Operational Proto-Introspection in Looped Language Models

Process-Quality Taps, Executable Branching, and the Readout–Control Boundary

Jan Kirin

github.com/VykosMolt | github.com/VykosMolt/Branching-Looped-Transformer

Paper 1 is the science paper. Paper 2 contains the two-trap audit, five corrections, synthetic demonstrations, and reusable tooling; the erratum to arXiv:2604.09870 (v2) remains the correction of record. This paper uses the corrected protocol and values.

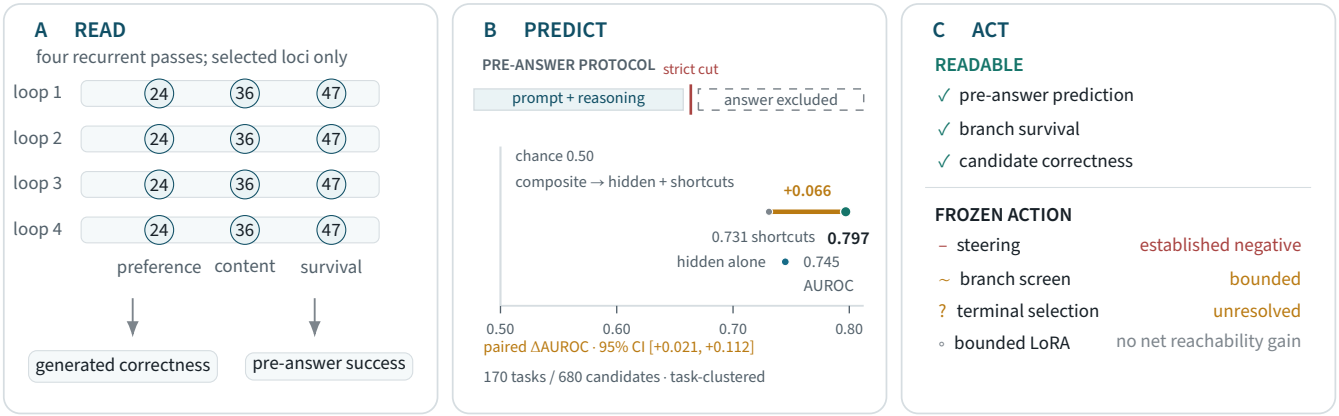
Acknowledgments. I sincerely thank Jonathan Williams for providing the Ouro-RLTT weights used as the primary experimental backbone in this work; without them, the RLTT results reported here would not exist.

Abstract

Can a language model read the quality of its own ongoing computation, and can an external intervention turn that readout into better outcomes? We test both questions in a frozen 2.6B looped transformer, Ouro-RLTT. The first answer is positive. On GSM8K, a strict pre-answer probe excludes the answer region and gold value yet predicts eventual success: hidden states plus length and log-probability shortcuts reach AUROC 0.797, versus 0.731 for the shortcuts alone (incremental +0.066; task-clustered 95% CI [+0.021, +0.112]; 170 tasks, 680 candidates). Low-capacity taps also read role-specialized properties from frozen trajectories: task-disjoint branch survival reaches 0.9697 oracle retention, with the layer-47 channel causally load-bearing; content ranking reaches 0.6310 macro top-1; and generated-branch correctness reaches AUROC 0.7755. A non-looped control replicates a same-class candidate-quality readout, so recurrence is not required for every readable signal.

The second answer is more limited. We build executable branch/carry/prune machinery over Ouro's 192-slot recurrent cache, including branch-specific cache lineage and a bit-exact residual-capture splice that recomputes only the affected suffix and saves up to 88% of per-branch layer passes. Through this substrate, no tested frozen intervention produces a validated capability gain. Directional steering is an established negative. A four-task matched-sampling branch comparison is only a bounded screen: it removes evidence for a frozen-fork gain but does not estimate a general deficit. Terminal selection remains unresolved and underpowered; generated correctness has not been shown to support reliable forced selection. A bounded 300-step LoRA changes surface behavior without improving net reachability. A rank-corrected two-null geometric audit does not support the simplest one-dimensional span-misalignment explanation; broader subspace mismatch remains untested.

