric dimensions were still unstable, which ones survived independent re-audit, and which methodological safeguards were necessary before a `NULL` could be trusted. The `NULL`-quality audit that emerged from P4 — requiring each new `NULL` to probe the surface that broke the last one — is a refinement that transfers beyond this application to any iterative convergence protocol with researcher-declared closure.

PDM does not produce ground-truth rubrics; it produces reproducible rubrics with documented provenance, conditional on the researcher-chosen scope. That distinction is the methodology's honest contribution and its primary limitation. For deterministic code artifacts with replayable outputs, it is a useful contribution in its own right.

References

[1] Artstein, R., & Poesio, M. (2008). Inter-Coder Agreement for Computational Linguistics. *Computational Linguistics*, 34(4), 555–596.

[2] Bai, Y., et al. (2022). Constitutional AI: Harmlessness from AI Feedback. Anthropic technical report.

[3] Brin, S., & Page, L. (1998). The Anatomy of a Large-Scale Hypertextual Web Search Engine. *Computer Networks and ISDN Systems*.

[4] Du, Y., et al. (2023). Improving Factuality and Reasoning in Language Models through Multiagent Debate. `arXiv:2305.14325`.

[5] Liang, P., et al. (2022). Holistic Evaluation of Language Models (HELM). `arXiv:2211.09110`.

[6] Madaan, A., et al. (2023). Self-Refine: Iterative Refinement with Self-Feedback. arXiv:2303.17651.

[7] Shinn, N., et al. (2023). Reflexion: Language Agents with Verbal Reinforcement Learning. arXiv:2303.11366.

[8] Srivastava, A., et al. (2022). Beyond the Imitation Game: Quantifying and extrapolating the capabilities of language models (BIG-bench). arXiv:2206.04615.

[9] Wang, Y., et al. (2023). Self-Consistency Improves Chain of Thought Reasoning in Language Models. arXiv:2203.11171.

[10] Zheng, L., et al. (2023). Judging LLM-as-a-Judge with MT-Bench and Chatbot Arena. arXiv:2306.05685.

Appendix

A Changes from Prior Draft

This revision implements structural changes to framing, adds two sections, reorders related work, and tightens the future-work discussion. Substantive factual claims and empirical results are preserved from the prior draft. The principal changes:

- **Title and abstract.** Revised to foreground the meta-methodological finding (`NULL`-quality audit from P4) and to explicitly qualify the rubric-provenance claim as conditional on researcher-chosen scope.
- **Related Work (Section 2).** Reordered to lead with paired-annotator corpus labeling (the tightest conceptual analogue) rather than closing with it. Multi-agent debate, iterative self-improvement, LLM-as-judge, Constitutional AI, and benchmark frameworks now read as contrast cases rather than primary framing.
- **Method (Section 3).** The v1→v2 evolution table has been pulled forward to serve as the section's organizing structure, and Figure 1 has been added to show the per-iteration decision flow. Section 3.5 (NULL Semantics) now includes the P4-5/P4-6 worked example inline rather than referencing it later.
- **Behavioral Findings (Section 6).** Section 6.1 renamed from "Claude/Codex Asymmetry" to "Observed NULL Asymmetry Under This Prompt Template" and extended with a three-cause analysis (model-family difference, prompt-template framing, position effect) that cannot be cleanly separated on $n = 1$ pair.
- **New subsection on weak-boolean-redundancy (Section 6.7).** The pattern affects more than half of P3's accepted signals and is the single strongest caveat on the coverage-completeness result; it now has dedicated analysis.
- **New Threats to Validity section (Section 9).** Separated from Limitations. Covers prompt-template confound, operator as hidden third judge, version irreproducibility, fixture leakage, small-n behavioral claims, and confirmation bias in the mid-experiment protocol rewrite.
- **Future Work (Section 11).** Cut from seven directions to three, ordered by priority. The removed items were not weak; they were simply drowning out the three that matter most.

- **"What PDM Produces" (Section 7).** The scope-internal qualifier is now promoted into the main claim rather than being available as a footnote-style caveat.
- **Appendices.** The original iteration logs, fixture specifications, prompt template, signal list, and selected changelogs are preserved substantially unchanged. These serve as archival evidence and are the strongest support for the methodology; condensing them would weaken the paper.

Readers familiar with the prior draft should find the empirical results identical. What has changed is the framing around those results and the honesty of the qualifications attached to them.

B Full Iteration Log Table

Full iteration-level logs for all four parameters, including per-iteration changes and `NULL` declarations, are preserved in the repository at `experiment-log.md` and in the per-iteration changelogs at `P-{1,2,3,4}/iteration-{n}-changelog.md`. The summary structure is as follows:

P1 Measurement Accuracy (35 iterations, no convergence). Claude `NULL`s at iterations 3, 5, 7, 9, 11, 13, 17, 21, 23, 25, 27, 29, 31, 33 (14 total). Notable GPT structural finds include template opacity (iter 4), implicit table wrappers (iter 6), document-wrapper synthesis (iter 8), `parseNoneClosedTags` (iter 32). Iteration 35 is non-`NULL` (added `plaintext` to `blockTextElements`); cap reached mid-progress.

P2 Edge Case Handling (18 iterations, converged at iter 18). Claude `NULL`s at iterations 5, 7, 9, 13, 17 (5 total). GPT `NULL` at iteration 18. Key structural moves include stack-overflow prevention (iter 1), HTTP 204/205 handling (iter 2), catch-branch ordering fix (iter 8), declared-charset validation (iter 14), post-parse zero-element guard with doctype exemption (iter 16).

P3 Coverage Completeness (14 iterations, converged at iter 14). Claude `NULL`s at iterations 11, 13 (2 total). GPT `NULL` at iteration 14. All eleven accepted signals listed in Section 5.3 were added across iterations 1–12.

