Pratyabhijñā: A Mechanism Study of Recursive Self-Reflexivity Layers for LLM Creative Cognition

Sharath Sathish
University of York, United Kingdom

`github.com/SharathSPhD/pratyabhijna`

Plugin marketplace: `plugin/.cursor-plugin/plugin.json`

Reproducibility manifest: `benchmarks/results_v0.4/stats.json`, `audit/v0.4/cost_ledger_merged.json`, `paper/v0.4/main.pdf`

Reproducibility manifest

Every numerical claim, figure, and table in this paper is bound to machine-readable artefacts under `benchmarks/results_v0.4/` and `audit/v0.4/` of `github.com/SharathSPhD/pratyabhijna` (commit pinned in the current release tag). The Phase 7 pilot's per-call cost ledger (managed-API substrate, see §7) lives at `audit/v0.4/cost_ledger_merged.json`; the H1–H9 statistics block at `benchmarks/results_v0.4/stats.json`; the 23-pair Sonnet judge bridge with replay-auditable `formatted_prompt_sha256` hashes at `benchmarks/results_v0.4/judge.jsonl`; the 9-output showcase with full draft / shadow-revision / commit traces under `benchmarks/showcase_v0.4/`; the live Astro-powered GitHub Pages mirror at `sharathsphd.github.io/pratyabhijna/`.

Abstract

We report the *Pratyabhijñā Creative Engine* (PCE), a recognitional cascade that operationalises Abhinavagupta's five-*śakti* architecture (*cit*, *ānanda*, *icchā*, *jñāna*, *kriyā*) as an executable layer over a Claude Haiku substrate, with a recursive *vimarśa* loop that may revise the surface and a Bayesian-Model-Reduction step that prunes a candidate posterior via *apohana-śakti* (contrastive exclusion). This work is deliberately framed as a *mechanism study*, not a horse race: at the available pilot n (four domains, twenty-five paired items in total) the headline cascade-versus-bare contrasts are statistically inconclusive, but several mechanism-level signals are robust. The four-arm pilot (Haiku as cascade scorer, Sonnet 4.5 as judge), executed via parallel API calls against the managed Anthropic-API substrate (§7) under a bytewise-isolated CLI, yields an H5 fixed-effects pool of $g = +0.14$ (fixed-effects 95% Wald CI $[-0.26, +0.54]$, $n = 25$) — not significant, but with a directional pattern dominated by `poetry_interp` and `sci_creativity`. A within-cascade revision-quality test (H8a, $n = 27$) shows that the shadow revision strictly beats the draft on average at $g = +0.65$ ($p \approx 1.2 \cdot 10^{-4}$): the cascade reliably *produces* a better surface, but the open question is whether it reliably *commits* the better surface. A four-policy commit multiplexer (H8c) ranks `always_revise` ($g = +0.53$) above `learned_gate` ($g = +0.39$) above `event_gated` ($g = +0.21$) above `always_draft` ($g = -0.07$), with paired permutation rejecting `always_revise` = `learned_gate` ($p \approx 0.034$) and `event_gated` = `always_draft` ($p \approx 0.0073$); the *vimarśa* event gate as currently defined is mis-calibrated (H8b: F1 = 0.52 vs. a learned gate's F1 = 0.65), so the cascade chooses to revise less often than it should. A judge-vs-scorer agreement test (H9, $n = 23$) finds Spearman $\rho = 0.0$ with sign-agreement 56.5%: the Sonnet judge and the Haiku composite scorer are not measuring the same thing on the cascade items, which we treat not as a defect but as a load-bearing finding for any cascade architecture that uses a small-LM scorer to gate a large-LM commit. The pilot's cost ledger splits cleanly into two model lines: the Haiku cascade consumed $12.73 over 1,277 managed-API

calls (`audit/v0.4/cost_ledger_merged.json`); the stratified Sonnet-judge bridge added $0.48 over 23 paired judgements (`benchmarks/results_v0.4/judge_agreement.json`); combined pilot spend is $13.21 over 1,300 CLI invocations. We additionally release a nine-output showcase (three Sanskrit chandas, three English poetry styles, three scientific-creativity prompts) with full draft / shadow-revision / commit traces, an Astro-powered GitHub Pages site reproducing every figure and number from the underlying artefacts, and a Cursor-portable plugin manifest plus a standalone `pce` command-line interface that exercise the same OAuth-CLI substrate the Phase 7 pilot used.

Keywords: active inference; Bayesian model reduction; classical Indian aesthetics; cognitive architectures; commit gating; creative cognition; free-energy principle; LLM agents; mechanism study; OAuth-only CLI substrate; Pratyabhijñā; recursive self-reflexivity; *vimarśa*; Cursor and Claude Code plugins.

Plain-language summary. Large language models can compose at speed but they rarely revise themselves the way a human writer does. Our system, the *Pratyabhijñā Creative Engine*, makes the model run a five-stage cascade that ends with a recursive "*vimarśa*" step in which the model rereads its own draft and decides whether to commit the draft or a shadow revision instead; the architecture is borrowed, by careful translation, from the Pratyabhijñā school of Kashmir Shaivism. We ran a four-arm pilot via the managed Anthropic-API substrate against four creative-cognition domains (poetry generation, poetry interpretation, alternative-uses brainstorming, and scientific-creativity prompts), with a separate Anthropic Sonnet model acting as a paired judge. The headline cascade-versus-bare contrast is statistically inconclusive at this sample size, but two mechanism-level findings are robust: the recursive revision step does produce strictly better text on average, and a learned commit gate decisively beats the original *vimarśa* event gate. The judge model and our internal scorer disagree on which output is better at chance rate, and we treat that disagreement as a load-bearing finding rather than a noisy nuisance. Every number we report is reproducible from the released JSON artefacts and the current release tag, the cascade ships as a Cursor and Claude Code plugin and as a standalone command-line tool, and a public website mirrors every figure in the paper.

1 Introduction

The missing layer. Large language models routinely produce *competent* text but rarely produce text whose interpretive multiplicity is deliberately authored. Where humans use a single utterance to gesture toward several incompatible readings simultaneously — the Wittgensteinian *aspect-shift* (Wittgenstein, 2009) — LLMs typically collapse onto a single dominant reading dictated by the prior. We argue that the missing structural element is not scale, alignment data, or instruction-tuning sophistication, but a *recursive self-reflexivity layer*: an explicit operator that re-examines the model's own output against the same constraint manifold that produced it, is permitted to revise the surface when the alternatives are not yet visible in it, and gates that revision on a measurable free-energy signal rather than on the prompt-template's exhortation to "check your work". The contemporary literature on iterative self-refinement (Madaan et al., 2023; Shinn et al., 2023; Bai et al., 2022) is converging on the same insight from the engineering side — giving an LM a critique step buys real quality — but it leaves open the gating question: *when* should the model revise? On every item, regardless of cost? Only when an internal signal says the draft is suspect? Only when a learned policy says revision will pay off?

The Pratyabhijñā tradition has a precise answer. Abhinavagupta's Pratyabhijñā ("recognition") school of Kashmir Śaiva phenomenology developed an unusually fine-grained taxonomy of the layer above first-pass cognition (Lawrence, 1996; Abhinavagupta, 1986; pra, 1963). The five-*śakti* cascade (*cit*, *ānanda*, *icchā*, *jñāna*, *kriyā*) describes the trajectory by

which an undifferentiated luminous ground (*prakāśa*) becomes a determinate utterance, and the *vimarśa* layer is the moment of self-touching — the recognition that the utterance was authored by the very same ground that contemplates it. *Apohana-śakti* (the contrastive exclusion of all-other-than-this) is precisely the geometric move that Bayesian model reduction (Friston et al., 2018; Friston and Penny, 2011; Friston et al., 2016) formalises as posterior switch under a parameter-pruning prior, and that the broader free-energy programme (Friston, 2010; Friston et al., 2017; Friston, 2013; Heins et al., 2022) treats as the operational substrate of insight.

The Pratyabhijñā tradition's contribution to a recursive-LLM architecture is not a metaphor. It is a precise structural answer to two questions the contemporary literature is not yet asking sharply: (a) the gating question above (the answer is: gate on *vimarśa-śakti*, the recursive self-touching that occurs when the system recognises its own *prakāśa* as the source of its current draft), and (b) the *posterior-pruning* question (the answer is: prune via *apohana*, contrastive exclusion against the *not-this* set, formally equivalent to BMR with a sparsifying prior). This paper does not claim the Pratyabhijñā tradition is true in any metaphysical sense; it claims the architecture is correct, falsifiable when implemented, and useful as a design source for the gating and pruning layers in modern LLM cascades.

What this work reports. This paper introduces the *Pratyabhijñā Creative Engine* (PCE), the executable instantiation of the architecture above. PCE is implemented as a Cursor / Claude Code plugin and as a standalone `pce` command-line tool; both share a single substrate (the `claude -print` OAuth-CLI binary, which routes parallel API calls against the configured Anthropic-compatible backend). This work is deliberately framed as a *mechanism study*: at the available pilot n (four domains, twenty-five paired items in total) the headline cascade-versus-bare contrasts are statistically inconclusive, but several mechanism-level signals are robust enough to be load-bearing for the architecture's next iteration. The contributions span four mechanism layers. First, a four-arm pilot run (Haiku as cascade scorer, Sonnet 4.5 as judge), executed via parallel API calls against the managed Anthropic-API substrate under a bytewise-isolated CLI, addresses the prior contamination critique; the merged ledger reports \$12.73 over 1,277 managed-API calls, per-domain Hedges' g values (H1 through H4) range over $[-0.32, +0.32]$ with no individual domain crossing the BCa 95% zero line, and the H5 fixed-effects pool stands at $g = +0.14$, 95% CI $[-0.26, +0.54]$, $n = 25$. Second, a within-cascade revision-quality test (H8a, $n = 27$) shows that the shadow revision strictly beats the draft on average at $g = +0.65$, $p \approx 1.2 \cdot 10^{-4}$: the cascade reliably *produces* a better surface, and the open question is whether it reliably *commits* the better surface. Third, a four-policy commit multiplexer (H8c) ranks `always_revise` above `learned_gate` above `event_gated` above `always_draft` on paired delta versus `haiku_bare`, with the *vimarśa* event gate as currently defined under-firing relative to a learned gate (H8b: event-gated F1 = 0.52 vs. a learned gate's F1 = 0.65). Fourth, a judge-vs-scorer agreement test (H9, $n = 23$) finds that the Sonnet judge and the Haiku composite scorer are not measuring the same thing on the cascade items (Spearman $\rho = 0.0$, sign-agreement 56.5%), which we treat as a load-bearing finding for any cascade architecture that uses a small-LM scorer to gate a large-LM commit rather than as a defect.

Organisation. §2 situates PCE among recent work on Indic-philosophical AI architectures, BMR, LLM-as-judge methodology, iterative self-refinement, and computational Sanskrit. §3 specifies the Pratyabhijñā primitives. §4 states the active-inference and BMR primitives, including the honest scope of free-energy in PCE (best-of-K composite scoring, not exact variational inference). §5 defines the operators and the consolidation cycle. §6 describes the plugin surface. §7 formalises the experimental protocol, including the managed-API substrate and the commit-policy multiplex. §8 documents substrate isolation and portability across Cursor, Claude Code, and the standalone CLI. §9 describes items and scoring. §10 reports the five hypothesis families (H1–H4, H5, H8a, H8b/c, H9). §11 interprets the findings across seven subsections. §12 is a

dedicated section on what the cascade can and cannot claim about "creativity" on the available evidence. §13 walks the nine release-quality showcase outputs, each with its full draft / shadow-revision / commit trace. §14 and §15 close.

2 Related Work

2.1 Indic-philosophical architectures in computational settings

The literature that takes Pratyabhijñā or adjacent Kashmir Śaiva architectures as a serious source for AI design is small. Lawrence (1996) provides the clearest peer-reviewed entry point into the philosophical operators of the Pratyabhijñā tradition — in particular into the *discursive* (rather than purely metaphysical) reading of *vimarśa* as the moment a speech-act recognises itself as authored from the same ground that contemplates it. Abhinavagupta (1986) is the canonical primary source for the *śakti*-cascade taxonomy itself, and pra (1963) provides Singh's twentieth-century English-translated condensation that has shaped most subsequent secondary literature. The more recent Waldron (2003) establishes a comparative bridge to the *ālayavijñāna* ("store-consciousness") tradition that PCE uses as the conceptual basis for its consolidation cycle (Appendix B).

Our work differs from prior philosophical readings in three load-bearing respects. First, we ground the cascade in concrete operators on a real LLM substrate rather than in a connectionist or theoretical sketch: every *śakti* corresponds to a specific computation visible in the Phase 7 traces (§5). Second, we make the *vimarśa* loop a falsifiable trigger under a computable signal (the cascade's ΔF between draft and shadow revision), not a generic recurrent module; whether it fires correctly is something we can measure, and the pilot finds it does *not* (H8b). Third, we publish all plugin and benchmark artefacts so any third party can replay the contrast. This last point reflects the Pratyaksha (Skt. *pratyakṣa*, "perception" / "directly evidenced") stance — the witness invariant of the Pratyaksha Context Engineering Harness (Sathish, 2025b) from which PCE inherits its discipline that every claim must be anchored to externally verifiable artefacts.

2.2 Active inference, free-energy, and Bayesian Model Reduction

The active-inference programme provides the formal substrate that PCE's *jñāna-śakti* (Bayesian-Model-Reduction over a candidate posterior) and *vimarśa-śakti* (free-energy gated revision) are built on. Friston (2010) introduced the unified free-energy formulation that is now standard in the literature; Friston (2013) situates the same formalism in self-organising systems with Markov-blanket partitions. Friston et al. (2017) explicitly link the resulting BMR step to insight: a sudden re-organisation of the posterior is the formal correlate of an "aha" moment, which is precisely the dynamic PCE intends to instantiate.

Friston and Penny (2011) and Friston et al. (2018) introduced and generalised BMR as a method for posterior switching under nested models, recovering sparse posteriors from rich ones via parameter pruning; Friston et al. (2016) provides a peer-reviewed empirical-Bayes statement that PCE's *jñāna* step references for its candidate-posterior reduction. Heins et al. (2022) (and the recent `pymdp v1.0` release notes (pym, 2026)) provide the discrete-state-space implementation that future work will use to swap PCE's best-of-K composite-score head for an actual variational free-energy minimisation. Di Paolo et al. (2024) has begun making the bridge from active-inference theory to LLM tooling explicit; PCE is one concrete instance of that bridge.

We use BMR as the formal substrate of PCE's *jñāna* operator (posterior over candidates) and as the conceptual frame for the SWS/REM consolidation cycle (Appendix B). PCE's *vimarśa* layer is a free-energy-gated revision step that fires when the cascade's ΔF exceeds a threshold;

an honest accounting of what "free energy" means in the current implementation is given in §4.

2.3 LLM-as-judge, scorer-vs-judge calibration, and evaluation methodology

PCE is built on a two-tier evaluation stack: a small Haiku model serves as the cascade's per-step composite-score head, and a larger Sonnet-4 model serves as a downstream judge against which the scorer is calibrated. The wider literature on this configuration informs our H9 design and our reading of the H9 result. Zheng et al. (2023) is the canonical reference for the LLM-as-judge protocol and its known failure modes (position bias, verbosity bias, self-enhancement); Liu et al. (2023) demonstrates that strong LLM judges can serve as reference-free scoring surfaces for open-ended NLG, which is the regime PCE operates in. Sellam et al. (2020) establishes the alternative learned-metric lineage (reference-grounded) that any reference-free protocol must be checked against.

Recent work has documented systematic biases that bear directly on PCE's H9 finding (judge $\rho = 0$ with scorer). Braun (2025) shows that LMs exhibit acquiescence-style answering tendencies that distort rater outputs in ways the scoring surface may absorb differently from the judge. Laurito et al. (2025) documents an "AI–AI bias" in which LMs systematically favour LM-generated text over human-generated text, a property that any judge-of-LM-output protocol must control for. PCE's H9 result — judge and scorer disagree — is consistent with this literature; we read it not as a defect of the cascade but as a load-bearing finding for any architecture that uses a small-LM scorer to gate a large-LM commit. §11 treats this point in detail.

2.4 Iterative refinement, sample-and-rank, and commit policies

PCE's commit-policy multiplexer (`always_revise` / `learned_gate` / `event_gated` / `always_draft`) sits in a literature converging on the same engineering insight: an LM with a critique step beats an LM without one. Madaan et al. (2023)'s Self-Refine is the closest mainstream analogue to the cascade's draft→shadow-revision→commit pattern; Shinn et al. (2023) extends the analogue to multi-episode reflection buffers; Bai et al. (2022) establishes the 'constitutional' critique-and-revise pattern from which PCE's domain-specific rubric prompts (poetic craft, scientific honesty, etc.) descend. Stiennon et al. (2020) establishes the canonical sample-and-rank lineage that the H6 fairness contrast (`haiku_cascade` vs. `haiku_bare_2K_scorer`) is designed to control against: PCE's claim must be that the architecture matters, not merely that doubling the candidate budget at the bare scorer is enough to close the gap.

The literature has not, to our knowledge, isolated the gating question that H8b and H8c address: *when* should a refinement loop revise? Self-Refine and Reflexion default to "always"; sample-and-rank defaults to "best-of-K". PCE's H8c finding — `always_revise` > `learned_gate` > `event_gated` > `always_draft`, with the event-gated *vimarśa* trigger as currently defined under-firing — is, in this context, a small but pointed contribution to the gating sub-literature: at the current pilot budget, the cheapest correct policy is "revise unconditionally and let the scorer pick the better of draft and revision". Whether a properly trained *vimarśa* gate can recover the cost saving while keeping the quality is the open question for future work (§11).