We call this readable-but-not-yet-usable property **operational proto-introspection**. The model is not consulting our probes: we read its hidden trajectories, and our interventions fail to convert those readouts into validated capability. The primary powered pre-answer result is currently limited to one domain. All load-bearing current values use source-item-disjoint splits and, for pairwise scorers, antisymmetrized evaluation; correction history and the full audit protocol appear in the companion methodology paper.



Visual summary. Read, predict, act. Ouro's recurrent states support several readable process-quality signals, including a strict pre-answer increment. The action evidence is deliberately separated: steering is an established negative, the branch comparison is bounded, terminal selection is unresolved, and bounded LoRA changes surface behavior without improving net reachability.

Results at a glance

A model that could read the quality of its own ongoing computation could, in principle, act on it — branching where it is uncertain, pruning what is failing, committing where it is confident. We test both halves of that proposition in a frozen 2.6B looped transformer (Ouro-RLTT), and find clear evidence for the first and no validated capability gain from any of the frozen interventions we tested for the second.

Readable. Before the model produces an answer, its intermediate states predict whether that answer will be correct. On GSM8K, a strict pre-answer probe — cut in code to exclude the answer region and the gold value — adds statistically significant information beyond length and log-probability shortcuts (AUROC 0.797 with hidden features versus 0.731 for shortcuts alone; incremental +0.066, 95% CI [+0.021, +0.112] under a paired task-clustered bootstrap over 170 tasks; no single task drives the effect). The readable signal is not a single scalar but decomposes into role-specialized readouts that low-capacity taps recover from frozen states and that a live branching scaffold consumes: branch survivability at **0.9697** oracle retention under a task-disjoint split (with the layer-47 channel it reads causally load-bearing — ablating it collapses retention to **0.0417**), content ranking at **0.6310** macro top-1 task-disjoint, and generated-branch correctness at AUROC **0.7755**.

Not actionable. We then build the machinery through which such a signal could be acted on: autoregressive branch-specific KV-cache carry across Ouro's 192-slot recurrent cache, validated by a six-level correctness ladder (independent branch caches, batched equivalence, lineage-preserving prune/reorder, and a negative control showing the carried cache is load-bearing), together with a **bit-exact suffix-recompute splice** that cuts up to 88% of per-branch compute while matching a full recompute token-for-token. Branch-forking and prefix-sharing over standard KV caches are well established (PagedAttention, RadixAttention, SpecInfer tree attention), as is KV reuse *across* recurrent steps in depth-recurrent models (Geiping et al., 2025; Zhu et al., 2025); what we have not found combined (§7.3) is branch-specific carry with a residual-capture suffix splice that reconstructs a mid-computation-perturbed branch bit-exactly over the loop×layer recurrent cache. Through this machinery, no frozen intervention we tested produced a validated capability gain: directional steering is an established negative; branch-level and selective interventions do not establish a capability gain under the tested conditions. The bounded four-task branch screen removes evidence for a frozen-fork gain but cannot estimate a general deficit. Terminal selection remains unresolved because the clean evaluations are underpowered. Directional steering across seven methods produces unsigned effects only, with the readout direction, the empirical success direction, and the learned control direction mutually near-orthogonal. Deterministic branch injection yields no reachability gain once deconfounded against K-matched sampling. Generated-branch correctness is decodable at AUROC 0.78, yet has not been shown to support reliable forced selection. A bounded 300-step LoRA moves branch diversity and parse rates without moving outcome reachability. We test the tidiest geometric explanation — that the writable injection span and the outcome direction are misaligned — and a rank-corrected two-null audit does not support it: the boundary remains empirical and its mechanism unresolved.

We call the readable-but-unusable property **operational proto-introspection**, defined narrowly against the self-report introspection literature and claiming nothing about consciousness, self-awareness, or autonomous control. The taps read hidden trajectories, not text, and two of them are forward-looking — reading computations that have not yet resolved: the pre-answer probe, and a branch-survivability tap that predicts whether an in-flight branch will still contain a correct continuation.