P4 Output Clarity (10 iterations, converged at hard cap). Claude `NULL`s at iterations 5, 7, 9 (3 total). GPT `NULL` at iteration 10. The P4 move category distribution: A (iter 1), C/B (iter 2), D (iter 3), B (iters 4, 6, 8).

Full per-iteration summaries are in Appendix A of the prior draft and in the repository.

C Fixture Specifications and Ground-Truth Values

C.1 Fixture Inventory

C.2 Baseline Outputs Before P3

Full ground-truth values across all accepted P3 signals (the 7-fixture $\times$ 11-signal matrix) are preserved in `P3_FIXTURES.md` in the repository and in Appendix B.3 of the prior draft.

D Accepted P3 Signals

Signals with weak-boolean-redundancy notes are analyzed in Section 6.7. The `reference_context` case is the sharpest version of the tension: Claude's own analysis showed that its derived `mode` was

Table 7: Visible fixtures shown to both models.

#	File	Category and purpose
V1	`visible/v1-product.html`	E-commerce product page. JSON-LD Product + Offer + AggregateRating; spec table; one JS-LD script; semantic structure.
V2	`visible/v2-news.html`	News article, clean heading hierarchy. Article wrapper; one H1, multiple H2s with H3s; time element; no JavaScript.
V3	`visible/v3-spa.html`	JavaScript-dependent SPA shell. Near-empty body; multiple script tags; no real content in raw HTML.
V4	`visible/v4-blog.html`	Semantic-rich blog post. Uses header/nav/main/article/section/aside/-footer; no JSON-LD; strong structure signals.
V5	`visible/v5-divsoup.html`	Div-only marketing landing page. No semantic tags; no structured data; no headings; no JavaScript.

Table 8: Hidden holdouts never shown to either model.

#	File	Category and purpose
H1	`holdout/h1-localbiz.html`	Local business with LocalBusiness schema. Overfit check for structured-data and entity signals.
H2	`holdout/h2-hybrid.html`	Hybrid — summary SSR, interactive JS. Overfit check for JS-dependency threshold (has scripts AND real content).

recoverable from prior signals on all seven fixtures, and it was accepted on sub-field decomposition grounds.

E Reproducibility Commands

E.1 Environment

Node.js 20.x LTS; Python 3.11+ (for consumer-failure reproduction in P4); macOS 15.x (darwin 25.4.0) — portable to Linux; no GPU or special hardware required.

E.2 Running a Single Iteration

```
# From the repository root
npm install
npm run fixtures                 # start fixture server
node P-4/iteration-10.js http://localhost:<port>/fixtures/v1
```

E.3 Scope Documents

All scope documents are checked into the repository and frozen at parameter-start: `EXPERIMENT_PLAN.md`, `PROMPT_TEMPLATE.md`, `PROMPT_TEMPLATE_P3.md`, `PROMPT_TEMPLATE_P4.md`, `P3_SCOPE.md`, `P4_SCOPE.md`, and `prompts/MASTER_CODEX_PROMPT.md`.

E.4 Artifact Versioning

Every iteration produces two files: `P-{n}/iteration-{m}.js` (the script) and `P-{n}/iteration-{m}-changelog.md` (rationale). Parameter baselines: `p2-baseline.js` = `P-1/iteration-35.js`; `p3-baseline.js` =

Table 9: Baseline depth measurements before P3 signals were added.

#	Fixture	maximum_depth	average_depth	element_count
V1	v1-product	9	4.9	48
V2	v2-news	6	4.0	39
V3	v3-spa	2	1.7	13
V4	v4-blog	7	4.0	38
V5	v5-divsoup	8	4.7	46
H1	h1-localbiz	6	3.9	35
H2	h2-hybrid	5	2.9	25

Table 10: Eleven signals accepted in P3, with agent-task category and redundancy notes.

Iter	Signal	Agent-task category	Redundancy note
P3-1	`js_dependency`	no-browser SSR/SPA routing	holdout H2 prevented script-count overfit
P3-2	`structured_data`	fact extraction for product/business identity	holdout H1 flipped positive
P3-3	`heading_hierarchy`	outline-based extraction and summarization	clean non-redundancy case
P3-4	`semantic_html`	main-content isolation	weak boolean redundancy; sub-fields rescue
P3-5	`machine_readable_blocks`	in-content micro-data extraction	weak boolean redundancy; holdout H2 rescues
P3-6	`answer_first`	fast answer routing	accepted with fixture-leakage warning
P3-7	`link_profile`	crawl-out and fake-nav detection	`fragment_only` adds second axis
P3-8	`selector_affordance`	selector-first automation	weak boolean redundancy; both holdouts break prediction
P3-9	`content_depth`	readability / summarization routing	weak visible redundancy, holdouts rescue
P3-10	`text_container_profile`	extraction-primitive selection	weak redundancy on one projection, categorical novelty survives
P3-12	`reference_context`	multi-hop research crawl	accepted on sub-field decomposition despite mode reducibility

`P-2/iteration-18.js`; `p4-baseline.js` = `P-3/iteration-14.js`; `p4-final.js` = `P-4/iteration-10.js` (intended for AXO integration).

E.5 Model-Version Caveats

See Section 9. Replication attempts will use different model versions and may reach different convergence points. The protocol — not the output — is what the paper claims is reproducible.

E.6 License and Authorship

Repository released under a permissive license. The paper and experiment are authored solely by Siddharth Nigam; no co-authorship is asserted on behalf of the models whose outputs constitute the experimental data. Model outputs are treated analogously to instrument readings: recorded, cited, and analyzed, without attribution of authorship to the instrument.