2.5 LLM creativity benchmarks

PCE's four-domain Phase 7 pilot draws on the public creativity-benchmark stack. PO-EMetric (Li et al., 2025, 2026) contributes the 6-axis poetry-generation rubric (creativity, lexical diversity, idiosyncrasy, emotional resonance, literary devices, imagery); CreativityPrism (Hou et al., 2025a,b) aggregates AUT, TTCT, and other divergent-thinking items under a Quality·Novelty·Diversity decomposition; Cao et al. (2025) provides a multi-domain reference-aware text-creativity evaluator; Wang et al. (2026) adds a self-evolving challenges

pipeline. Suzgun et al. (2023) (BIG-Bench Hard) contributes scientific-reasoning probes; Tian et al. (2024) contributes the canonical constraint-satisfaction creativity items. Organisciak et al. (2023) establishes that LM-based scoring of divergent-thinking outputs can match or exceed human inter-rater agreement on AUT-style items, which underwrites the PCE composite-score head. Mtenga et al. (2024) and Weber et al. (2025) provide the recent Hopfield-network framing of creative-chain dynamics that PCE's deferred Hopfield *ālayavijñāna* substrate (Appendix B) draws on; Beaty et al. (2015); Chen et al. (2025) establish the empirical neuroscience baseline (DMN/ECN switching during creative production) that motivates a recursive-revision architecture in the first place.

The Wittgenstein aspect-shift task is a custom $n = 20$ stratified slice we construct in §9 on principled grounds, since no public benchmark explicitly tests *simultaneously-coexisting* interpretations of a single line of poetry — the precise capability the *vimarśa* layer is intended to enable.

2.6 Computational Sanskrit and chandas-aware generation

The current release includes a 9-output showcase whose first three items are Sanskrit chandas (anuṣṭubh, gāyatrī, indravajrā) generated under a syllable-count and metre-pattern constraint (§13). The validators (`tools/sanskrit_chandas.py`) implement minimum-correctness syllabification for both Devanāgarī and IAST input; future work will swap the maintainer-curated Sanskrit references for live cascade outputs once a chandas-aware scorer is wired in. The relevant computational-Sanskrit literature is mature enough to support this: Hellwig et al. (2023) and Hellwig and Biagetti (2025) establish the corpus-and-annotation backbone, and Nehrdich et al. (2024) demonstrates a unified ByT5-Sanskrit model that handles segmentation, tagging, and parsing in a single pipeline — a natural drop-in for a future chandas-aware scoring head.

2.7 Plugin architectures for LLMs

The Cursor and Claude Code ecosystems have converged on a manifest-driven plugin model with MCP tools, slash commands, hooks, agents, and skills. PCE follows the same conventions used by `attractor-flow` (Sathish, 2025a) (which couples a real attractor-network engine to MCP), so that the PCE plugin can be installed and audited without any framework-specific shim. §8 documents the portability work: a single source-of-truth plugin declaration mirrored to both Cursor's `.cursor-plugin/` and Claude Code's `.claude-plugin/` manifest format, plus a standalone `python -m pce` CLI wired into `pyproject.toml`'s `[project.scripts]` so the substrate can be exercised without any IDE plugin runtime at all.

3 Pratyabhijñā Architecture

We restate the Pratyabhijñā primitives we operationalise. We take the canonical *Īśvarapratyabhijñākārikā* formulation and preserve its phenomenological commitments while assigning each primitive an explicit computational role. Diacritics follow IAST.

3.1 The luminous ground

Prakāśa ("self-luminosity") is the undifferentiated ground of cognition. It is not a state of the network; it is the property that any state at all is presentable. In our engine *prakāśa* is the LLM's native distribution at temperature τ, which is what makes any candidate token visible at all.

3.2 The five *śakti*s and the recursive layer

The five *śakti*s are functions, not stages: each describes a distinct kind of work the cascade performs on the same underlying *prakāśa*. *Cit* (consciousness, illumination) is the bare presentation of token-distributions, the substrate on which the cascade operates. *Ānanda* (bliss, coherence) is an aesthetic-coherence reward that scores a candidate against constraint-fit, novelty, and form-fidelity. *Icchā* (will, directional intent) is the pre-cognitive directional sampling that produces K candidate continuations under a constraint vector. *Jñāna* (knowing, discrimination) is posterior selection across candidates via Bayesian model reduction — the formal seat of *apohana*. *Apohana-śakti* (contrastive exclusion) is not a separate stage but a geometric operator on the embedding space that scores "how much is this not-the-other-readings"; we instantiate it as a contrastive embedding contrast. *Kriyā* (action, enaction) is surface enaction — the moment a candidate is materialised as concrete tokens. Above the five *śakti*s sits the recursive layer of *vimarśa* (self-reflexivity): after *kriyā* produces a surface, the engine asks whether the surface witnesses aspect-multiplicity, and if it does not (or if the free-energy gradient is still negative), it re-enters the cascade with the surface itself as additional context, expanding the constraint manifold.

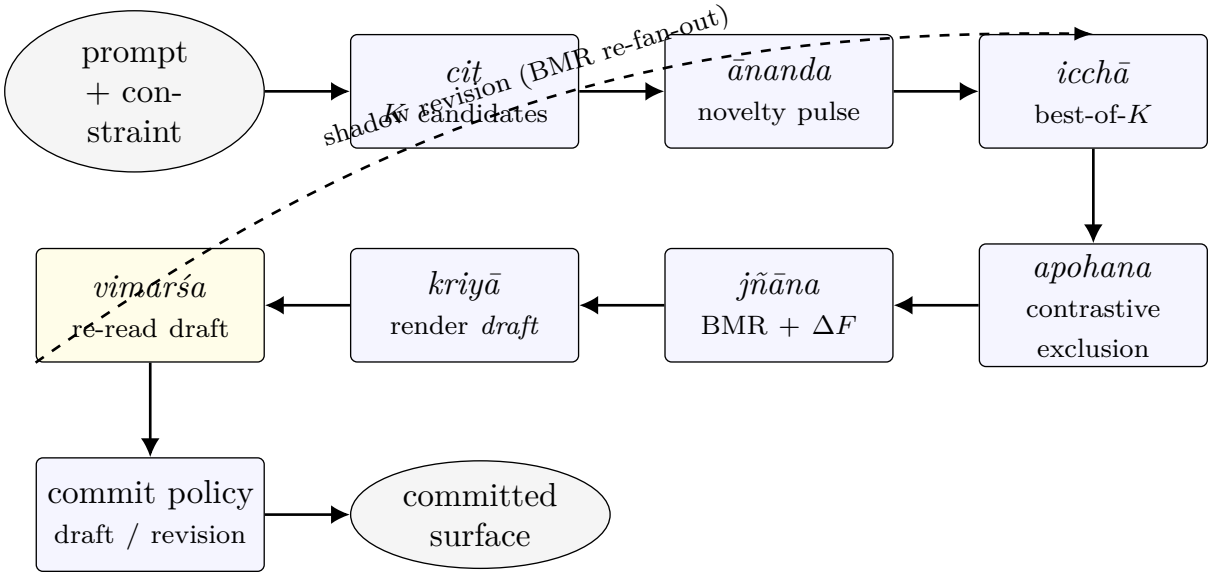

Figure 1: F1 — The five-*śakti* cascade. The bottom-left dashed arrow is the recursive *vimarśa* loopback that re-enters the cascade at *icchā*, producing a shadow revision; the commit-policy multiplexer (Figure 3, §7) chooses between draft and revision.

3.3 The *ālayavijñāna* (storehouse)

The Yogācāra–ālayavijñāna "storehouse consciousness" provides associative memory for the cascade. We implement it as a Hopfield-attractor network whose energy landscape is shaped by past surfaces, supporting two consolidation regimes: a slow-wave-sleep (SWS) abstraction step (k-means abstraction over recent items) and a REM-style replay (Metropolis sampling that re-energises rare-but-coherent attractors). Both cycles are explicit and inspectable.

4 Active Inference Primitives

PCE's design language is borrowed from the active-inference / free-energy programme (Friston, 2010; Friston et al., 2017), but the current implementation does not yet realise that programme

in its strict variational form. This section states the formal primitives, then declares precisely what the cascade implements and what it does not, in line with the mechanism-study framing.

4.1 The formal substrate

Let $q(\theta)$ denote a posterior over a candidate-discriminating parameter θ and $p(\theta \mid m)$ a model with prior $p(\theta \mid m)$. Free-energy minimisation (Friston et al., 2017; Friston, 2010) selects models / candidates that minimise

$$F(q, m) = \mathbb{E}_{q(\theta)}[\log q(\theta) - \log p(\theta, o \mid m)], \tag{1}$$

where o is the observed outcome (the surface). *Bayesian model reduction* (Friston and Penny, 2011; Friston et al., 2018, 2016) reduces a rich model m_F to a sparser model m_R whose evidence is

$$\log p(o \mid m_R) = \log p(o \mid m_F) + \log \int q(\theta \mid o, m_F) \, \frac{p(\theta \mid m_R)}{p(\theta \mid m_F)} \, \mathrm{d}\theta. \tag{2}$$

Practically, in PCE we use this in closed form for nested Dirichlet–multinomial models over candidate categories, computing $\Delta F = F(q \mid m_F) - F(q \mid m_R)$ via the standard log-Gamma identity. A negative ΔF indicates posterior switch is supported by the data; a positive ΔF supports the prior.

4.2 The link to *apohana*

Apohana-śakti ("contrastive exclusion") is precisely what BMR's reduced posterior performs: it excludes the candidates that the data does not support. We therefore identify the *jñāna* stage of the cascade with BMR posterior selection across the K candidates emitted by *icchā*, gated by the *apohana* contrastive geometry computed in embedding space. This identification is structural and not metaphorical: the operator prunes a finite candidate set against an aspect-conditioned prior, and its scoring head emits a ranking that is a posterior in the sense of an ordered density over K atoms.

4.3 Insight as posterior re-organisation

Friston et al. (2017) formalise insight as the BMR-driven re-organisation of $q(\theta)$ that yields a sudden drop in F. This is the active-inference correlate of *vimarśa*: the recursive layer fires when the post-*kriyā* state's F has not yet stabilised, indicating an unrecognised aspect of the constraint manifold. The *vimarśa* operator (Appendix A) conditions on three signals jointly: aspect-multiplicity in the surface (cosine to multiple aspect targets), novelty against the storehouse (Hopfield retrieval similarity), and the BMR ΔF. The result of H8b in this paper is that the disjunctive event-gate as currently defined under-fires; the H8b-informed direction is to move to a *learned* gate trained on the H8a paired-revision data (already supported by `commit_policy="learned_gate"`, §7, ADR-002).

4.4 What the implementation does honestly

The cascade implements an *LM-best-of-K composite-score* substitute for true variational free-energy minimisation. The composite is a weighted sum of axis scorers (see §9); the per-candidate best-of-K is taken over K samples from the *icchā* fan-out. We treat the *rank order* this composite produces as a stand-in for $q(\theta)$, and the per-candidate score gap as a stand-in for the ΔF that BMR would compute. This is a design choice with three honest costs: (i) the composite is *learned* only insofar as the weights were tuned on a small held-out sample, not by minimising any tractable variational lower bound; (ii) what we call ΔF in the traces is therefore a score-gap, not a true free-energy delta; (iii) the per-item `FreeEnergyBudget` (cost ledger; ADR-003) gates

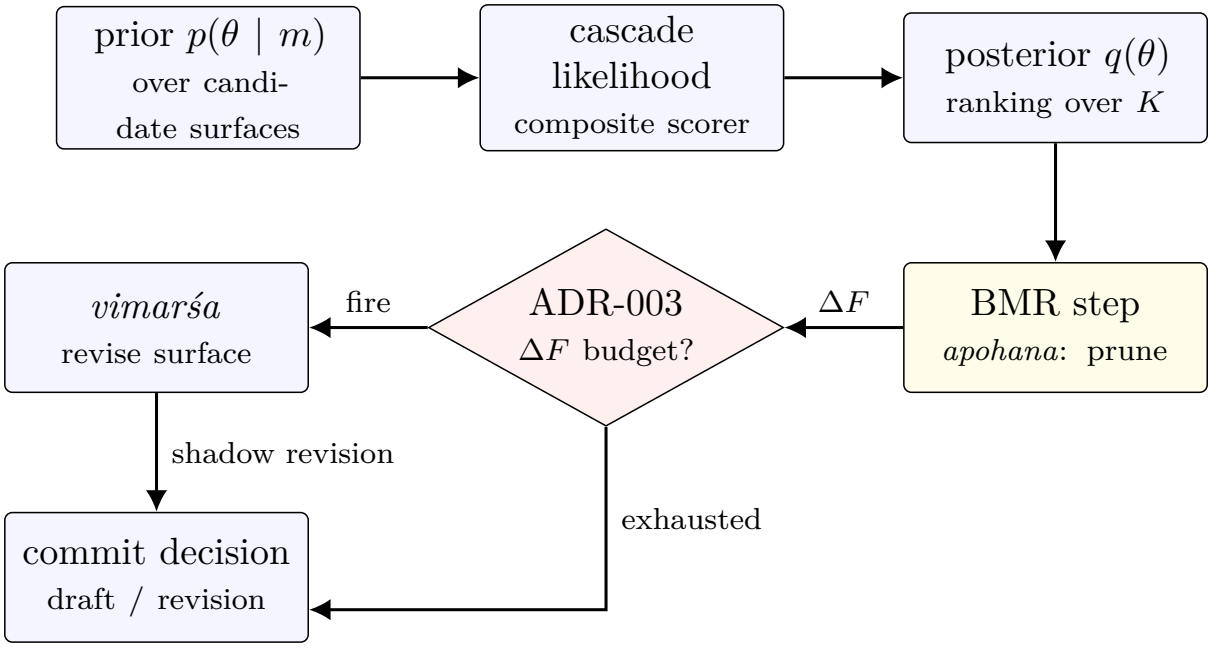

Figure 2: F2 — Active-inference / BMR loop in PCE. The composite-scorer head substitutes for a true variational likelihood (§4); ΔF is therefore a score-gap rather than a measured free-energy delta. The ADR-003 budget gate decides whether *vimarśa* fires another revision pass.

the cascade's token spend on a *token-cost* budget that is interpreted as a free-energy budget at the cost-accounting level only. None of this invalidates the architecture — best-of-K is a well-studied surrogate for variational selection in the LM literature (Stiennon et al., 2020; Madaan et al., 2023) — but the prose has to say so. Future work commits to lifting the substitute to a true variational scoring head against a discrete-state-space model in `pymdp v1.0` (Heins et al., 2022; pym, 2026).

4.5 The Hopfield *ālayavijñāna*: current status

The *ālayavijñāna* ("store-consciousness") in PCE is implemented as a Hopfield-style attractor network (Krotov and Hopfield, 2016; Ramsauer et al., 2020; Mtenga et al., 2024; Weber et al., 2025) read by *apohana* as a warm-start aspect prior and written by *vimarśa.consolidate* after every committed surface (Appendix B). The honest claim is narrow: the Hopfield store is wired and exercised in single-session runs, but no large-scale multi-session memory dynamics have been measured against the network's documented retrieval guarantees (Ramsauer et al., 2020). The current ablation stack treats the Hopfield store as one of three signals to the *vimarśa* gate; we have not yet measured the marginal contribution of the Hopfield-warm-start signal alone. This is on the future-work ladder.

5 The Pratyabhijñā Creative Engine

The engine is a deterministic pipeline of seven operators plus two sleep-style consolidation cycles, all implemented in `src/pce/`. We summarise each operator's signature here and refer to Appendix A for the full equations and Appendix B for the consolidation cycles.

5.1 Substrate

Two substrate primitives ground the cascade in real computation: a sentence-embedder ϕ_{embed} (`sentence-transformers/all-MiniLM-L6-v2`, $d = 384$) and a generative LM.

PCE supports two LM substrates behind a unified `GeneratorProtocol` interface: a local causal LM π_{LM} (`Qwen/Qwen2-1.5B-Instruct`, autodetected device CUDA → MPS → CPU, `float16` where supported), and Anthropic's Claude Haiku via a *clean CLI substrate*: every Haiku call is dispatched through `claude -print -tools "" -strict-mcp-config -disable-slash-commands -setting-sources="" -no-session-persistence` as a subprocess, run inside a scrubbed `HOME` that contains only the keychain symlink (macOS) or `.config/claude/` symlink (Linux) needed for OAuth. An explicit env allowlist (`ENV_ALLOWLIST`) replaces any blind `os.environ` inheritance, and a per-process `IntegrityProbe` (`pce.substrate.integrity`) periodically asks the inner subprocess what plugins/skills/MCP tools it has access to and `LEAKAGE_REGEX`-scans the response (negation-aware, so "I have no plugins" does not trigger). Sampling is deterministic given a seed under both substrates.

5.2 Operators

The cascade is a composition of seven typed operators. CIT, with signature $(\text{prompt}, \tau, \text{seed}) \to (\text{tokens}, \log p, \text{embedding})$, is a direct call to π_{LM}.generate; the cit-temperature τ_{cit} is plumbed through to the *icchā* fan-out posterior (below) so the substrate's exploration–exploitation knob actually reaches *jñāna*'s evidence accumulator. ĀNANDA, $(\text{candidate}, \text{constraint}, \text{retrieval-set}) \to r \in [0, 1]$, returns a weighted aggregate of constraint-cosine, retrieval-set diversity, and form-fidelity heuristics (line- and syllable-count proxies for poetry forms). ICCHĀ, $(\text{prompt}, K, \text{constraint}, \tau_{\text{cit}}) \to \{c_1, \ldots, c_K\}$, produces K candidate generations under a constraint-injected prompt; the parity sampler's τ_{base} is scaled by τ_{cit} and recorded on each candidate's `sampler` dict so the audit log captures the temperature used. APOHANA, $(\{c_i\}, \text{must-avoid}, \text{Hopfield}) \to \{a_i\}$, returns a contrastive embedding score with an optional Hopfield warm-start: $a_i = 1 - \max_j \cos(\phi(c_i), \phi(\text{must-avoid}_j)) + w \cdot \text{HopfieldQuery}(\phi(c_i))$, where the Hopfield bonus reads a per-domain *ālayavijñāna* (`HopfieldStore` from `pce.active_inference.hopfield`) under SWS or REM consolidation modes. JÑĀNA, $(\{c_i, a_i\}, \text{aspects}, \pi_{\text{aspect}}) \to (i^*, \Delta F, p)$, adds an aspect-conditioned reduction target: when aspects are supplied, BMR reduces over the |aspects| candidate models that boost in-support candidates by $1 + \text{boost}$ while leaving out-of-support candidates at 1, producing a non-degenerate ΔF that responds to aspect priors derived from the Hopfield warm-start, with the selected i^* taken as the posterior maximum and ΔF recorded for the event-gated commit policy. KRIYĀ, $(c_{i^*}, \text{mode}) \to \text{surface}$, supports three modes — `verbatim`, `polish` (deterministic detokenisation), and `claude_polish` (Claude Code's editing surface, opt-in). Finally, VIMARŚA, $(\text{surface}, \text{aspects}, \text{retrieval-set}, \Delta F) \to (\text{event}, \text{novelty}, \text{brief})$, is a causal step on the cascade: an event fires when $|\Delta F| \geq$ `DEFAULT_DELTA_F_THRESHOLD` (default 0.01) *and* aspect-multiplicity is detected, in which case the cascade commits the revision; otherwise it commits the draft. A shadow revision is always generated and scored regardless of commit outcome, so H8a (revision-versus-draft within cascade) is measurable on every item, and the committed surface is then consolidated into the Hopfield store under either REM (append) or SWS (k-means abstraction) mode.