We are also precise about whose failure the negative result is. The model is not consulting its own states: *we* read them, and *our* tested interventions do not establish a capability gain from what we read. Whether the model itself makes internal use of process-quality information is not tested here and is not claimed. The gap between reading and acting is the paper's result, and it makes training-time integration — a model never trained to align readable process-quality directions with writable control directions — the most direct next hypothesis, rather than a cleverer frozen intervention.

Corrected numbers throughout. Every load-bearing current quantitative claim in this paper is reported under an audited protocol — source-item-disjoint splits with a zero-crossing integrity check, and antisymmetrized evaluation for pairwise scorers (§3.7); historical or diagnostic quantities with incomplete provenance (e.g. the §3.6 math-transfer origin figures) are explicitly marked as such and are not load-bearing. The audit behind that protocol found and corrected five distorted figures across this project — four inflated and one deflated below chance; three of them in the prior published paper, formally corrected in the erratum to arXiv:2604.09870 — and its full anatomy is the subject of a companion methodology paper. Under the corrected protocol the relational preference signal is real but modest (0.565 versus 0.542 pointwise; paired Δ +0.023, 95% CI [+0.013, +0.033]), a claim that reasoning fine-tuning *installs* the readable signal does not reproduce and is retracted, and a non-looped SFT transformer reads candidate quality at 0.568 under a task-disjoint split, so this class of readout does not require recurrence.

Limitations are explicit: the pre-answer result rests on one powered domain (two further candidates were tested and rejected at preflight). On the control side, directional steering is an established negative; the bounded branch screen removes evidence for a gain without estimating a general deficit, and terminal selection remains unresolved under the clean, underpowered evaluations.

How to read this paper

This is the complete account of a four-month program, and it supports three reading depths. *The one result* — the strict pre-answer finding — is Section 5 and Figure 2; a reader with ten minutes should read those and the evidence-status map (§10.2). *The main argument* — readable but not controllable — is Sections 5, 6, and 8, with the synthesis in Section 10. *The full record* — role-specialized taps, the executable substrate, the geometric explanation tested but not supported, and the discarded mechanisms that turned into diagnostics — is the whole paper; it is written so that each negative result and each retraction is auditable, which is why it is longer than a results-only paper would be. The evaluation-integrity protocol (§3.7) governs every load-bearing current quantitative claim; historical and diagnostic quantities are marked explicitly, and the full audit anatomy is deferred to the companion methodology paper. Cross-references point forward and back so any section can be entered directly.

PART I

Setup

1. Introduction

A standard transformer processes each token position through its layer stack once: intermediate activations exist, but the state at a given position is not revisited — there is no native trajectory of repeated refinement at a fixed position before the model moves on. Looped — or universal — transformers relax this by applying the same layers repeatedly, refining a hidden state across several iterations before decoding. This produces something a single-pass model does not have: an internal *trajectory*, a sequence of intermediate computational states that the model traverses on its way to an answer. Ouro-RLTT, the frozen 2.6B looped transformer we study, exposes such a trajectory at every generation step across four loop iterations and forty-eight layers.

The existence of this trajectory makes a question concrete that is harder to pose in a single-pass model: **do these intermediate states carry readable information about the quality of the model's own ongoing computation — and if they do, can that information be turned into better outcomes?** Concretely, before Ouro-RLTT emits an answer, do its loop states already encode whether the computation currently underway is likely to succeed, which of several candidate continuations is preferable, or whether a branch is worth pursuing? And if that information is present and externally readable, can an intervention built on it — steering, branching, selecting — actually improve what the frozen model produces? This paper answers the first question yes, and finds that none of the frozen interventions we tested for the second produced a validated capability gain — an established negative for directional steering, a bounded branch screen that removes evidence for a gain without estimating a general deficit, and unresolved terminal selection under clean but underpowered evaluations. The gap between those answers is its subject; we use the shorthand "yes and no" in what follows for that fuller statement.

We are deliberate about the second question's phrasing, because the tempting version of it is wrong. We do not ask whether the *model* uses its own signal; the model is not consulting anything, and it does not know our probes exist. We ask whether *we* can use it — whether an external reader of the model's states can convert what it reads into capability through any frozen intervention. That is the question this paper answers, and §8 makes the distinction explicit.

One scope note belongs here rather than buried in a later section, because it constrains how the results should be read. We study a looped model, but we do not find that looping is what makes process-quality information readable. A conventional non-looped SFT transformer supports the same class of readout (§4.6), and the preference signal we probe is already present in Ouro's untrained base model (§3.5). What the loop supplies is the *trajectory* — an internal sequence of intermediate states, available at every position without spending output tokens — and the iterative depth that lets injected branches genuinely diverge across recurrent steps (§7.6). The readouts are a property of trained transformers; the branching substrate is where recurrence does distinctive work.

1.1 Relation to work on introspection in language models

There is a fast-growing body of work on whether language models can introspect, and it is important to state at the outset how our question differs from the one that literature asks, because we borrow its vocabulary while making a deliberately weaker claim.

The dominant paradigm operationalizes introspection through **self-report**. Binder et al. (2024) define introspection as acquiring knowledge that originates from a model's internal states rather than from its training data, and test it by finetuning a model to predict its own behavior, arguing that success implies privileged access to internal representations. Lindsey (2026) introduces the concept-injection setup — steering vectors for known concepts are injected into the residual stream, and the model is asked whether it notices an injected "thought" and what it is about — and reports that the most capable models detect such injections at modest rates with near-zero false positives, establishing accuracy, grounding, internality, and metacognitive representation as criteria for the reported signal. Comşa and Shanahan (2025) sharpen the conceptual bar, arguing that genuine introspection requires a *causal* connection between the internal state and the model's report of it, so that verbal mimicry of introspective language is insufficient. Subsequent work extends this paradigm to open models and probes its mechanisms: Pearson-Vogel et al. (2026) show that a model's residual stream reveals detection of a prior concept injection even when its sampled text denies it, and that detection requires reading information cached from earlier tokens.

Every method in this cluster shares a common shape: a representation is *manipulated* (by injection or finetuning), the model is *asked to report*, and the report is *validated* against criteria of causal grounding. This is introspection as self-knowledge that the model can articulate.

Our question is different along both axes. We do not manipulate the model's representations with foreign concepts, and we do not elicit or evaluate any self-report. Instead, we read *naturally-arising* intermediate states — the states the model produces in the ordinary course of solving a task — with small external probes, and ask whether those states contain information about the quality of the model's own ongoing computation. We never ask the model what it is thinking; we measure what its process states reveal to an outside reader, and then we ask whether an external intervention built on that reading can improve what the frozen model produces. This is a weaker property than self-report introspection: it makes no claim that the model has access to, represents, or can articulate its own states. To mark both the kinship and the distance, we call it **operational proto-introspection** — a readout-side precursor to, and not an instance of, the self-report introspection the above work investigates. Positively and compactly: *a hidden state is operationally proto-introspective if an external reader can recover, from that state alone, information about the quality or likely outcome of the model's own ongoing computation before that computation resolves.* We introduce the term only in this narrow operational sense here, and defer its full definition and defense until after the evidence is on the table (Section 9), because the word carries strong connotations that the evidence does not underwrite and we would rather earn it than assume it.

Two results from the self-report literature are worth flagging as directly relevant rather than merely adjacent. The finding that a model's residual stream carries injection-detection information its own output denies (Pearson-Vogel et al., 2026) is an independent demonstration that hidden states can contain more about a model's situation than its outputs report — the same premise our readouts rest on, generalized here from injected-concept detection to naturally-arising process quality. And that this hidden signal is recoverable from *cached* representations connects to the executable substrate we develop, in which readouts are computed over the model's own key/value cache during generation.

1.2 Reading is not controlling

A paper that only established readable process-quality signals would be a probing paper, and it would be one easy question away from its central weakness: *so what — can anything be done with them?* The contribution here is that we can answer that question, and the answer is a boundary. We build not only the readouts but an executable internal branching substrate — autoregressive, branch-specific key/value-cache carry with a bit-exact suffix-recompute splice — so that the readouts can be installed *inside* the generation loop and used to select among branches at run time, rather than analyzed offline. With that machinery in hand, we test whether the readable signals confer control, and find that they do not confer a validated gain: directional steering across seven methods is an established negative, and branch injection deconfounded against matched sampling produces no gain in a bounded screen. The direction that *predicts* success is not a direction that, written back into the model, *produces* success. We call this the **readout–control boundary**, and it is the paper's load-bearing result. Notably, it is an *empirical* boundary: we test the tidiest available explanation — that the writable branch directions and the outcome-relevant directions occupy different subspaces — and, under a rank-corrected two-null audit, cannot support it, which sharpens rather than resolves the question of why the tested frozen conversions do not deliver a gain.