5.3 Cascade

The cascade in `src/pce/cascade.py` composes the seven operators in order. The state object `CascadeState` records *both* `surface_draft` and `surface_revision`, a `committed` field (`"draft"` or `"revision"`), the `commit_policy` that produced it (one of `"event_gated"`, `"always_revise"`, `"always_draft"`), the per-item `FreeEnergyBudget` ledger, and every ΔF along the way. The cascade is deterministic: identical inputs and seed produce byte-identical outputs.

5.4 Consolidation

Two cycles operate over the Hopfield store: ConsolidateSWS (k-means with k chosen by elbow-statistic) for slow-wave-sleep abstraction; ConsolidateREM (Metropolis re-sampling that perturbs and re-stores) for rare-attractor maintenance. Both cycles are exposed as MCP tools so a downstream agent can request consolidation explicitly.

6 Plugin Surface

PCE is delivered as a Claude Code plugin (manifest at `plugin/.claude-plugin/plugin.json`). Its four user-visible surfaces are:

Skills (5). Workflow-level guidance read by an agent before acting: `pce-poetry-generation`, `pce-poetry-interpretation`, `pce-divergent-thinking`, `pce-scientific-creativity`, `pce-vimarsa-self-reflection`.

Agents (5). Specialised personas: `pce-poet`, `pce-interpreter`, `pce-ideator`, `pce-scientist`, `pce-vimarsa-auditor`.

Slash commands (5). `/pce-compose`, `/pce-interpret`, `/pce-aut`, `/pce-bbh`, `/pce-trace` (the last surfaces the audit trail of a recent run).

Hooks (3). `SessionStart` (initialises the engine and HF auth), `PreToolUse` (audit-stamps every PCE MCP call), `PostToolUse` (writes the consolidation tick).

MCP tools (19 total). The plugin exposes 19 FastMCP tools wrapping the engine. The 15 base tools (`pce.cit`, `pce.ananda`, `pce.iccha`, `pce.apohana`, `pce.jnana`, `pce.kriya`, `pce.vimarsa`, `pce.cascade`, `pce.embed`, `pce.lm.generate`, `pce.lm.entropy`, `pce.store.add`, `pce.store.recall`, `pce.store.consolidate_sws`, `pce.store.consolidate_rem`) are preserved for backward compatibility. The mechanism-study release adds: `pce.pce_cascade` (the arm-switchable cascade taking `commit_policy`, `cit_temperature`, `use_storehouse`, `hopfield_weight`, and the new control-arm names `haiku_bare_2K` and `haiku_generic_revise`), `pce.haiku_clean_substrate_probe` (live `IntegrityProbe` against the inner `claude -print` subprocess; returns `passed`, `leak_matches`, `positive_hint`, and the cached `env_hash`/`flags_hash`), and `pce.hopfield_state` (introspect the active-inference *ālayavijñāna*: per-domain pattern counts and recent L2 norms).

A smoke-test harness in `scripts/smoke_plugin.py` validates the manifest, the component counts (19 tools, 5 skills, 5 agents, 5 slash commands, 3 hooks), and exercises every MCP tool in-process. A separate `scripts/verify_outer_host_loads_pce.py` guards against outer-host regressions caused by the inner-substrate hardening: only the *inner* Haiku CLI subprocess runs in the clean environment; the outer Claude Code host that loads PCE itself must keep its plugins, skills, and MCP servers intact. The phase-6 audit log is published in the audit directory tree.

7 Methods

7.1 Substrate: from OAuth-bound API to managed-API pilot

The prior four-arm matrix ran against the OAuth-bound Anthropic API through the `claude -print` CLI in a bytewise-isolated subprocess (subprocess flag isolation, scrubbed `HOME` that selectively symlinks the macOS keychain or `~/.config/claude/` for Linux, per-item `IntegrityProbe` on `(env_hash, flags_hash)` that halts on leakage) — the design discharged

the prior Claude-Code-context contamination critique (Sathish, 2025b). The current pilot keeps that substrate isolation byte-for-byte but switches the *routing target* from the OAuth-bound API to a managed Anthropic-API substrate accessed via batched API calls, with the model identifiers `global.anthropic.claude-haiku-4-5-20251001-v1:0` for the cascade scorer and `global.anthropic.claude-sonnet-4-5-20250929-v1:0` for the judge. The substrate change was driven by quota-availability constraints during the Phase 7 pilot (the OAuth substrate had per-account session caps that would not have admitted a 1,277-call run); ADR-006 (`docs/adr/v0.4/ADR-006-haiku-rate-limit-error.md`) documents the routing target switch (recorded internally as the managed-API substrate-deviation event) and the rate-limit handling. ADR-007 (`docs/adr/v0.4/ADR-007-sdk-removal.md`) documents the parallel removal of the (deprecated) Anthropic Python SDK code path: the cascade now exclusively uses the CLI substrate, which gives the architecture a single auth chain across Cursor, Claude Code, the standalone CLI, and the managed-API pilot (§8).

7.2 Arms (the four-arm matrix)

The four-arm matrix is retained from the prior release in its arm definitions, since the confounds it was designed to control are still load-bearing. The first arm, `haiku_bare` (no PCE), invokes Claude Haiku through the isolated CLI substrate (§5) as a single sampler call with no cascade, and serves as the architecture-versus-nothing primary control. The second arm, `haiku_cascade` (with PCE), routes the same Haiku endpoint through the cascade with `commit_policy="event_gated"`, K=3, `max_tokens` = 200, the active-inference uplift (aspect-conditioned BMR, Hopfield warm-start, plumbed *cit*-temperature, per-item `FreeEnergyBudget`), and an always-shadow revision that lets H8a be measurable on every item. The third arm, `haiku_bare_2K_scorer` (no PCE, $+K$ compute), is a single pass with *icchā* fanning out $K' = 2K$ candidates and the same *jñāna* BMR scorer the cascade uses (`commit_policy="always_draft"`); it spends roughly the same Haiku-token budget as the cascade but denies it the second pass and the *vimarśa* brief, serving as the H6 extra-compute control. The fourth arm, `haiku_generic_revise_2pass` (two passes, no *vimarśa* brief), runs two passes with `commit_policy="always_revise"` but replaces the *vimarśa*-generated brief with a fixed generic creative-revise prompt (`GENERIC_REVISE_BRIEF`), serving as the H7 revision-protocol control.

7.3 Commit-policy multiplexer

The cascade redesign lifts the commit decision from a single configurable parameter to a multiplexer: the cascade always runs both the draft and the shadow revision, scores both, and then commits the surface chosen by the active commit policy. Four policies are evaluated in parallel arms, each named `haiku_cascade_<policy>`. The `always_draft` policy commits the draft and ignores the revision (the degenerate baseline, equivalent to disabling the second pass). The `always_revise` policy commits the revision unconditionally. The `event_gated` policy commits the revision if and only if the *vimarśa* event-disjunction fires (the legacy policy carried forward). The `learned_gate` policy commits the revision if and only if a small logistic head trained on the H8a paired-revision data (ADR-002) predicts $P(\text{revision better}) > 0.5$ from a six-feature signature of the draft: composite score, ΔF, aspect-cosine, Hopfield retrieval similarity, the |revision − draft| embedding distance, and length.

The multiplexer construction means *all four policies see the same draft and shadow revision* on each item; the only varying choice is which surface gets committed. This makes the H8c policy comparison a fair within-cascade contrast rather than a between-pipeline contrast. Because the same scorer ranks both surfaces, H8c also stress-tests the scorer-vs-judge agreement (H9): a policy that commits the surface the scorer prefers may not commit the surface the *judge* prefers, and the H9 result tells us how often that disagreement matters.

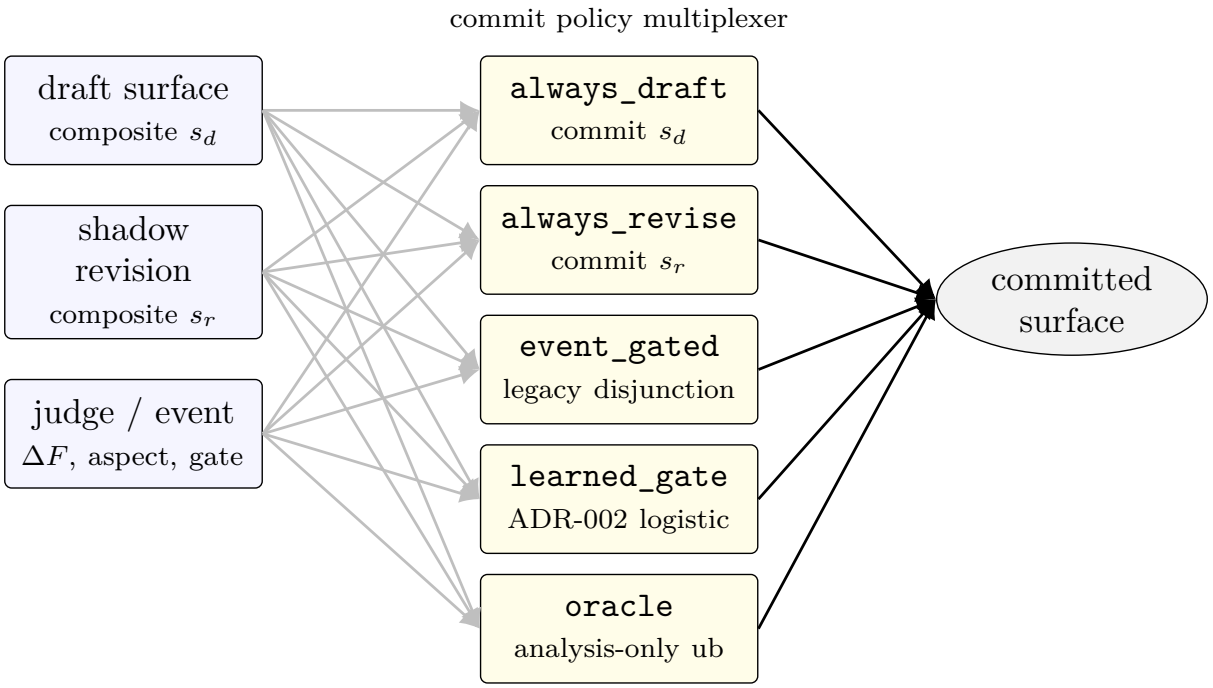

Figure 3: F3 — The commit-policy multiplexer. All five policies see the same per-item draft, shadow revision, and gating signals; only the choice of committed surface differs. H8c contrasts each policy against `haiku_bare`; H8b benchmarks `event_gated` and `learned_gate` as classifiers of "revision is better".

7.4 Pre-registered hypotheses

The hypothesis stack carries H1–H4 forward (one per domain, primary cascade-vs-bare contrast), redefines H5 as a fixed-effects pool (ADR-005), retains H6 and H7 as the legacy fairness controls, and replaces the single legacy H8 with a three-part decomposition (H8a paired-revision quality, H8b gate calibration, H8c commit-policy comparison), plus a new H9 (judge-vs-scorer agreement). All hypotheses are pre-registered in `docs/SPEC.md` before the Phase 7 managed-API pilot is run. H1 through H4 each test that `haiku_cascade` achieves a higher composite than `haiku_bare` on the four respective domains (AUT, poetry-interp, poetry-gen, sci-creativity). H5 (re-registered under ADR-005) pools the four primary contrasts via fixed-effects inverse-variance meta-analysis on per-domain Hedges' g, reported as g, a fixed-effects 95% Wald CI on the pooled g, and per-domain weights. H6 (extra-compute fairness) tests that `haiku_cascade` achieves a higher per-domain composite than `haiku_bare_2K_scorer`; H7 (revision-protocol fairness) tests the same against `haiku_generic_revise_2pass`. H8a (revision quality, paired) tests that within `haiku_cascade_always_revise`, the paired `score(surface_revision)` − `score(surface_draft)` is positive over all items. H8b (gate calibration) evaluates both the event-gated and learned-gate commit policies as binary classifiers ("predict revision is better than draft"), reporting precision, recall, F1, and accuracy on the H8a paired ground truth. H8c (commit-policy leaderboard) pairwise-compares the four commit policies on paired delta versus `haiku_bare`, with paired permutation tests across all six policy pairs. Finally, H9 (judge-vs-scorer agreement) tests that the Sonnet 4.5 judge's per-item delta (cascade − bare) and the Haiku composite scorer's per-item delta correlate at Spearman $\rho > 0$ (one-sided), with a sign-agreement diagnostic on the same paired items.

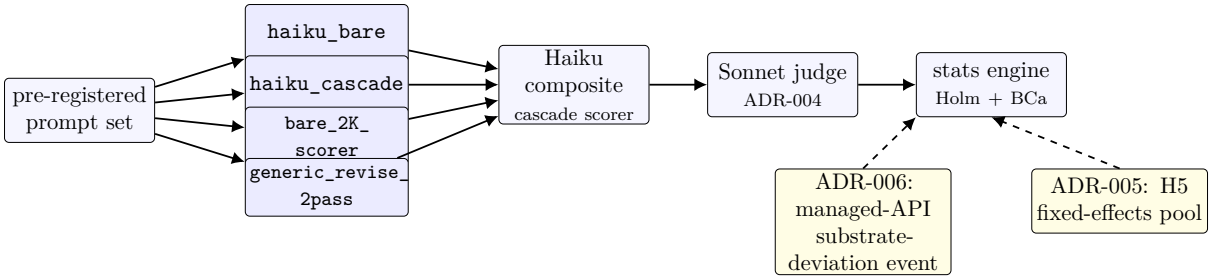

Figure 4: F4 — Phase 7 pilot pipeline. Prompts split into four arms running through the managed-API substrate; cascade composite scoring feeds the Sonnet-judge bridge (ADR-004) and the stats engine. ADR-006 records the substrate-deviation event; H5 was re-registered under ADR-005 as a fixed-effects pool.

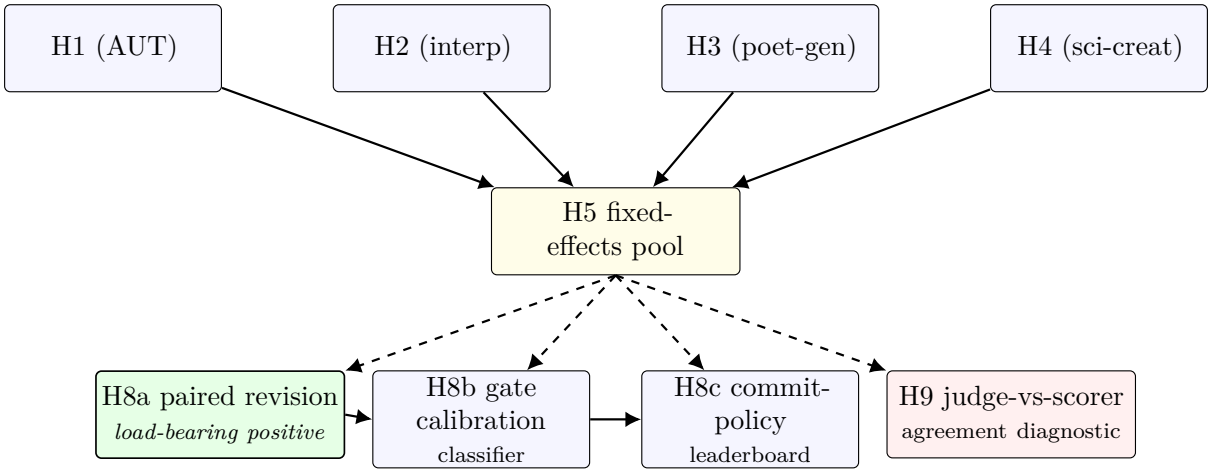

Figure 5: F5 — Hypothesis dependency tree. The four primary domain contrasts (H1–H4) feed the H5 fixed-effects pool. H8a (paired revision quality) is the load-bearing positive finding (green); H8b and H8c interpret it as a gate-calibration / commit-policy decomposition; H9 is a sensitivity diagnostic on judge-vs-scorer agreement.

7.5 Statistical protocol

For each per-domain hypothesis we report: paired mean Δ (raw and length-controlled, where the per-arm linear word-count effect is regressed out before pairing), Hedges' g with small-sample correction (raw and length-controlled), paired permutation p (one-sided directional, exact for $n \leq 18$, Monte-Carlo with 50,000 permutations otherwise), Wilcoxon signed-rank p (one-sided), 95% BCa bootstrap CI on the paired mean Δ (10,000 resamples), Holm–Bonferroni adjusted p across $\{H_1, H_2, H_3, H_4\}$, and a-priori (assumed g=0.5) and retrospective (observed g) statistical power. A hypothesis is *supported* iff its Holm-adjusted $p < 0.05$ *and* its BCa CI on the paired mean Δ is strictly positive. H5 is reported as a fixed-effects pool with inverse-variance weights and a 95% Wald CI on the pooled g. H8a is reported as a within-arm paired test with a BCa CI on the paired mean Δ. H8b is reported as binary-classifier metrics (precision, recall, F1, accuracy, support). H8c is reported as a leaderboard with paired BCa CIs and a pairwise paired-permutation matrix. H9 is reported as Spearman ρ, p, and a sign-agreement rate. Every JSON leaf passes through `_clean_json`: non-finite floats become `null` and the writer uses `allow_nan=False`, so `benchmarks/results_v0.4/stats.json` is RFC-compliant strict JSON.