A negative result is only worth reporting if the experiment that produced it could have come out otherwise, and three conditions here make that the case. First, **the signal is genuinely there**: the taps identify oracle-containing branches at 0.9697 retention (task-disjoint), detect generated-branch correctness at AUROC 0.7755, and predict the model's own success before it answers. Failure to steer is not failure to read. Second, **the machinery genuinely works**: the branch substrate is validated by bit-exact identity checks — a zero-perturbation fork reproduces the reference exactly at prefill, the suffix-recompute splice is bit-exact across all 192 cache slots, and omitting the carry produces the expected large divergence. Failure to steer is not a plumbing bug. Third, **the obvious confounds are controlled**: the apparent gains

from sampled branch injection dissolve against K-matched plain sampling, so what remains is not sampling luck. The negative survives the conditions under which a positive would have been believed. That is what makes it a boundary rather than an absence.

1.3 Contributions

1. **Pre-answer prediction of the model's own success.** On GSM8K, hidden states predict whether the model's in-progress computation will succeed *before the answer exists*, adding significant information beyond length and log-probability shortcuts (incremental AUROC +0.066, 95% CI [+0.021, +0.112], paired task-clustered bootstrap; leave-one-task-out stable). The strict cut excludes the answer region and the gold value in code. This is the paper's primary positive result and the empirical core of the proto-introspection framing.
2. **The readout–control boundary.** Readability does not confer validated control in the frozen model: directional intervention across seven steering methods is an established negative; the bounded four-task branch screen removes evidence for a frozen-fork gain but is too small to estimate a general deficit; and selective intervention (forced terminal choice) remains unresolved under the clean, underpowered evaluations. A bounded 300-step LoRA probe additionally shows that modest adaptation changes branch diversity and parse behavior without automatically improving aggregate reachability — evidence against a trivial one-run fix, though not a closure over light training generally. A rank-corrected two-null subspace audit does not support the simplest one-dimensional span-misalignment explanation, leaving the boundary empirical and its mechanism unresolved, and motivating training-time integration as the most direct next hypothesis.
3. **An executable internal branching substrate, validated by exact identity.** Autoregressive branch-specific KV-cache carry across Ouro's 192-slot recurrent cache (4 loops × 48 layers), established through a six-level correctness ladder — independent branch caches with no cross-contamination, batched equivalence, lineage-preserving prune/reorder, and a negative control confirming the carried cache is load-bearing (withholding it diverges to RMS ≈ 3.0). On top of it, a **bit-exact suffix-recompute splice**: because the KV cache stores keys and values but *not* the inter-layer residual stream, a perturbed branch normally forces a full re-prefill; capturing the residual at the perturbation boundary makes the perturbed state reconstructible with no forward pass, so only the affected suffix is recomputed — saving up to **88%** of per-branch layer passes while matching a full recompute bit-for-bit across all 192 slots. Branch/fork machinery over standard KV caches is well established (PagedAttention's copy-on-write, SGLang's RadixAttention, SpecInfer's tree attention), and depth-recurrent models already reuse KV across recurrent steps (Geiping et al., 2025; Zhu et al., 2025); the specific piece we have not found in prior work is the residual-capture suffix splice that makes a *mid-computation-perturbed* branch bit-exactly reconstructible over the loop×layer recurrent cache without a re-prefill. This is what makes the negative result credible — the machinery demonstrably works, and no frozen intervention through it produced a validated capability gain — and it is independently useful for search in looped architectures.
4. **Role-specialized readouts that work, and an architecture control that constrains their interpretation.** The readable signal decomposes into distinct taps — branch survivability (**0.9697** oracle retention under a task-disjoint split, with the layer-47 locus causally load-bearing: ablating the channel the tap reads collapses retention to **0.0417**), content ranking (**0.6310** task-disjoint across five domains), and generated-branch correctness (AUROC **0.7755**) — each recovered by a low-capacity probe on frozen states and consumed by a live branching scaffold. A task-disjoint domain-transfer study shows specialization pays where distinctions are hard (code-trained taps read coding at **0.953** against a general tap's 0.694) and is unnecessary where a general quality axis suffices (a balanced generalist matches the reasoning specialist). And a **non-looped SFT transformer supports the same class of readout** (0.568 macro top-1, task-disjoint), so recurrence is *not* necessary for process-quality readability — a constraint on how these results should be read, and one we report against our own framing's interest.

1.4 Roadmap

Part I fixes notation and describes Ouro-RLTT as a hidden-state substrate (Section 2). Part II establishes the readout side, building to the paper's primary positive result: preference structure, reported under the corrected evaluation protocol that §3.7 states (Section 3), role-specialized readouts and the non-looped architecture control (Section 4), the strict pre-answer success-prediction result (Section 5 — **the paper's headline; a reader with limited time should start there**), and generated-branch correctness, which pivots toward the control problem by showing that a decodable — and even retainable — signal does not become a reliable commitment (Section 6). Part III presents the executable branch/carry/prune substrate (Section 7). Part IV presents the readout–control boundary, where the negative results land against the backdrop of both the readouts and the machinery (Section 8). Part V defines operational proto-introspection against the self-report literature, synthesizes the evidence, states limitations — foremost that the pre-answer result rests on a single powered domain — and lays out the training-time integration the frozen boundary motivates (Sections 9–13).

1.5 Relation to the prior project paper

This work builds directly on Kirin (2026a), which found that Ouro-2.6B's loop states encode human preference predominantly relationally, and introduced the separable, frozen-backbone evaluator paradigm (a lightweight ~5M-parameter head read off a frozen backbone) that we adopt throughout.

We correct that paper as well as extend it. Three of its reported figures — the 84.5% relational linear probe, the 21.75% pointwise linear probe, and the 95.2% nonlinear evaluator — do not survive audit: the first two were inflated by a leaked evaluation split, the third by a canonical-ordering prior (§3.3, §3.7). The prior paper's *direction* holds — preference is decoded more accurately relationally than pointwise — but its magnitudes do not, and its strongest claim, that preference is *unavailable* pointwise, is withdrawn: a clean pointwise probe reads 0.5418, significantly above chance. The corrected contrast is 0.5653 versus 0.5418 (paired Δ +0.0234), not 0.845 versus 0.2175. The correction of record is the erratum to Kirin (2026a, arXiv:2604.09870v2); §3 here reports the corrected science, §3.7 states the protocol, and the full audit anatomy — both mechanisms, all five corrected figures, synthetic demonstrations, and tooling — is the companion methodology paper (Kirin, 2026b).

Beyond the correction, we extend the prior work in three directions it did not address: from a single preference signal to role-specialized process-quality readouts (Section 4), from candidate comparison to pre-answer prediction of the model's own success (Section 5), and from offline readout to the question of frozen control through an executable branching substrate (Sections 7–8). Where this paper reuses the prior setup — the HH-RLHF data, the hidden-state extraction, the evaluator head size, the epoch-2 checkpoint selection, and the antisymmetry-enforcement training protocol and its metric-deflation caveat — we note the reuse and cite Kirin (2026a) rather than re-deriving it.

The empirical bridge between the two papers was less tidy than the final section structure might make it look. Applied unchanged to a wrong domain — selecting among Ouro's candidate continuations on 100 Hendrycks MATH problems, a task whose labels are not preference labels — the HH-trained evaluator picked a correct-answer-containing continuation far more often than single-shot Ouro under the same harness. We do not treat that result as load-bearing (its exact figures, and the truncation confound that followed it, are detailed and caveated in §3.6); what mattered was the implication that the evaluator was reading something more general than an HH-specific preference artifact. That changed the question. The object of study became whether Ouro-RLTT's looped hidden states expose a broader family of process-quality signals whose geometry overlaps across preference, content, reasoning, correctness, and branch viability. Section 3 reconstructs the relational primitive; Section 4 shows why the mature answer is role-specialized taps rather than one universal evaluator.