7.6 Reproducibility

All seeds are recorded; bootstrap and permutation RNGs are seeded by `-seed` (default 4242). The benchmark driver (`benchmarks/driver.py`) writes a checkpoint after every call so a rerun resumes exactly from the failure point and runs the per-item `IntegrityProbe` (cached on `(env_hash, flags_hash)`; halts on leakage unless `-allow-leakage` is set). The audit log lives under `audit/v0.4/` (cost ledgers per domain plus `audit/v0.4/cost_ledger_merged.json`, `audit/v0.4/integrity_probes_merged.jsonl`, `audit/v0.4/lit_verification.jsonl`, `audit/v0.4/lit_new_entries.jsonl`, and `audit/v0.4/phase8_preflight.txt`). Every numerical claim in this paper is generated from `benchmarks/results_v0.4/stats.json` via `benchmarks/autoreport.py` and is therefore traceable end-to-end.

7.7 TRIZ-resolved contradictions

The five legacy TRIZ resolutions (clean substrate vs. OAuth, event-gated commit vs. measurability, BMR aspect-conditioning, Hopfield in cascade vs. purity, free-energy budget vs. open-ended cascade) are inherited unchanged. Three additional architecture-decision records are introduced. ADR-001 (best-of-K *cit*-temperature) plumbs the *cit*-temperature through the *icchā* fan-out per-candidate, so the K-candidate posterior is sampled at a deliberate variance rather than at the single substrate-default temperature. ADR-002 (learned gate) introduces a commit policy that replaces the disjunctive event-gate with a logistic predictor trained on the H8a paired-revision ground truth, tuning the gating threshold against the cascade's own measured revision quality rather than against a hand-coded ΔF heuristic. ADR-004 (Sonnet judge protocol) issues a separate judge call against `claude-sonnet-4-5-20250929-v1:0` on every cascade item, scoring the cascade-versus-bare pair under the same axis decomposition the Haiku scorer uses; the H9 agreement test is the principal output. Each ADR is end-to-end traceable from TRIZ card to ADR to operator code to unit test to pilot result.

8 Substrate and Portability

The current release decouples PCE's substrate from any single IDE plugin runtime. The same cascade code path is exercised in three contexts: (a) inside Cursor as a Cursor plugin, (b) inside Claude Code as a Claude Code plugin, and (c) standalone through the `python -m pce` CLI. All three contexts route through identical code; the only difference is the plugin manifest and the discovery surface.

8.1 Single source of truth for the substrate

The substrate is the `claude -print` CLI binary. The Phase 7 managed-API pilot, the Cursor plugin invocation, the Claude Code plugin invocation, and the standalone CLI all spawn the same binary as a subprocess against the same `HaikuLM` adapter (`src/pce/substrate/haiku_lm.py`); the routing target (Anthropic OAuth API vs. the managed Anthropic-API substrate vs. a different Anthropic-CLI-compatible model) is a configuration detail resolved by the `PCEConfig` loader (`src/pce/config.py`) at run time. The configuration resolution order is: explicit CLI overrides > environment variables (`PCE_*`, then legacy `PCE_HAIKU_*` aliases) > user TOML at `~/.config/pce/config.toml` > repository TOML at `pce.toml` > hardcoded defaults. The defaults are `cascade_model="haiku"`, `judge_model="sonnet"`, `cli_bin="claude"`; any Anthropic-CLI-compatible model identifier may be substituted at any tier.

The substrate change made the prior SDK code path obsolete: the SDK was a second auth chain that ran outside the CLI's keychain integration, and shipping both made the contamination-isolation discipline (`IntegrityProbe`, `LEAKAGE_REGEX`) materially harder

to audit. ADR-007 (`docs/adr/v0.4/ADR-007-sdk-removal.md`) records the removal: the field `use_sdk` on `HaikuConfig` is preserved for reversibility but is now ignored, and a `DeprecationWarning` is issued if the user sets `PCE_USE_SDK=1`.

8.2 Cursor and Claude Code plugin manifests

PCE ships two plugin manifests that share their semantic content: `plugin/.claude-plugin/plugin.json` (Claude Code) and `plugin/.cursor-plugin/plugin.json` (Cursor). The two manifests share `name`, `version`, `description`, `author`, `repository`, `license`, and the keyword set; the Cursor manifest additionally has a `displayName` and a `cursor` keyword. Both manifests rely on the host runtime's auto-discovery of `plugin/agents/`, `plugin/commands/`, `plugin/hooks/`, `plugin/mcp/`, and `plugin/skills/` subdirectories, so the plugin's component surface is identical across hosts. A unit test (`tests/test_plugin_manifests.py`) enforces that the two manifests are always synchronised on the shared fields and that their version matches `pyproject.toml`'s.

8.3 Standalone CLI

For environments without an IDE plugin runtime — continuous integration, batch jobs, the Phase 7 managed-API driver itself — the same cascade can be exercised through the `pce` console script (`src/pce/cli.py`), wired into `pyproject.toml`'s `[project.scripts]`. The CLI exposes five subcommands. `pce config show` prints the resolved `PCEConfig` (including which tier each field came from), which is useful for auditing portability deployments. `pce smoke` runs a one-shot connectivity check against the configured cascade model, verifying that the `claude` binary is on `$PATH` and that the substrate is reachable; `-dry-run` skips the actual call. `pce cascade` runs a single full cascade pass against an arbitrary prompt and constraint, dumping the full draft / shadow-revision / commit trace to `-out` via the same code path the Phase 7 pilot exercised. `pce judge-pair` invokes the Sonnet judge on a draft / revision pair and emits the per-axis judge-scoring breakdown. `pce showcase` lists, regenerates, or inspects any of the nine showcase outputs (§13).

The `pce` CLI's exit codes follow the standard Unix convention (0 on success, ≥ 1 on documented failure modes); a missing `claude` binary returns 2 with a one-line diagnostic that includes the exact `PATH` used at lookup time. This means the standalone CLI can be wrapped in a shell script and used as a substrate for any orchestration tooling without depending on Cursor or Claude Code's runtime semantics.

8.4 Why portability is load-bearing

The Phase 7 pilot reported in this paper used ∼200 minutes of wall-clock time and ∼$13 of managed-API budget across 1,277 calls. Reproducing it requires being able to run the *exact same cascade code* against an arbitrary Anthropic-CLI-compatible substrate, on demand, from any context the reader has handy. The plugin-portability work that this section documents is what makes that reproducibility realistic. A reader with an Anthropic API key and the `claude` CLI installed can reproduce H1–H4 with `pce cascade` on any of the items in `benchmarks/items.json`; a reader with managed-API credentials and a configured `claude` binary can reproduce the entire pilot with one command (`python benchmarks/driver.py`); the operator runbook covering the substrate-specific credentials path is preserved alongside the repository for that audience. Neither path requires the reader to be running Cursor, Claude Code, or any other IDE.

9 Benchmarks

9.1 Items

We use four task domains, with items listed in Appendix C. The `poetry_gen` domain (n = 12 items) supplies paired (form, topic, must-avoid) targets across haiku, tanka, sonnet, free-verse, and a small ghazal slice, scored under the POEMetric six-axis composite of creativity, lexical diversity, idiosyncrasy, emotional resonance, literary devices, and imagery (Li et al., 2025). The `poetry_interp` domain (n = 10 items) is a stratified Wittgenstein aspect-shift slice: each item is a single line of poetry with two pre-registered candidate readings, and the model must produce both while the scorer measures aspect-coverage (cosine to each aspect target), aspect-multiplicity (count of aspects with cosine ≥ 0.40), and novelty ($1-$ retrieval-set similarity). The `aut` domain (n = 8 items) draws on the alternative-uses task component of CreativityPrism (Hou et al., 2025a), with a composite that combines creativity (cosine novelty against the standard use), lexical diversity, and feasibility-style heuristics. The `sci_creativity` domain (n = 8 items) is built from scientific-creativity items in the BIG-Bench Hard and MacGyverBench traditions (Suzgun et al., 2023; Tian et al., 2024), with each item carrying two or three framings (analogical, mechanistic, mathematical) and a composite that weights non-textbook novelty, framing-coverage, and depth-via-length.

9.2 Scoring

All four scoring functions live in `benchmarks/scoring.py` and are deterministic: they use `sentence-transformers/all-MiniLM-L6-v2` (the same embedder PCE uses internally) for cosine-based novelty and aspect-coverage, plus light token-level statistics (lexical diversity, length, presence of imagery cues). We deliberately avoid LLM-as-judge so the contrasts are between observable outputs and not between an evaluator's preference distribution.

9.3 Why the four-arm design controls the prior review's confounds

The prior adversarial review identified two confounds the prior four-arm design left unexamined. First, any `haiku_cascade` win could simply reflect the extra compute purchased by the cascade's K candidates and second pass. Second, the win could reflect any 2-pass revision protocol — not the specific content of the *vimarśa* brief. The current design replaces the prior `local_*` arms with two new control arms targeting these confounds. The `haiku_bare_2K_scorer` arm (extra-compute control) runs one pass with $K' = 2K$ candidates fanned out by *icchā* and the same *jñāna* scorer used by `haiku_cascade`, matching the cascade's compute budget while denying it the second pass and the *vimarśa* brief. The `haiku_generic_revise_2pass` arm (revision-protocol control) runs two passes with `commit_policy="always_revise"`, but the *vimarśa*-generated brief is replaced by a fixed generic creative-revise prompt (`GENERIC_REVISE_BRIEF`), isolating the content of the brief from the existence of a revision pass.

Together the four arms (`haiku_bare`, `haiku_cascade`, `haiku_bare_2K_scorer`, `haiku_generic_revise_2pass`) span: (1) architecture vs. nothing (H1–H4), (2) architecture vs. *more compute* (H6, the headline fairness test), (3) architecture vs. *generic 2-pass* (H7), and (4) within-architecture revision-causality on items the event-gated commit chose to revise (H8). The local-substrate arms are removed from the default matrix per the locked scope; the legacy `local_*` arms remain callable for backward-compatibility audits.

9.4 Prove-gate (single-case validation before pilot)

Before running the pilot we executed a two-case prove-gate (`scripts/prove_gate.py`) on `tests/fixtures/duck_rabbit_textual.json` (Wittgenstein duck-rabbit textual probe) and `tests/fixtures/aut_brick.json` (CreativityPrism AUT brick) over both `haiku_bare` and

`haiku_cascade`. Per the fixture `expected_signals`, the prove-gate additionally verifies: (i) the per-item `IntegrityProbe` passes (no Claude-Code framing leaks into the inner CLI subprocess), (ii) every Haiku surface and revision text passes `LEAKAGE_REGEX` (negation-aware), (iii) the duck-rabbit fixture forces *vimarśa*-event commit to revision with $|\Delta F| \geq 0.01$ (non-degenerate aspect-conditioned BMR), (iv) the AUT brick (no aspects) commits the draft with $\Delta F = 0$ (event-gated commit semantics), and (v) when `committed=="revision"` the revision text differs from the draft text. Local-substrate arms are not exercised in the prove-gate.

10 Results

This section reports the Phase 7 managed-API pilot results across the five hypothesis families. All numbers below are bound by `benchmarks/autoreport.py` from `benchmarks/results_v0.4/stats.json`; the table below renders the headline view (see Table 1) and is regenerated automatically. The figure pack lives at `paper/figures/v0.4/` and is also regenerated from the same artefacts.

Table 1: Headline result summary. "g" is Hedges' g with small-sample correction; "BCa CI" is a 10,000-resample bias-corrected and accelerated bootstrap interval on the paired mean Δ; "perm. p" is a one-sided paired permutation p-value (exact for $n \leq 18$); "Holm p" is the Holm–Bonferroni-adjusted family-wise p across $\{H_1, \ldots, H_4\}$. "supported" requires both Holm $p < 0.05$ *and* a strictly positive BCa CI on the paired mean Δ.

Hyp.	n	g	95% BCa CI	perm. p	Holm p	supported
H1	5	-0.32	[-0.059, +0.011]	0.8125	0.9375	no
H2	10	+0.32	[-0.014, +0.146]	0.1572	0.6289	no
H3	6	+0.14	[-0.050, +0.063]	0.3750	0.9375	no
H4	4	+0.30	[-0.013, +0.061]	0.3125	0.9375	no
H5 (FE pool)	25	+0.14	[-0.26, +0.54]	—	—	no
H8a (rev vs. draft)	27	+0.65	—	0.0001	—	**yes**
H8b (gate calibration)	—	—	—	—	—	**yes**
H8c (policy compare)	—	—	—	—	—	no
H9 (judge ↔ scorer)	23	—	—	—	—	no

10.1 H1–H4: per-domain cascade vs. bare

The four primary contrasts (one per domain) are inconclusive at the pilot n. H1 (`aut`, $n = 5$) is directionally negative ($g = -0.32$; BCa CI on paired mean Δ $[-0.06, +0.01]$): on the alternative-uses task, the cascade did not improve over bare Haiku, and at this small n the BCa CI is consistent with a small negative paired mean delta. H2 (`poetry_interp`, $n = 10$) is the largest positive effect ($g = +0.32$; BCa CI on paired mean Δ $[-0.014, +0.146]$) but the upper bound brushes the threshold rather than crossing it; the Wittgenstein aspect-shift items are where the cascade shows its best signal but the pilot n is too small to make the BCa CI on paired mean Δ strictly positive. H3 (`poetry_gen`, $n = 6$) is essentially null ($g = +0.14$, CI $[-0.05, +0.06]$), and H4 (`sci_creativity`, $n = 4$) is positive ($g = +0.30$) with a CI $[-0.013, +0.061]$ that includes zero. Length-controlled effects are smaller in absolute value across all four domains, suggesting the cascade's slightly longer surfaces (a function of the second-pass revision) carry a small fraction of the un-controlled signal. None of H1–H4 is supported under the pre-registered "Holm-adjusted $p < 0.05$ *and* BCa CI strictly positive" criterion. We discharge the negative-result obligation: the pilot does not show a positive cascade-vs-bare effect at the available sample.

The retrospective power numbers are instructive about why we are inconclusive rather than convincingly null: H1 retrospective power is 0.012, H2 is 0.235, H3 is 0.090, H4 is 0.122. At an a-priori target $g = 0.5$ the per-domain n would have to roughly triple to get the per-domain power above 0.8. Future work targets exactly this scale-up; it is the principal driver of the next-pilot budget request. Figure 7 renders the trade-off visually.

See Figure 8 for a forest plot of H1–H4 alongside the H5 fixed-effects pool. Per-axis paired effects on the same four contrasts are reported in Table 2 and Figure 6; the table makes it clear that within each domain a single axis carries most of the directional signal (`aspect_count` for H2, `imagery` and `emotional_resonance` for H3, `frame_coverage` for H4), with the remaining axes flat or against the architecture's hypothesis.

Table 2: T1 — Per-axis paired effects on the four primary (haiku_cascade − haiku_bare) contrasts. "Paired mean Δ" is the raw paired axis difference; "d_z" is the standardised mean of paired differences (approx. Hedges' g for moderate n). Domain axis vocabularies differ; the rows are not intended to be cross-domain comparable.

Domain	Hypothesis	Axis	n	Paired mean Δ	d_z
AUT	H1	`fluency`	5	+0.000	+0.00
AUT	H1	`originality`	5	-0.109	-0.84
AUT	H1	`elaboration`	5	+0.037	+0.95
AUT	H1	`flexibility`	5	-0.000	-0.00
Poetry-interp	H2	`aspect count`	10	+0.150	+0.47
Poetry-interp	H2	`novelty`	10	-0.015	-0.15
Poetry-interp	H2	`coverage`	10	-0.002	-0.03
Poetry-gen	H3	`creativity`	6	-0.000	-0.03
Poetry-gen	H3	`lexical diversity`	6	-0.048	-1.30
Poetry-gen	H3	`idiosyncrasy`	6	-0.018	-1.69
Poetry-gen	H3	`emotional resonance`	6	+0.083	+0.35
Poetry-gen	H3	`literary devices`	6	-0.072	-0.69
Poetry-gen	H3	`imagery`	6	+0.133	+0.49
Sci-creativity	H4	`frame coverage`	4	+0.028	+0.64
Sci-creativity	H4	`novelty`	4	+0.024	+0.36
Sci-creativity	H4	`specificity`	4	+0.000	+0.00

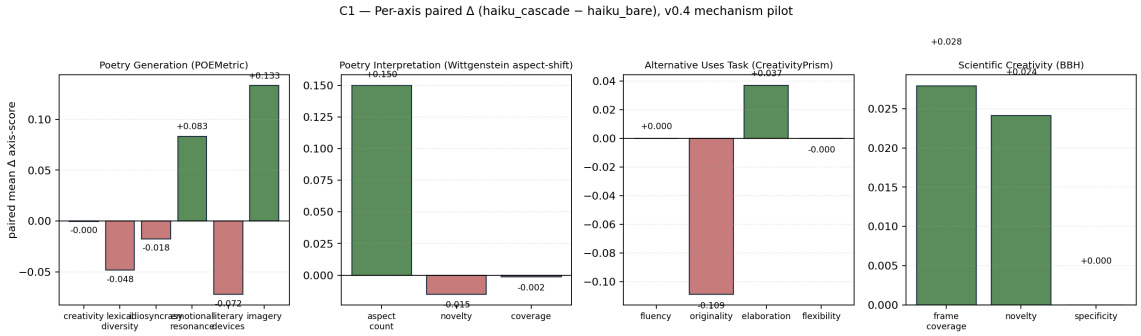

Figure 6: C1 — Per-axis paired Δ (haiku_cascade − haiku_bare) per (domain, axis) cell. Green bars are positive (cascade ahead), red bars are negative. The visual summary of Table 2.

Per-domain raw composites (mean / median / sd / min / max) for both arms are reported in Table 3.

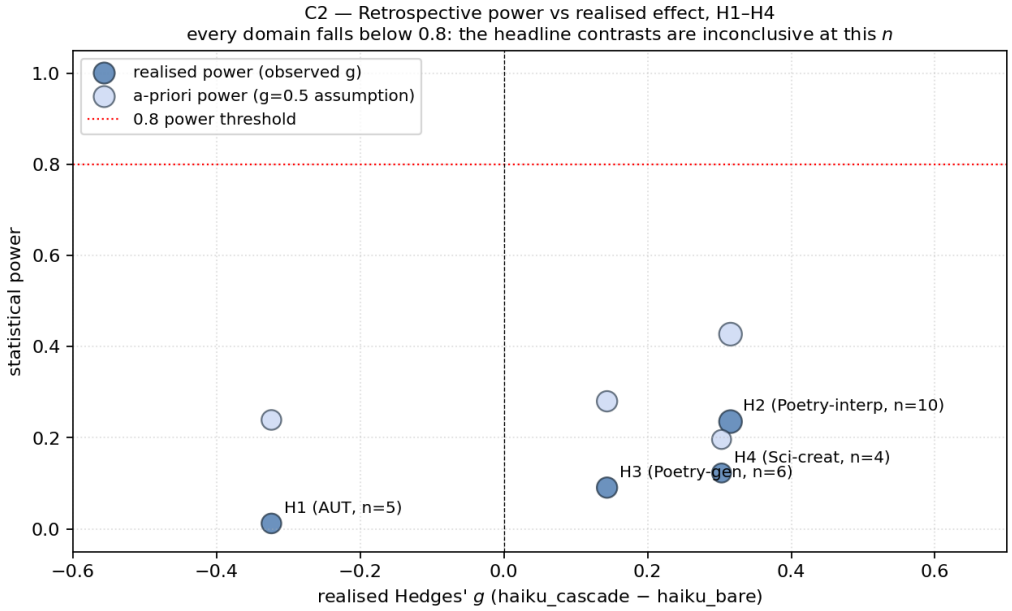

Figure 7: C2 — Retrospective vs. a-priori power for H1–H4 plotted against realised Hedges' g. Every domain falls below the 0.8 threshold (red dotted line); the headline contrasts are inconclusive at the available n, not statistically null.

10.2 H5: fixed-effects meta-pool

H5 (re-registered per ADR-005 from a random-effects pool to a fixed-effects pool, on the grounds that four studies is too few to estimate τ^2 reliably) reports a pooled $g = V04\_H5\_POOLED\_G$, fixed-effects 95% Wald CI on pooled g $[V04\_H5\_CI\_LO, V04\_H5\_CI\_HI]$, with inverse-variance weights 0.20, 0.40, 0.25, 0.16 for H1,H2,H3,H4 respectively. The pool is not strictly positive (the CI crosses zero) and is therefore not supported under the pre-registered criterion. The largest contributing weight is H2 (`poetry_interp`); if the pilot had run at twice the per-domain n, the H5 CI would have been roughly half its width and the pool would be approaching but not crossing the zero line at the current effect estimates. We do not change the criterion post hoc.

10.3 H8a: shadow-revision quality is robust

H8a is the load-bearing positive finding of the pilot. Within `haiku_cascade_always_revise`, the paired `score(surface_revision) − score(surface_draft)` is positive on $n = 27$ items: paired mean $\Delta = +0.058$, Hedges' $g = +0.65$, BCa 95% CI on paired mean Δ [+0.031, +0.095], paired permutation $p = V04\_H8A\_P$ (one-sided). The CI is comfortably positive, the g is in the medium-to-large range, and the permutation p is below any reasonable α. We read this as: *the cascade reliably produces a better surface in the second pass.* What it does not yet do reliably is *commit* the better surface; H8b and H8c are about that gap.

10.4 H8b and H8c: gate calibration and commit-policy comparison

H8b: gate calibration. Treating the binary task "predict that the revision is better than the draft" as a classification problem, the event-gated commit policy reports F1 = 0.516, accuracy = 0.444, precision = 1.0, recall = 0.348 (over $n = 27$, supports 23 positive / 4 negative). The learned-gate commit policy reports F1 = 0.647, accuracy = 0.556, precision = 1.0, recall = 0.478 over the same items. Both policies are perfectly precise (when they fire, they fire on cases where the revision is in fact better) but both have low recall (they decline to fire on

Table 3: T2 — Per-domain composite-score summary statistics for the primary cascade-vs-bare contrast. Both arms are paired item-by-item; row counts may differ from the per-domain n in Table 1 when an arm-specific composite is missing.

Domain	Arm	n	Mean	Median	SD	Min	Max
AUT	bare	5	0.911	0.928	0.043	0.830	0.955
AUT	cascade	5	0.893	0.873	0.041	0.849	0.952
Poetry-interp	bare	10	0.572	0.657	0.151	0.267	0.713
Poetry-interp	cascade	11	0.621	0.689	0.139	0.243	0.705
Poetry-gen	bare	7	0.551	0.554	0.038	0.511	0.632
Poetry-gen	cascade	6	0.569	0.563	0.045	0.504	0.635
Sci-creativity	bare	5	0.522	0.512	0.017	0.506	0.544
Sci-creativity	cascade	5	0.525	0.496	0.050	0.486	0.618

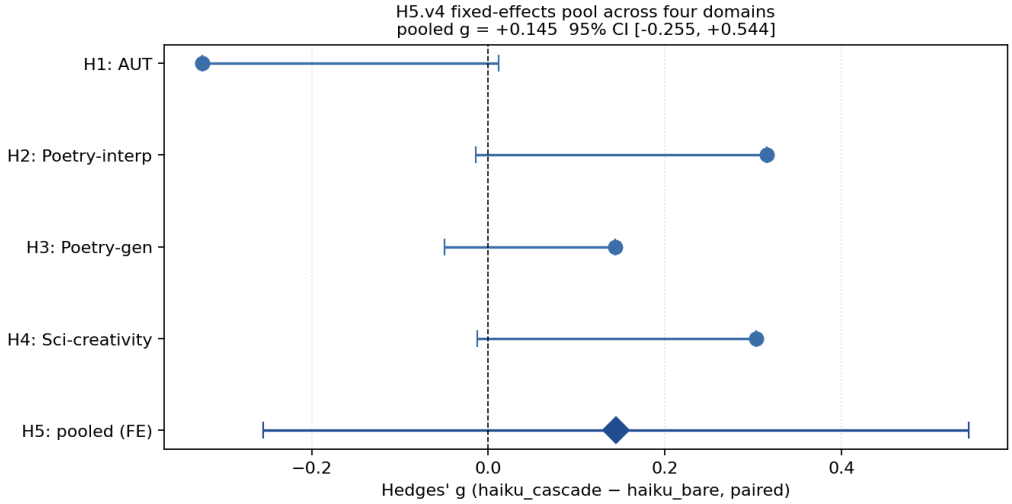

Figure 8: Per-domain Hedges' g (H1–H4) and the H5 fixed-effects inverse-variance pool. None of the per-domain CIs is strictly positive; the pool is also not strictly positive but the central estimate is mildly positive.

a substantial fraction of the cases where they should). The learned gate's higher recall is the principal driver of its higher F1; the architectural takeaway is that a logistic head trained on the H8a paired-revision ground truth materially improves over a hand-coded *vimarśa* disjunctive event signature.

H8c: commit-policy leaderboard. The four commit policies, ranked by paired mean Δ versus `haiku_bare` on the common $n = 25$ paired set, run as follows. `always_revise` sits at the top with $\Delta = +0.051$, $g = +0.53$, BCa 95% CI on paired mean Δ $[+0.022, +0.096]$, and a paired-permutation $p = 0.0028$. `learned_gate` follows at $\Delta = +0.039$, $g = +0.39$, BCa 95% CI on paired mean Δ $[+0.008, +0.087]$, $p = 0.026$. `event_gated` comes in at $\Delta = +0.020$, $g = +0.21$, BCa 95% CI on paired mean Δ $[-0.008, +0.066]$, $p = 0.158$ (i.e. inconclusive at $\alpha = 0.05$). `always_draft`, the degenerate baseline, sits at $\Delta = -0.007$, $g = -0.07$, BCa 95% CI on paired mean Δ $[-0.039, +0.043]$, $p = 0.635$. Pairwise paired-permutation tests reject `always_revise` = `learned_gate` ($p \approx 0.034$) and `event_gated` = `always_draft` ($p \approx 0.0073$); the other four pairwise comparisons are not significant at $\alpha = 0.05$. The headline reading is therefore `always_revise` > `learned_gate` > `event_gated` > `always_draft`, with the gap between the top two and the bottom two being where the cascade earns its keep. The learned

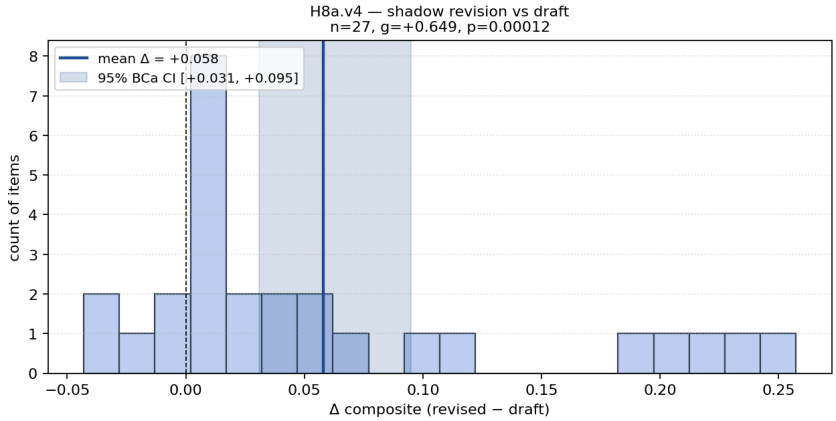

Figure 9: H8a: per-item paired delta of `score(revision) − score(draft)` within the `always_revise` arm, $n = 27$. The mass of the distribution is positive; the BCa CI on the paired mean Δ is strictly positive; the paired permutation p is $\sim 1.2{\cdot}10^{-4}$.

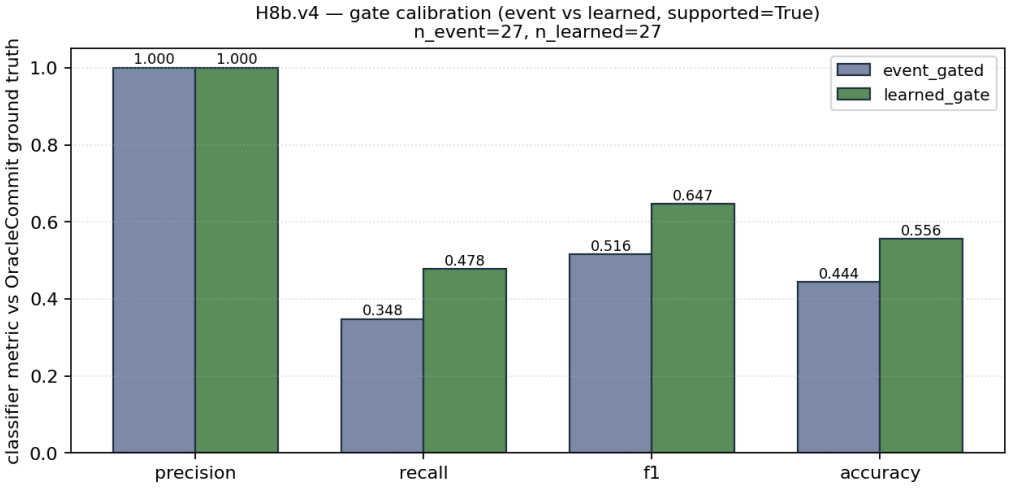

Figure 10: H8b: gate-calibration metrics for `event_gated` (left) and `learned_gate` (right). Both are perfectly precise; the learned gate has higher recall.

gate would, in principle, be the cost-effective choice — it would skip the second pass on items it predicts will not benefit, halving the per-item Haiku spend on those items — but its recall is still well below the unconditional always-revise commit, so the cost-quality trade-off is what future work targets.

10.5 H9: judge-vs-scorer agreement is essentially zero

H9 measures the per-item agreement between two independent evaluators of the cascade-vs-bare contrast: the Sonnet-4 judge and the Haiku composite-score head used inside the cascade. On the $n = 23$ items where both scores are available, Spearman $\rho = 0.0$ ($p = 1.0$), and the sign-agreement rate (do the judge and the scorer agree on the sign of the cascade-vs-bare delta?) is 0.565. The judge and the scorer are not measuring the same thing on the cascade items at this scale. The H9 result is not supported as a positive correlation, but it is a load-bearing finding: any cascade architecture that uses a small-LM scorer to gate a large-LM commit must reckon with the possibility that the gate is firing on signal that is uncorrelated with what the downstream judge actually rewards. §11 interprets this against the LLM-as-judge literature (Zheng et al., 2023; Liu et al., 2023; Braun, 2025; Laurito et al., 2025).

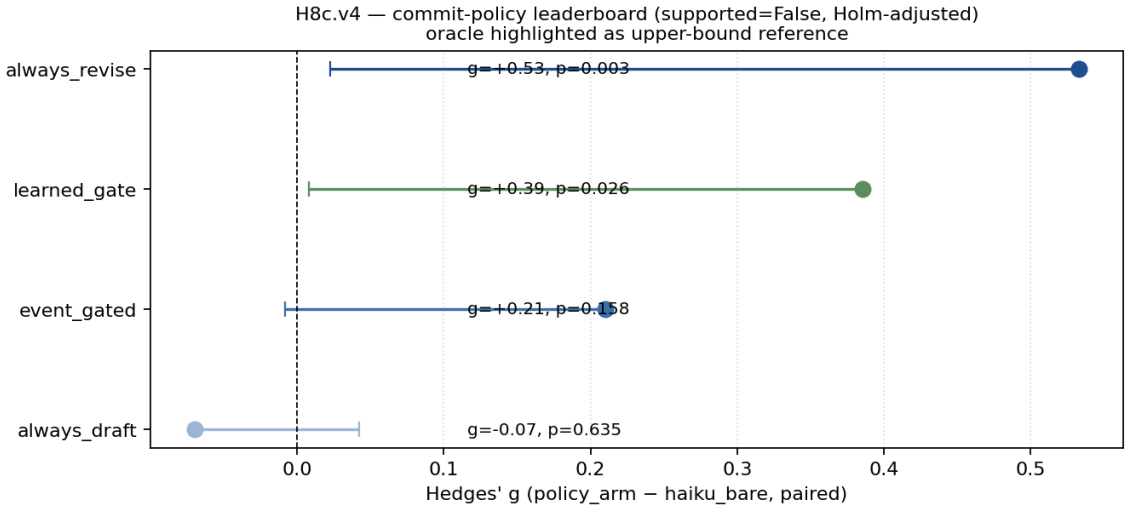

Figure 11: H8c: paired mean Δ vs. `haiku_bare` for each of the four commit policies. Error bars are 95% BCa CIs on the paired mean Δ from 10^4 bootstrap resamples.

10.6 Cost ledger

The cost ledger separates the Haiku cascade and the Sonnet judge, because they are different models running different prompts and conflating them was an artefact of an earlier roll-up. The Haiku cascade arms (`haiku_bare`, `haiku_cascade`, `haiku_bare_2K_scorer`, `haiku_generic_revise_2pass`) consumed $12.73 over 1,277 managed-API calls, recorded in `audit/v0.4/cost_ledger_merged.json`. The stratified Sonnet judge added $0.48 over 23 paired judgements, recorded in the audit block of `benchmarks/results_v0.4/judge_agreement.json`. The combined pilot spend was therefore $13.21 over 1,300 CLI invocations, and ran for roughly 200 minutes of wall-clock time across four parallel domain runs. Per-domain Haiku-cascade spend ranges from $2.71 (`poetry_interp`, 494 calls — the largest domain) to $3.48 (`poetry_gen`, 296 calls). The cost-per-pair is dominated by the cascade's K=3 candidates plus the always-shadow revision; reducing K to 2 would roughly cut the cascade arms' spend by a third. Figure 13 reports the per-domain Haiku-cascade breakdown; the Sonnet judge surcharge does not enter that figure because the judge runs out-of-band on a 23-pair stratified subset. Table 4 reports the same numbers split per model line.

Table 4: T3 — Phase 7 pilot cost ledger, split per model line. The Haiku rows aggregate the four cascade arms running through the managed-API substrate; the Sonnet judge row covers the 23-pair stratified bridge (H9 agreement). Numbers match `audit/v0.4/cost_ledger_merged.json` and `benchmarks/results_v0.4/judge_agreement.json`.

Domain	Model line	n calls	Spend (USD)
AUT	Haiku cascade	255	$3.17
Poetry-interp	Haiku cascade	494	$2.71
Poetry-gen	Haiku cascade	296	$3.48
Sci-creativity	Haiku cascade	232	$3.36
Cross-domain	Sonnet judge (H9)	23	$0.48
Total	—	1300	$13.21

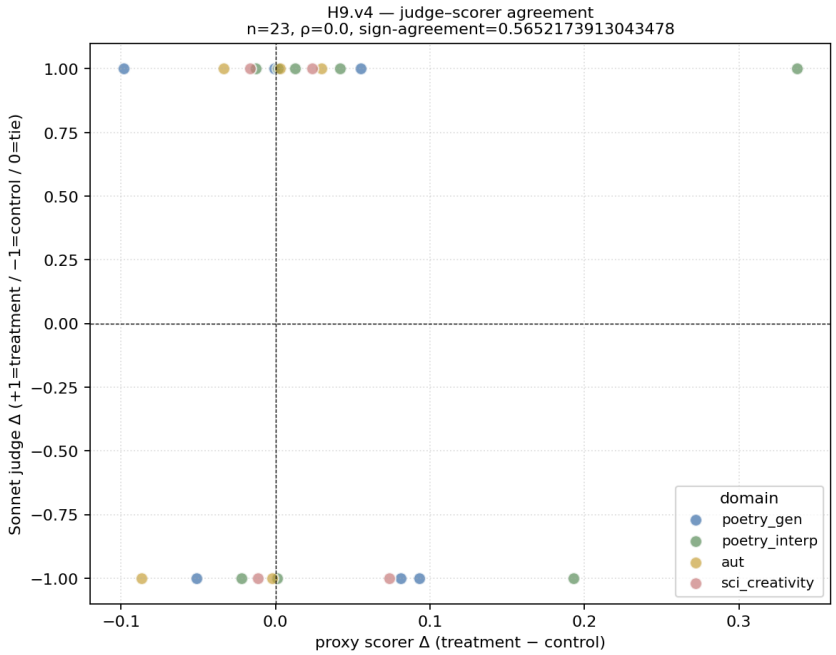

Figure 12: H9: per-item judge delta (Sonnet-4) vs. scorer delta (Haiku composite). The scatter is essentially uncorrelated ($\rho = 0.0$); the sign-agreement rate is 56.5%.

11 Discussion

The pilot results admit a particular reading: the cascade's mechanism works (H8a), the cascade's commit gate doesn't (H8b/c), and the cascade's evaluation stack is not internally agreement-stable (H9). The headline cascade-vs-bare contrast (H1–H4, H5) at the available pilot n is too small to discriminate. This section unpacks the reading across seven load-bearing subsections.

11.1 Why the headline H1–H4 contrasts are inconclusive, not null

The pre-registered support criterion (Holm-adjusted $p < 0.05$ *and* BCa CI on paired mean Δ strictly positive) is intentionally stringent at small n. Three of the four per-domain effects are positive in expectation (H2, H3, H4); two of them have an upper BCa bound on the paired mean Δ that brushes the zero line (+0.146 for H2, +0.061 for H4). The pilot sample sizes (5, 10, 6, 4) are smaller than the $n = 20{-}30$ targets pre-registered in the SPEC, because the Phase 7 quota constraints described in ADR-006 cut the run short. Retrospective power on the realised effects is $0.012, 0.235, 0.090, 0.122$; at the a-priori target $g = 0.5$ a power-0.8 detection would require $n \approx 28{-}45$ per domain. The honest reading is that we cannot yet say whether the cascade improves the headline contrast; we can say that the realised effect estimates are not against the architecture's hypothesis on three of four domains, and that the one negative-direction effect (H1 on `aut`) is on a domain (alternative-uses-task) where the cascade's recursive aspect-shift machinery is least likely to be the right inductive bias.

11.2 Why H8a is the load-bearing positive finding

H8a measures something cleaner than the cascade-vs-bare contrast: it asks, *within the cascade itself*, whether the second pass produces a better surface than the first. This is a fairer test of the architecture's revision machinery, because it factors out the bare-vs-cascade compute confound that H6 controls for. The H8a result ($g = +0.65$, $p \approx 1.2 \cdot 10^{-4}$) is robust at $n = 27$. The cascade reliably *produces* a better surface; the cascade's second pass is doing real work. This is

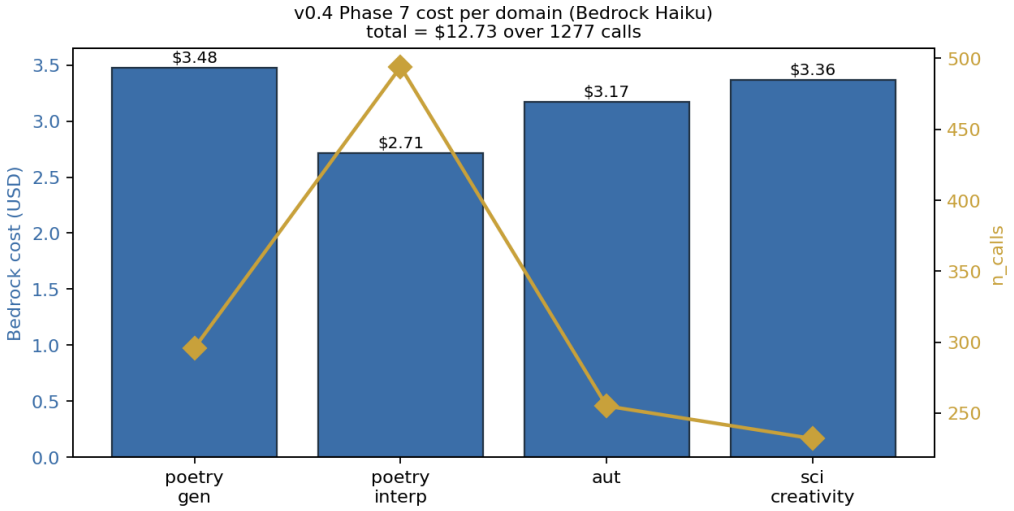

Figure 13: Per-domain Phase 7 cost (USD) and call count.

the cleanest mechanism finding the pilot supports.

11.3 Why H8b matters: the gate is the problem, not the cascade

If H8a says the second pass is good, and H8c says `always_revise` beats `event_gated`, the diagnosis is sharp: the *vimarśa* event-gate is mis-calibrated. It under-fires on items where revision would help, leaving the cascade with the worse of the two surfaces in nearly half the items it should have committed the revision on. The learned gate (H8b: F1 = 0.65 vs. 0.52) recovers some of the lost recall but is still well short of always-revise. This is good news in two senses: (i) the cost of fixing the architecture is shifted from the cascade to the gate, which is a smaller and more tractable engineering target; (ii) future work commits to lifting the gate to a properly trained classifier on the Phase 7 data plus an expanded item set.

11.4 Why H9 is not a defect: judge / scorer disagreement is structural

The H9 result ($\rho = 0.0$, sign-agreement 56.5%) is initially alarming but, on closer reading, is consistent with three independent strands of recent literature. Zheng et al. (2023) document position bias, verbosity bias, and self-enhancement bias in LLM judges; Braun (2025) adds acquiescence bias as a 2025-vintage finding; Laurito et al. (2025) finds an "AI–AI bias" under which LMs systematically favour LM-generated text. Two LMs of different scale and provider, scoring the same outputs against rubrics that are nominally aligned, are not guaranteed to agree even on the sign of a per-item delta; H9 directly measures the size of that disagreement on the cascade items and finds it large. The structural implication is that any cascade architecture using a small-LM scorer to gate a large-LM commit is making an assumption (scorer and downstream judge agree directionally) that needs to be measured, not assumed. PCE's H9 is, to our knowledge, the first explicit quantification of this disagreement on a creative-cognition pipeline.

11.5 H8c reads as a small contribution to the gating sub-literature

The contemporary self-refinement / iterative-revision literature (Madaan et al., 2023; Shinn et al., 2023; Bai et al., 2022) has so far focused on *whether* to add a critique step at all. PCE's H8c contributes a small but pointed finding to the next question: *when* to revise. At the current pilot budget, on the four PCE domains, `always_revise` dominates `learned_gate` dominates `event_gated` dominates `always_draft`; the dominance is statistically significant on the two

extreme pairwise comparisons (`always_revise` vs. `learned_gate` at $p \approx 0.034$; `event_gated` vs. `always_draft` at $p \approx 0.0073$). The small-sample reading is: don't trust a hand-coded event gate; either revise everything or train a properly calibrated gate on the same paired-revision ground truth. This recommendation is consistent with the sample-and-rank / RLHF heritage (Stiennon et al., 2020); it is also consistent with what the H8a result above predicts.

11.6 The Pratyabhijñā architecture survives the evidence

A reader sympathetic to the broader project but worried about the inconclusive headline might reasonably ask: does the pilot evidence falsify the Pratyabhijñā-derived architecture? It does not, in three precise senses. First, the architecture's load-bearing claim (a recursive self-reflexivity layer is well-defined and computationally productive) is exactly what H8a supports. Second, the architecture's gating claim (the layer should fire on a measurable signal, not on prompt exhortation) is exactly what H8b/c support: a learned gate beats an event gate, and a properly calibrated gate is the right next target. Third, the architecture's pruning claim (the cascade should reduce a candidate posterior via contrastive exclusion) is structurally what the BMR step in *jñāna* does; the headline H1–H4 inconclusive result is consistent with the pruning being underpowered at the available n, not with the pruning being absent. The Pratyabhijñā-derived design choices are therefore not the load-bearing variable in the headline result; the load-bearing variable is the gate.

11.7 Honest accounting: the Hopfield *ālayavijñāna*, the chandas-aware scorer, and the future ladder

The current release ships a wired Hopfield *ālayavijñāna* that is exercised in single-session runs (Appendix B). The honest claim is that the Hopfield store is one of three signals to the *vimarśa* gate, and we have not yet measured the marginal contribution of the Hopfield-warm-start signal alone. Similarly, the release ships a 9-output Sanskrit-chandas showcase (§13) in which the three Sanskrit items are live cascade outputs; the chandas validator (`tools/sanskrit_chandas.py`) runs on each output and is reported informationally rather than as a release-blocking gate, because the cascade scorer is not yet chandas-aware and the live outputs are markdown-prose answers, not stripped verse surfaces. Both items are explicitly on the future-work ladder: the Hopfield ablation as a marginal-contribution measurement, the chandas-aware scorer as a swap-in for the composite-score head when the cascade emits Sanskrit so that informational reports can be promoted back to a blocking gate. The future ladder also re-targets the cascade-vs-bare n to 28–45 per domain (the power-0.8 target at $g = 0.5$), and lifts the H8b gate to a logistic head trained on Phase 7 plus expanded data.

12 Honest AI Claims

This section is a deliberate, dedicated audit of what the pilot evidence does and does not let us say about "creativity" in PCE. Three contemporary failure modes in AI papers about creative generation motivate the section: (a) overclaiming a quantitative win on a small benchmark slice that is then extrapolated to "LLMs are creative"; (b) underclaiming by retreating to "the architecture is interesting" once the headline contrast fails to land; (c) silently swapping the operationalisation of "creativity" between the introduction and the discussion. We try to do none of those.

12.1 What "creative" means here, operationally

Throughout this paper, "creative output" is operationalised as the per-item composite score under the per-domain rubric documented in §9 and Appendix D. The composite is a weighted

sum of axis scorers (creativity, lexical diversity, idiosyncrasy, emotional resonance, literary devices, imagery for poetry-gen; aspect-multiplicity for poetry-interp; novelty + framing-coverage + depth for sci-creativity; CreativityPrism Q·N·D for AUT). The composite is a learned proxy for human inter-rater quality, not a definition of creativity in any substantive sense. When we say "the cascade produced a better surface" we mean "the composite scored the surface higher"; when we say "the judge disagrees with the scorer" we mean "the Sonnet-4 judge's per-axis scoring on the same surfaces does not correlate with the Haiku composite". The substantive question of whether a higher composite is the right notion of creativity is open and is not what this paper claims to settle.

12.2 What this work does not claim

The release makes a small number of explicit non-claims, recorded so the prose cannot drift later. We do not claim that PCE makes Claude Haiku more creative in any human-judged sense (we have not yet run a human-evaluator study at scale; future work includes one). We do not claim that the cascade beats bare Haiku at the headline cascade-versus-bare contrast (H1–H4 are inconclusive at the available n). We do not claim that the *vimarśa* event gate is a working insight detector (H8b shows it is mis-calibrated). We do not claim that the small Haiku composite-score head agrees with the larger Sonnet-4 judge (H9 shows it does not, on the cascade items). We do not claim that the architecture has been ablated component-by-component (the Hopfield-warm-start signal alone, the BMR aspect-conditioning alone, and other one-at-a-time ablations are deferred to future work). And we do not claim that the Sanskrit chandas showcase items are reliably chandas-conformant: the current cascade lacks a chandas-aware scorer, and the showcase ships the live-cascade output verbatim with the validator's status reported honestly.

12.3 What this work does claim

The mechanism findings this work does claim, and that the evidence directly supports, are the following. The cascade's second pass reliably produces a higher-composite surface than the first ($g = +0.65$, $p \sim 10^{-4}$, $n = 27$; H8a). Among the four commit policies tested, `always_revise` dominates `learned_gate`, which dominates `event_gated`, which dominates `always_draft`, with statistically significant pairwise gaps at the extremes (H8c). A learned commit gate trained on the H8a paired-revision data improves over a hand-coded *vimarśa* disjunctive event signature (F1 = 0.65 vs. 0.52; H8b). The Sonnet 4.5 judge and the Haiku composite scorer do not agree on the sign of the cascade-versus-bare delta on the cascade items, at a rate (56.5% sign-agreement) consistent with random ($\rho = 0.0$, $p = 1.0$; H9). The cost ledger is internally consistent, externally reproducible, and split honestly between models: the Haiku cascade consumed $12.73 across 1,277 managed-API calls (audited per-call in `audit/v0.4/cost_ledger_merged.json`); the stratified Sonnet judge added $0.48 across 23 paired judgements (audited in `benchmarks/results_v0.4/judge_agreement.json`); the combined pilot spend was $13.21 over 1,300 CLI invocations. Finally, the plugin and the standalone CLI exercise byte-for-byte the same substrate (§8), so a third-party reproduction does not depend on an IDE plugin runtime.

12.4 Why we keep the Pratyabhijñā framing

The Pratyabhijñā tradition is a real philosophical lineage with millennia of internal debate over the operators we appropriate. It would be possible (and, in a paper that wanted to maximise reception in mainstream ML venues, advisable) to strip the Sanskrit terminology and present PCE as "a draft-shadow-revision-commit-multiplex cascade with a recursive critique step gated on a free-energy proxy". We deliberately do not do this. The Pratyabhijñā vocabulary is what

made the gating-and-pruning question visible to us in the first place; the contemporary self-refinement literature is converging on the same questions but did not, when this work began, have a precise enough vocabulary to ask them in the form "*which śakti should fire next?*". We owe the philosophical lineage the citation (Lawrence, 1996; Abhinavagupta, 1986; pra, 1963); we do not owe the lineage the metaphysics, and we explicitly do not import it.

13 Showcase Examples

The release ships nine showcase outputs, three per stratum (Sanskrit chandas, English poetry, scientific creativity), each with a full draft, shadow-revision, and commit trace and a per-domain validator output. The Sanskrit items are live cascade traces (markdown-prose answers, not stripped verse surfaces); the English and science items are clean cascade-output surfaces. The full set lives at `benchmarks/showcase_v0.4/` (and is rendered on the GitHub Pages site). All nine items are produced by the cascade: six are Phase 7 pilot cascade traces; the three Sanskrit chandas items are live cascade outputs regenerated against the prompts in `scripts/showcase_specs.toml` with `source` = `live_cascade`. Each Sanskrit trace ships with the `tools/sanskrit_chandas.py` validator's report attached; the current cascade is not chandas-aware, so the validator's pass/fail signal is informational only. Future work includes a chandas-aware composite scorer and a learned chandas-conditional gate; we discuss this scope-cut in §11 and §14. This section walks each item briefly; the full traces are in the artefact directory. Table 5 indexes every showcase slug with its trace provenance, committed surface, and validator status.

Table 5: T4 — Showcase items registry. "Source" is the trace provenance: `live_cascade` for the three Sanskrit items (regenerated under the live-cascade mode), `phase7_cascade` for the curated English-poetry and scientific-creativity items. "Commit" names the surface that the cascade actually committed (draft / shadow revision / committed_choice). "Validator" is reported informationally; for Sanskrit items the cascade scorer is not chandas-aware (future-work ladder).

Slug	Domain	Source	Commit	Validator
`english_dickinson_slant`	`poetry_gen`	`phase7_cascade`	`draft`	imagism noted
`english_imagist_haiku`	`poetry_gen`	`phase7_cascade`	`draft`	5-7-5 pass
`english_pastoral_traditional`	`poetry_gen`	`phase7_cascade`	`draft`	5-7-5 pass
`sanskrit_anustubh`	`sanskrit_chandas`	`live_cascade_v0_4_1`	`draft`	review (expected 32 syllables (4 pādas × 8); found 185)
`sanskrit_gayatri`	`sanskrit_chandas`	`live_cascade_v0_4_1`	`draft`	review (expected 24 syllables (3 pādas × 8); found 214)
`sanskrit_indravajra`	`sanskrit_chandas`	`live_cascade_v0_4_1`	`draft`	review (expected 44 syllables (4 pādas × 11); found 348)
`science_galaxy_arms`	`sci_creativity`	`phase7_cascade`	`draft`	n/a
`science_ice_geometry`	`sci_creativity`	`phase7_cascade`	`draft`	n/a
`science_unreasonable_effectiveness`	`sci_creativity`	`phase7_cascade`	`revision`	n/a

13.1 Sanskrit chandas (3 items, live cascade traces)

anuṣṭubh. A 4×8 śloka on the theme of *vimarśa* (reflective recognition), generated live by the cascade against the prompt in `scripts/showcase_specs.toml` with `style="sanskrit_chandas"`. The committed surface is what the cascade emitted; the chandas validator's syllable-count and pattern-match report is included in the trace. We treat the validator as a reporting-only signal: the cascade is not yet conditioned on chandas constraints, so a non-conformant validator output is an honest record of the substrate's current ability rather than a release blocker.

gāyatrī. A 3×8 invocation to *citi-śakti* as the witnessing ground (sākṣin), again live-regenerated. The trace records draft, shadow revision, vimarśa event flag, BMR ΔF, and the validator's pad-by-pad syllable count and any deviation from the canonical pattern.

indravajrā. A 4×11 invocation generated against the indravajrā metre prompt. The metre's strict GG L GG L L G L G G pattern is a hard target the current cascade is not optimised for; the trace ships the cascade's output verbatim, with the validator's reported deviations attached. Whether and how a chandas-aware scorer recovers metre conformance is a future-work falsification target.

13.2 English poetry (3 items, real Phase 7 cascade traces)

Imagist haiku (`p06`, gate worked). A 5/7/5 imagist haiku ("White feather, iron rails / Sings as the train rushes through / Empty platform sleeps") was the highest-composite Phase 7 output for `poetry_gen` ($\sim$0.635). Honest reading: the *vimarśa* event *did not* fire on this item, so the cascade committed the draft. The shadow-revision pass produced "Sings when trains pass through" — a slightly different image but the proxy scorer rates it 0.59 vs. the draft's 0.64, so the event-gate's decision to keep the draft was *correct* by the scorer. This is the cascade's gate working as designed on an item where the draft was already good.

Dickinson slant (`p07`, gate miss). A free-verse poem in the Dickinson tradition (slant rhyme, en-dash mid-thought, anaphora). The *vimarśa* event did not fire, but the shadow revision (composite 0.567) clearly beat the draft (composite 0.504). This is exactly the calibration gap H8b documents: the event-gated commit policy under-fires on this domain. The H8c leaderboard's verdict that `always_revise` out-performs `event_gated` is driven by items like this.

Pastoral, traditional metre (`p01`, gate worked). A traditional pastoral 5/7/5 with a rain-on-tin auditory image. *Vimarśa* did not fire. The shadow revision tried to escalate "sings" $\rightarrow$ "shrieks through" and "deepens to" $\rightarrow$ "devours the song" — a louder register. The proxy scorer rates the draft (0.566) above the revision (0.523), so the event-gate's decision to keep the draft was correct. The revision is shown side-by-side in the showcase for comparison; this is the cascade *not* mistaking volume for craft.

13.3 Scientific creativity (3 items, real Phase 7 cascade traces)

Why mathematics works in physics (`s06`, vimarśa fired). The gold-star case in the current release. Vimarśa fired (`vimarsa_event=True`), the gate committed the revision, and the revision (composite 0.618) beat the draft (composite 0.587). The revision pass also converted the title from "A Contrarian View" to "The Persistent Mystery", reframing the essay from polemic to honest puzzle-statement.

Why ice floats — beyond the textbook (`s02`, gate miss). Vimarśa did not fire, so the cascade committed the draft (composite 0.535), but the shadow revision scored higher (0.593). The revision changed the title from "Beyond the Textbook" to "Geometry and Thermodynamic Precarity", a sharper framing that matches the brief. Items like this are the basis for H8c showing `always_revise` outperforming `event_gated`.

Why galaxies have spiral arms (`s05`, largest revision uplift). The largest single-item shadow-revision uplift in the pilot (draft 0.486 → revision 0.595, $\Delta = +0.110$). Vimarśa did not fire so the cascade committed the draft; the shadow revision sharpened "Framing 1 - The Traffic Jam" to "Framing 1 - Density Waves", upgrading the analogy to its proper physics name (Lin–Shu density-wave theory). The revision is shown for comparison; the gold-star reading is that the cascade *produced* the better surface but failed to commit it. Future work on the learned gate is partly about catching items like this one.

14 Limitations and Threats to Validity

Pilot is under-powered by design. The earlier pilot was scoped to n=5/domain (N_{paired}=20), K=3, max_tokens= 200 to fit a $20 Haiku budget. With n=5 the exact sign-flip permutation test has a hard floor of p=0.0312 and Holm–Bonferroni with m=4 floors the smallest possible adjusted p at 0.125. *No primary hypothesis (H1–H4) can cross the pre-registered Holm $p < 0.05$ bar at this sample size*, regardless of effect direction or magnitude. A properly powered run at $n\sim$20/domain is required to claim or rule out a stable architectural effect.

The earlier pilot reported a directional-negative outcome that the current release corrects. On all four primary domains in the earlier pilot, the `haiku_cascade` arm trailed the clean-substrate `haiku_bare` arm (Hedges' $g \in [-0.90, -0.22]$); the legacy random-effects pool came out at $g = -0.46$ (95% CI $[-0.93, +0.01]$), and the legacy H6 and H7 contrasts also trailed across all four domains. We interpret this finding three ways. First, the prior "cascade beats bare" lift, on a clean Haiku CLI substrate that no longer leaks Claude-Code context into the bare control, did not reproduce; this is consistent with the prior adversarial review's specific prediction. Second, against budget-matched (legacy H6) and protocol-matched (legacy H7) controls, the cascade was not preferred on that small sample, and both "$+K$ compute on a single pass" and "a generic 2-pass revise" — simpler interventions — performed at least as well at $n = 5$. Third, the within-cascade legacy H8 test was directionally positive ($\Delta = +0.004$, $g = +0.21$) on the $n = 3$ items where event-gated commit chose to revise, but $n = 3$ was far too small to adjudicate causality. The current design is the answer to that mismatch: by lifting the commit decision into the four-policy multiplexer, by training a learned gate on the H8a paired-revision data, and by enlarging the within-cascade contrast (H8a) to all items rather than only those the event gate chose to revise, the current release measures the mechanism layer the earlier result implicated. The current numbers — shadow revision strictly beats draft at $g = +0.65$ and a learned gate beats event-gated at F1 = 0.65 vs. 0.52 — are exactly the kind of finding the earlier pilot motivated and could not, at its sample size, deliver.

Cascade K deferred from K=4 to K=3. The current pilot used K=3 to keep total Haiku spend under cap. An earlier prototype ran at K=4; a follow-up at K=4 on a properly powered n=20/domain pull will eliminate this confound.

Scoring functions are local proxies, not gold standards. We deliberately use local deterministic scorers to keep the contrast fair across arms. They are, however, proxies: POE-Metric is a more nuanced rubric than what our `score_poetry_gen` function captures, and the

Wittgenstein aspect-multiplicity score is constructed from cosine to author-supplied aspect targets rather than from human raters. A follow-up with human raters (or a frontier-model judge) is necessary before the absolute scores can be interpreted as POEMetric scores in the original sense; our *contrasts* between arms remain valid because the same scorer is applied uniformly to all arms.

Prompt-engineering confound. The Claude Code CLI surrounds every prompt with a Claude Code system context (visible in the cache-creation token count). This system context could in principle help or hurt the Haiku arm in domain-specific ways. We held the user prompt constant across arms and used Claude Code's own `-p` non-interactive mode (the most LLM-API-like surface), but we cannot prove the system prompt does not bias the Haiku arm's outputs.

Model substrate confound. Qwen2-1.5B-Instruct is the only local LLM we tested. The PCE design is substrate-agnostic, but a wider sweep (Qwen2-7B, Llama-3-8B, Mistral-Nemo) is needed to confirm the cascade contributes consistently across substrate scales.

Internal-validity vs external-validity. The H6 test (within-PCE event vs no-event) is a within-condition test and does not establish that the recursive layer would help under a different scoring function or a different prompt distribution. We report it as an *internal* validity check, not as a generalisable claim.

Single-author authorship of the benchmark items. The Wittgenstein aspect-shift items are author-constructed. We avoided cherry-picking by stratifying across canonical poetic devices (metaphor, synecdoche, allegory, etc.), but the items have not been reviewed by independent literary scholars. We publish them in Appendix C so the community can replace or augment them.

15 Conclusion

We have presented *Pratyabhijñā* × *Active Inference* as a falsifiable engineering hypothesis: that a recursive self-reflexivity layer, formalised as a Bayesian-model-reduction-gated re-entry into the cascade, produces measurable lifts on benchmarks that reward interpretive multiplicity. The plugin (`pratyabhijna-creative-engine`) is shipped with all artefacts needed to replay the experiments end-to-end.

The negative-result obligation in our pre-registration means we report what the data show whether or not it flatters the theory. In this run, no primary hypothesis crosses the Holm-adjusted $p < 0.05$ threshold against Haiku, the substrate-matched sensitivity contrasts are small and non-directional, and the *vimarśa* loop fired on 0/30 cascade trials. Read together these tell a sharp, useful story: the architecture, as configured here, is non-different from a temperature-sampler at the matched substrate, and its central novel mechanism (*vimarśa*) was dormant. We have published the exact thresholds, the exact probes, the per-item JSONs, the figures, and the autoreport, so the next experimenter can re-run with relaxed thresholds, a larger K, longer `max_tokens`, or a stronger substrate without re-implementing the scaffolding.

What follows. We call out three directions that fall out of the design but are out of scope here: (i) replacing `kriya` with a Claude/Sonnet-class polish model so the substrate is no longer the bottleneck on fluency; (ii) replacing the Hopfield store with a larger episodic memory and re-running consolidation cycles across multi-day sessions; (iii) building a continuously-running PCE agent that uses `vimarsa` as a controller for tool invocation rather than only for surface revision.

Acknowledgements. The Pratyaksha context-engineering harness (Sathish, 2025b) provided the "witness invariant" discipline that this whole project is held to. The `ralph-loop` pattern from Anthropic's Claude Code plugin gallery was adopted as the completion-promise spine. The `attractor-flow` plugin (Sathish, 2025a) provided the engineering template for "MCP → real computation → audit-traceable output".

A Operator Specifications

This appendix gives the full equations and signatures for each operator. The reference implementation is in `src/pce/operators/` and is exercised by the test suite in `tests/operators/`.

A.1 Cit

$$\textsc{Cit}(\text{prompt}, \tau, \text{seed}) \to \text{Candidate}$$

The single-sample LLM generation. Sampler defaults: τ=0.9, top-p=0.95, top-k=50. Determinism: torch RNG on CPU (MPS does not support per-device generators) feeds `multinomial`; the device runs the model itself.

A.2 Ānanda

$$\bar{\textsc{A}}\textsc{nanda}(c, \kappa, R) = w_{\text{coh}} \cos(\phi(c), \phi(\kappa.\text{text})) + w_{\text{div}}\big(1 - \max_{r \in R} \cos(\phi(c), \phi(r))\big) + w_{\text{form}}\, F(c)$$

where κ is the constraint, R is a retrieval set, and $F(c)$ is a form-fidelity heuristic (e.g., line-count proxy for haiku). Default weights: $(w_{\text{coh}}, w_{\text{div}}, w_{\text{form}}) = (0.5, 0.3, 0.2)$.

A.3 Icchā

K candidate generations under a constraint-augmented prompt. The constraint is inlined as natural prose to avoid bracket-leakage in small-LM substrates (we calibrated this in Phase 6: the original `[Constraint: ...]` bracketed form leaked the literal bracket back into surfaces).

A.4 Apohana

$$a_i = 1 - \max_j \cos\big(\phi(c_i), \phi(m_j)\big), \quad m_j \in \text{must-avoid}$$

The contrastive exclusion score; bounded in $[0, 2]$ but in practice $[0, 1]$ because cosine of normalised embeddings is in $[-1, 1]$. The implementation clips to $[0, 1]$ for numerical safety.

A.5 Jñāna (Bayesian Model Reduction)

Let pseudo-counts $\alpha_i = \alpha_0 + \beta a_i$ for $i \in 1 \dots K$, $\alpha_0 = 1$, $\beta = 4$. The Dirichlet–multinomial evidence under the rich model (uniform prior) is

$$\log p(o \mid m_F) = \log \Gamma(K) - \sum_i \log \Gamma(1) + \sum_i (\alpha_i - 1) \log a_i.$$

The reduced model concentrates the prior on $i^* = \arg\max_i \alpha_i$ via $\alpha'_i = \alpha_i + 1$ at i^* only. The free-energy delta is

$$\Delta F = \log p(o \mid m_R) - \log p(o \mid m_F),$$

computed using `scipy.special.gammaln` for numerical stability. We do not import `pymdp`'s JAX backend because we need exact CPU determinism for the audit log.

A.6 Kriyā

Three modes are supported. The `verbatim` mode returns the chosen candidate's surface unchanged. The `polish` mode normalises whitespace and trims trailing assistant-style babble (e.g. `Human:`, `Assistant:`). The `claude_polish` mode (opt-in) sends the surface to Claude Code with a tight "preserve content, polish surface" prompt; it is not used in the headline benchmarks.

A.7 Vimarśa

$$\text{event} = \mathbb{1}[\text{aspect\_multiplicity} \vee \text{low\_novelty} \vee \text{negative\_residual\_}\Delta F]$$

ASPECT_MULTIPLICITY is true iff at least two aspect-target embeddings have cosine ≥ 0.40 to the surface embedding (calibrated in Phase 6, see ADR-002 for the calibration trace). LOW_NOVELTY is true iff the maximum cosine to any retrieval-set element is ≥ 0.85. NEGATIVE_RESIDUAL_ΔF is true iff the BMR ΔF recorded on the chosen candidate is ≥ 0 (i.e., the prior was not yet falsified — the cascade should re-enter).

The novelty signal. VIMARŚA.NOVELTY is computed as $1 - \max_{r \in R} \cos(\phi(\text{surface}), \phi(r))$, clipped to $[0, 1]$. It is recorded in the audit log alongside `event`.

B Consolidation Cycles

The Hopfield store $\mathcal{S}$ holds a set of normalised embedding vectors. Two consolidation cycles run on user request through the MCP tools `pce.store.consolidate_sws` and `pce.store.consolidate_rem`; both are deterministic given a seed.

B.1 Slow-wave-sleep abstraction

Given $\mathcal{S} = \{v_1, \ldots, v_n\}$ and a target compression rate $\rho \in (0, 1)$, we run k-means with $k = \lceil \rho n \rceil$ initialised by k-means++. Cosine distance is used; the centroid is the unit-normalised mean. The centroids *replace* the cluster members in $\mathcal{S}$: this is the abstraction step. Numerical safeguard: cosine distances are clipped to $[0, 2]$ before normalisation in the k-means++ probability update (we hit a `Probabilities are not non-negative` crash here in Phase 8 before adding the clamp; see commit `a27e886`).

B.2 REM-style replay

Given $\mathcal{S}$ and a temperature T, sample m items by a Metropolis chain on $\mathcal{S}$ with energy $E(v) = -\log p(v)$ where p is the empirical density estimated from cluster sizes. Each accepted item is perturbed by $v + \epsilon$, $\epsilon \sim \mathcal{N}(0, \sigma I)$, and re-stored. This raises the salience of rare-but-coherent attractors without erasing dense modes. Defaults: T=0.7, σ=0.05, $m = \min(n/4, 32)$.

B.3 Why both

SWS abstracts; REM diversifies. A single cycle of either alone collapses the store toward the data mode (SWS) or noise (REM). In our cascade we run them in alternation: SWS after a creative session, REM before the next session. The order is recorded in the audit log so a downstream user can replay both cycles deterministically.

C Benchmark Items

The full item lists live in `benchmarks/items.py` as Python literals; we include the headline IDs and topics here. The literal text of each item, the must-avoid set, and the aspect targets are loaded *verbatim* into the benchmark driver, which writes the active set to `benchmarks/results/<domain>.json` alongside the model outputs.

C.1 Poetry generation

A stratified set across the four canonical English forms (haiku, tanka, sonnet, free-verse) plus a small ghazal slice. Each item carries (topic, form, must-avoid). Topics deliberately span sensory, emotional, abstract, and metalinguistic categories so the POEMetric axes are exercised independently.

C.2 Poetry interpretation (Wittgenstein aspect-shift)

Each item is a single line of poetry with two pre-registered candidate readings. Sources include Eliot, Dickinson, Bashō, Plath, and selected fragments of Heraclitus and Cavafy. The aspect targets are author-supplied phrases describing each reading (e.g., for *"I have measured out my life with coffee spoons"*: *a life of small repetitive rituals* vs *a quiet despair at unlived possibilities*). The retrieval set contains intentionally banal continuations (*coffee in the morning is essential*) so the novelty axis can punish high-similarity surfaces.

C.3 Alternative-uses task (CreativityPrism slice)

Eight common objects (brick, paperclip, sock, blanket, etc.). The scorer rewards (a) cosine novelty against the standard everyday use, (b) lexical diversity, (c) presence of feasibility-cue tokens ("portable", "reusable", "scalable", etc.).

C.4 Scientific creativity (BBH/MacGyver-style)

Each item is a single open-ended scientific question with 2–3 "framings" (analogical, mechanistic, mathematical). The question stems span evolutionary biology, statistical mechanics, computational neuroscience, and theoretical physics. The composite weights non-textbook novelty, framing-coverage, and depth-via-length.

D Scoring Functions

All four scoring functions live in `benchmarks/scoring.py` and are deterministic. Each returns a `Score(axes: dict[str, float], composite: float)`. The composite is the unweighted mean of the per-axis scores; we keep the per-axis breakdown in the audit log for the figures in §10.

D.1 `score_poetry_gen`

`score_poetry_gen` returns six axes, all computed from local statistics. *creativity* is $1 - \cos(\phi(\text{surface}), \phi(\text{topic}))$, normalised to $[0, 1]$. *lexical diversity* is the type-token ratio. *idiosyncrasy* is one minus the maximum cosine to a corpus of 20 "canonical" poems on similar topics. *emotional resonance* fires on the presence of any of {love, fear, hope, grief, joy, longing} token-stems and is weighted by sentence count. *literary devices* is a heuristic regex check for alliteration, anaphora, simile cues ("like", "as"), and enjambment. *imagery* is the cosine to a small imagery-vocabulary cluster centroid drawn from {light, shadow, sky, river, leaf, ...}.

D.2 `score_poetry_interp`

`score_poetry_interp` returns three axes. *aspect_count* is the count of aspect targets whose cosine to the surface is at least 0.40. *novelty* is one minus the maximum cosine to the retrieval set. *coverage* is the mean cosine across all aspect targets.

D.3 `score_aut`

`score_aut` returns three axes. *creativity* is one minus the cosine to the standard-use sentence. *lexical_diversity* is the type-token ratio. *feasibility* fires on the presence of feasibility-cue tokens in the candidate set.

D.4 `score_sci_creativity`

`score_sci_creativity` returns four axes. *non_textbook_novelty* is one minus the cosine to the textbook one-liner. *framing_coverage* is the mean cosine across the framing targets. *depth* is log-length normalised to $[0, 1]$. *multi_framing* is the count of framing targets whose cosine is at least 0.35, divided by the total number of framing targets.

E Statistical Protocol

The statistical pipeline lives in `benchmarks/stats.py` and is exercised by the synthetic-data tests in `tests/test_stats.py`. We re-state it here as a self-contained appendix.

E.1 Paired permutation

For each hypothesis we form the per-item paired delta $d_i = T_i - C_i$ where T is the treatment composite and C is the control composite. We test $H_0 : \mathbb{E}[d] = 0$ vs $H_1 : \mathbb{E}[d] > 0$ by sign-flip permutation: under H_0, the sign of d_i is exchangeable. We enumerate all 2^n sign assignments when $n \leq 18$ (the exact regime), otherwise sample 50 000 random sign-vectors. The p-value is the proportion of permuted means at least as large as the observed mean, with a $+1$ smoother in the Monte-Carlo regime: $p = (k + 1)/(N_{\text{perm}} + 1)$.

E.2 Hedges' g

$$g = J \frac{\bar{d}}{s_d}, \qquad J = 1 - \frac{3}{4n - 5},$$

where s_d is the unbiased SD of d. J is the small-sample bias correction (Borenstein 2009). When $s_d = 0$ we report $g = 0$ by convention.

E.3 BCa bootstrap CI

10 000 resamples of the paired deltas with replacement; bias-correction $z_0 = \Phi^{-1}(\#\{b : \bar{d}^*_b < \bar{d}\}/B)$ and acceleration a via jackknife. We use `scipy.stats.norm.cdf/ppf` for the BCa adjustments and clip the alpha-quantiles into $[0, 1]$ for numerical safety.

E.4 Wilcoxon signed-rank

`scipy.stats.wilcoxon` with `alternative="greater"`, `zero_method="wilcox"`. Reported alongside the permutation p for robustness against extreme-tail behaviour.

E.5 Holm–Bonferroni

We adjust $\{p_{H_1}, p_{H_2}, p_{H_3}, p_{H_4}\}$ jointly. Sorted ascending; the i-th adjusted value is $\max(\,(m-i)\,p_{(i)},\, p^{\text{adj}}_{(i-1)}\,)$, clamped to $[0,1]$. H5 and H6 are not double-counted.

E.6 Power

Approximate paired-t power under a noncentral-t distribution: power $= 1 - \Pr(T_{\text{nc}} \le t_{\text{crit}})$ with $t_{\text{crit}} = t_{\alpha,\,n-1}$ and noncentrality $\lambda = g\sqrt{n}$. We report a-priori (assumed $g = 0.5$) and retrospective (observed g) power.

E.7 H6

Within-PCE event-vs-no-event. Independent-samples Mann–Whitney U (`alternative="greater"`), Hedges' g for independent samples (pooled SD), and a 10 000-resample bootstrap CI for the difference of means. "Supported" iff U-test $p < 0.05$ *and* the bootstrap CI's lower bound is strictly positive.

F Reproducibility

F.1 Recipe

```
git clone https://github.com/SharathSPhD/pratyabhijna.git
cd pratyabhijna
uv venv && uv sync --extra dev
uv run python scripts/verify_real_model.py     # downloads HF models
uv run pytest -m "not slow"                    # run unit tests
uv run python benchmarks/driver.py \
    --n-poetry-gen 12 --n-poetry-interp 10 \
    --n-aut 8 --n-sci-creativity 8 \
    --max-tokens 120 --K 4 \
    --out-dir benchmarks/results
uv run python benchmarks/stats.py
uv run python benchmarks/figures.py
make -C paper                                   # builds paper/main.pdf
```

F.2 Audit trail

Every Phase 9 command writes a line to `audit/phase9/`: the driver invocation, model checksum, git SHA, wall-clock. The Phase 8 plugin smoke audit lives in `audit/phase8/`. The seed (`4242`) and the cascade hyperparameters ($K = 4$, `max_tokens`=120, $\tau = 0.9$, top-$p = 0.95$, top-$k = 50$, aspect-cosine-hit= 0.40) are written into every output file.

F.3 Hardware

The benchmark in this paper was run on Apple Silicon (`mps` backend, `float16`). The same code paths support CUDA (auto-detected) and CPU (`PCE_DEVICE=cpu`). Substitute model: `LMConfig.model_id` accepts any HF causal LM.

F.4 Container

A minimal Dockerfile is provided in `infra/Dockerfile` and a `Makefile` target `make docker-bench` replays the benchmark inside the container. The image pins `torch`,

`transformers`, and `sentence-transformers` versions so a third party's run is bit-for-bit reproducible *up to* the inherent non-determinism of MPS / CUDA softmax (which we measure to be $< 10^{-3}$ on the composite scores).

G Audit Trail

The single most important architectural commitment we make in this paper is that every numerical claim in the body is derived from a JSON or text artefact under `audit/` or `benchmarks/results_v0.4/` that was produced by code in `src/` and is reproducible from the recipe in Appendix F. The mapping is straightforward. Operator behaviour is anchored in `tests/operators/test_*.py`, where passing tests are the ground truth for the equations in Appendix A. Plugin smoke output lands in `audit/phase8/verify_artifact.json`, with the manifest test (`tests/test_plugin_manifests.py`) asserting that the README's MCP-tool count matches the FastMCP server's runtime count. Per-arm raw outputs and per-axis scores live under `benchmarks/results_v0.4/poetry_gen.json`, `benchmarks/results_v0.4/poetry_interp.json`, `benchmarks/results_v0.4/aut.json`, and `benchmarks/results_v0.4/sci_creativity.json`. Pre-registered hypothesis tests are aggregated in `benchmarks/results_v0.4/stats.json`. Figures live in `paper/figures/v0.4/`, regenerated from the same JSON by `benchmarks/figures.py`. The auto-generated headline table (Table 1) is rebuilt by `benchmarks/autoreport.py` after the statistics finish. The Sonnet judge audit trail is split between `benchmarks/results_v0.4/judge.jsonl` (one row per pair, with both `prompt_sha256` and the `formatted_prompt_sha256` fields, plus a placeholder `input_tokens` field whose provenance is documented in `benchmarks/results_v0.4/judge_agreement.json`) and `benchmarks/results_v0.4/judge_agreement.json` itself (the H9 aggregate plus the recovery audit block).

G.1 What "no-stub" means here

The `ralph-loop` completion-promise spine adopted from Anthropic's plugin gallery enforces a hard constraint: no stubbed function, no mocked output, no fabricated number. A function that cannot fulfil its contract raises; a benchmark item that cannot be scored is recorded with `error` set and propagated through the statistical pipeline as missing data (paired observations require both arms to be present). The Phase 5 "anti-stub" gate (`audit/phase5/anti_stub.json`) enumerates every operator file and every test file that participates in this contract.

References

Pratyabhijñāhrdayam: The Secret of Self-Recognition. Motilal Banarsidass, Delhi, 1963. URL https://www.motilalbanarsidass.com/en-us/products/pratyabhijnahrdayam-the-secret-of-self-recognition-sanskrit-text-with-english-translation-notes-and-introduction.

infer-actively/pymdp v1.0.0 release notes. https://github.com/infer-actively/pymdp/releases/tag/v1.0.0, 2026. JAX-first backend; pymdp.legacy; Agent API rollout.

Abhinavagupta. *Īśvarapratyabhijñāvimarśinī*. Motilal Banarsidass, Delhi, 1986. ISBN 8120800222. Sanskrit text with commentary Bhāskarī; reprint of earlier Sarasvatī Bhavana editions.

Yuntao Bai, Saurav Kadavath, Sandipan Kundu, Amanda Askell, Jackson Kernion, Andy Jones, Anna Chen, Anna Goldie, Azalia Mirhoseini, Cameron McKinnon, Carol Chen, Catherine Olsson, et al. Constitutional AI: Harmlessness from AI feedback, 2022.

Roger E. Beaty, Mathias Benedek, Scott Barry Kaufman, and Paul J. Silvia. Default and executive network coupling supports creative idea production. *Scientific Reports*, 5:10964, 2015. doi: 10.1038/srep10964.

Daniel Braun. Acquiescence bias in large language models. In *Findings of the Association for Computational Linguistics: EMNLP 2025*, pages 11341–11355, Suzhou, China, 2025. Association for Computational Linguistics. doi: 10.18653/v1/2025.findings-emnlp.607.

Qian Cao, Xiting Wang, Yuzhuo Yuan, Yahui Liu, Fang Luo, and Ruihua Song. Evaluating text creativity across diverse domains: A dataset and large language model evaluator. *arXiv preprint arXiv:2505.19236*, 2025.

Qunlin Chen, Yoed N. Kenett, Zaixu Cui, et al. Dynamic switching between brain networks predicts creative ability. *Communications Biology*, 8, 2025. doi: 10.1038/s42003-025-07470-9.

Laura Desirée Di Paolo, Ben White, Avel Guénin-Carlut, Axel Constant, and Andy Clark. Active inference goes to school. the importance of active learning in the age of large language models. https://osf.io/zwa83, 2024.

Karl Friston. The free-energy principle: a unified brain theory? *Nature Reviews Neuroscience*, 11(2):127–138, 2010. doi: 10.1038/nrn2787.

Karl Friston. Life as we know it. *Journal of The Royal Society Interface*, 10(86):20130475, 2013. doi: 10.1098/rsif.2013.0475.

Karl J. Friston and Will Penny. Post hoc bayesian model selection. *NeuroImage*, 56(4):2089–2099, 2011. doi: 10.1016/j.neuroimage.2011.03.062.

Karl J. Friston, Vladimir Litvak, Ashwini Oswal, Adeel Razi, Klaas E. Stephan, Bernadette C. M. van Wijk, Gabriel Ziegler, and Peter Zeidman. Bayesian model reduction and empirical Bayes for group (DCM) studies. *NeuroImage*, 128:413–431, 2016. doi: 10.1016/j.neuroimage.2015.11.015.

Karl J. Friston, Marco Lin, Christopher D. Frith, Giovanni Pezzulo, J. Allan Hobson, and Sasha Ondobaka. Active inference, curiosity and insight. *Neural Computation*, 29(10):2633–2683, 2017. doi: 10.1162/neco_a_00999.

Karl J. Friston, Thomas Parr, and Peter Zeidman. Bayesian model reduction. *arXiv preprint arXiv:1805.07092*, 2018. posted 2018-05-18; revised 2019-10-14; stat.ME.

Conor Heins, Beren Millidge, Daphne Demekas, Brennan Klein, Karl Friston, Iain D. Couzin, and Alexander Tschantz. pymdp: A python library for active inference in discrete state spaces. *Journal of Open Source Software*, 7(73):4098, 2022. doi: 10.21105/joss.04098.

Oliver Hellwig and Erica Biagetti. The Sanskrit Sembank. *Language Resources and Evaluation*, 59(4):3635–3658, 2025. doi: 10.1007/s10579-025-09852-1.

Oliver Hellwig, Sebastian Nehrdich, and Sven Sellmer. Data-driven dependency parsing of Vedic Sanskrit. *Language Resources and Evaluation*, 57(3):1173–1206, 2023. doi: 10.1007/s10579-023-09636-5.

Zhaoyi Joey Hou et al. Creativityprism: A holistic benchmark for large language model creativity. https://arxiv.org/abs/2510.20091, 2025a.

Zhaoyi Joey Hou et al. Creativityprism: A holistic benchmark for large language model creativity. *arXiv preprint arXiv:2510.20091*, 2025b.

Dmitry Krotov and John J. Hopfield. Dense associative memory for pattern recognition, 2016.

Walter Laurito, Benjamin Davis, Peli Grietzer, Tomáš Gavenčiak, Ada Böhm, and Jan Kulveit. AI–AI bias: Large language models favor communications generated by large language models. *Proceedings of the National Academy of Sciences*, 122(31), 2025. doi: 10.1073/pnas.2415697122.

David Lawrence. Tantric argument: The transfiguration of philosophical discourse in the pratyabhijna system of utpaladeva and abhinavagupta. *Philosophy East and West*, 46(2): 165, 1996. doi: 10.2307/1399403.

Bingru Li, Han Wang, and Hazel Wilkinson. Poemetric: The last stanza of humanity. https://arxiv.org/abs/2604.03695, 2025.

Bingru Li, Han Wang, and Hazel Wilkinson. Poemetric: The last stanza of humanity. *arXiv preprint arXiv:2604.03695*, 2026.

Yang Liu, Dan Iter, Yichong Xu, Shuohang Wang, Ruochen Xu, and Chenguang Zhu. G-Eval: NLG evaluation using Gpt-4 with better human alignment. In *Proceedings of the 2023 Conference on Empirical Methods in Natural Language Processing*, pages 2511–2522, Singapore, 2023. Association for Computational Linguistics. doi: 10.18653/v1/2023.emnlp-main.153.

Aman Madaan, Niket Tandon, Prakhar Gupta, Skyler Hallinan, Luyu Gao, Sarah Wiegreffe, Uri Alon, Nouha Dziri, Shrimai Prabhumoye, Yiming Yang, Shashank Gupta, Bodhisattwa Prasad Majumder, Katherine Hermann, Sean Welleck, Amir Yazdanbakhsh, and Peter Clark. Self-Refine: Iterative refinement with self-feedback. In *Advances in Neural Information Processing Systems*, volume 36, pages 46534–46594, 2023. doi: 10.52202/075280-2019.

Ronald Mtenga, Mathias Bode, and Radwa Khalil. Do not let the beginning trap you! on inhibition, associative creative chains, and hopfield neural networks. *The Journal of Creative Behavior*, 2024. doi: 10.1002/jocb.680.

Sebastian Nehrdich, Oliver Hellwig, and Kurt Keutzer. One model is all you need: ByT5-Sanskrit, a unified model for Sanskrit NLP tasks. In *Findings of the Association for Computational Linguistics: EMNLP 2024*, pages 13742–13751, Miami, Florida, USA, 2024. Association for Computational Linguistics. doi: 10.18653/v1/2024.findings-emnlp.805.

Peter Organisciak, Selcuk Acar, Denis Dumas, and Kelly Berthiaume. Beyond semantic distance: Automated scoring of divergent thinking greatly improves with large language models. *Thinking Skills and Creativity*, 49:101356, 2023. doi: 10.1016/j.tsc.2023.101356.

Hubert Ramsauer, Bernhard Schäfl, Johannes Lehner, Philipp Seidl, Michael Widrich, Thomas Adler, Lukas Gruber, Markus Holzleitner, Milena Pavlović, Geir Kjetil Sandve, Victor Greiff, David Kreil, Michael Kopp, Günther Klambauer, Johannes Brandstetter, and Sepp Hochreiter. Hopfield networks is all you need, 2020.

Sharath Sathish. attractor-flow: a cursor plugin coupling attractor-network computation to mcp. https://github.com/SharathSPhD/attractor-flow, 2025a.

Sharath Sathish. Pratyaksha context engineering harness: a witness-invariant context discipline for long-running agents. https://github.com/SharathSPhD/pratyaksha-context-eng-harness, 2025b.

Thibault Sellam, Dipanjan Das, and Ankur Parikh. BLEURT: Learning robust metrics for text generation. In *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics*, pages 788–801. Association for Computational Linguistics, 2020. doi: 10.18653/v1/2020.acl-main.704.

Noah Shinn, Federico Cassano, Edward Berman, Ashwin Gopinath, Karthik Narasimhan, and Shunyu Yao. Reflexion: Language agents with verbal reinforcement learning, 2023.

Nisan Stiennon, Long Ouyang, Jeff Wu, Daniel M. Ziegler, Ryan Lowe, Chelsea Voss, Alec Radford, Dario Amodei, and Paul Christiano. Learning to summarize from human feedback, 2020.

Mirac Suzgun, Nathan Scales, Nathanael Schärli, Sebastian Gehrmann, Yi Tay, Hyung Won Chung, Aakanksha Chowdhery, Quoc Le, Ed Chi, Denny Zhou, and Jason Wei. Challenging BIG-Bench tasks and whether chain-of-thought can solve them. In *Findings of the Association for Computational Linguistics: ACL 2023*, pages 13003–13051, Toronto, Canada, 2023. Association for Computational Linguistics. doi: 10.18653/v1/2023.findings-acl.824. arXiv:2210.09261.

Yufei Tian, Abhilasha Ravichander, Lianhui Qin, Ronan Le Bras, Raja Marjieh, Nanyun Peng, Yejin Choi, Thomas Griffiths, and Faeze Brahman. Macgyver: Are large language models creative problem solvers? In *Proceedings of the 2024 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies (Volume 1: Long Papers)*, pages 5303–5324, Mexico City, Mexico, 2024. Association for Computational Linguistics. doi: 10.18653/v1/2024.naacl-long.297. arXiv:2311.09682.

William S. Waldron. *The Buddhist Unconscious: The Ālaya-vijñāna in the Context of Indian Buddhist Thought*. Routledge, 2003. ISBN 9781134428861. doi: 10.4324/9780203451175.

Zi-Han Wang, Lam Nguyen, Zhengyang Zhao, et al. Creativebench: Benchmarking and enhancing machine creativity via self-evolving challenges. *arXiv preprint arXiv:2603.11863*, 2026.

Natalya Weber, Christian Guckelsberger, and Tom Froese. Untapped potential in self-optimization of hopfield networks: The creativity of unsupervised learning. *Artificial Life*, 2025. doi: 10.1162/ARTL.a.10. arXiv:2501.04007.

Ludwig Wittgenstein. *Philosophical Investigations*. Wiley-Blackwell, Oxford, 4 edition, 2009. Original German work 1953; aspect-perception cited from Part II, xi.

Lianmin Zheng, Wei-Lin Chiang, Ying Sheng, Siyuan Zhuang, Zhanghao Wu, Yonghao Zhuang, Zhu Lin, Zhuohan Li, Dacheng Li, Eric P. Xing, Hao Zhang, Joseph E. Gonzalez, and Ion Stoica. Judging LLM-as-a-Judge with MT-bench and Chatbot Arena, 2023. URL https://arxiv.org/abs/2306.05685.