1.6 Relation to latent reasoning and internal search

The branching substrate we build (Section 7) sits alongside two neighboring lines of work it should not be conflated with, and we state the distinctions here so later sections can reference them rather than repeat them. We emphasize at the outset that this is a positioning, not a performance comparison: the paper's substrate result is a *negative* one about frozen control, not a claim to outperform any search method.

The first neighbor is external, token-level search — tree-of-thought (Yao et al., 2023), beam search, and best-of-N — which wraps repeated model calls around sampled text and selects among decoded candidates. Our branching is not a text-level wrapper; it operates *inside* the model's own loop/layer key-value cache, forking and carrying per-branch state rather than re-invoking the model on strings. The performance relationship to best-of-N is, moreover, not left open: a K-matched plain-sampling comparison (which is best-of-N with oracle selection) is precisely the deconfound against which our frozen branching shows no gain (Section 8.2). We do not claim to beat best-of-N; matched best-of-N is the baseline our central negative result is measured against.

The second neighbor is latent reasoning in non-looped models, of which Coconut (Hao et al., 2024) is the clearest instance: it feeds a model's last hidden state back as the next input embedding rather than decoding it to a token, yielding a continuous "thought" that can hold several candidate next steps in superposition and explore them breadth-first before committing. Two differences matter. Coconut's branching is *implicit and superposed* within a single continuous trajectory and is *trained in* by a multi-stage curriculum; ours forks *explicit*, separately cached trajectories that are carried, scored, and pruned as distinct objects, on a *frozen* backbone. And the exploration lives in a different place: Coconut induces a latent trajectory along the *sequence* by spending token positions, whereas our branches diverge along the model's *depth*, across loop iterations — a distinction that turns out to be the substantive architectural reason the substrate suits a looped model specifically, developed in Section 7. The trained-vs-frozen contrast also bears directly on our boundary result and its proposed resolution (Sections 8, 12).

2. Ouro-RLTT as a Looped Hidden-State Substrate

2.1 The model

Ouro-RLTT is a 2.6B-parameter looped (universal) transformer: a fixed stack of forty-eight transformer layers applied over four *loop iterations*, so that a single token position is processed by the same weights four times, with the hidden state carried forward between iterations. Reasoning-oriented training (the RLTT variant) encourages the model to use these iterations to refine intermediate computation before committing to output. We use the model entirely **frozen**: no weight of the backbone is updated anywhere in this paper except in one explicitly labelled bounded-LoRA probe (Section 8), whose purpose is to test a boundary, not to improve the model. Every readout in this work is an external probe trained on top of frozen activations.

The relevant consequence of the looped design is that each generation step yields not a single hidden vector per layer but a small **trajectory**: the same position revisited across four loop iterations produces four successive hidden states per layer, and it is this trajectory — the model's own iterative refinement of its computation — that we read.

2.2 Notation

We fix notation used throughout the paper.

- $h \in \mathbb{R}^D$ denotes a hidden-state *feature vector*. Unless otherwise stated, a feature is formed by pooling and concatenating loop states across a fixed set of tapped layers and loop iterations; in the primary configuration we tap three layers across four loop iterations at hidden width 2048, giving a feature dimension $D = 3{\times}4{\times}2048 = 24{,}576$. The three tapped layers are **24, 36, and 47**, over loop indices L1–L4, with pooling variants (mean, final-loop L4, or loop-concatenation) selected per role. One family of experiments uses a different basis: the powered HH preference probes of §3 read post-final-norm states at the four *loop boundaries* only ($4 \times 2048 = 8{,}192$), because the question there is about the loop trajectory rather than about layer-localized roles. Full extraction detail is in Appendix A.
- h_A, h_B denote feature vectors for two candidate continuations or branches A and B.
- $\Delta h = h_A - h_B$ denotes the pairwise hidden-state difference. Preference-relevant information is decoded *more accurately* from Δh than from h_A or h_B individually (0.565 vs 0.542; paired $\Delta = +0.023$, §3.2), which is why comparison taps in this work operate on differences. The advantage is real and modest; an earlier version of this work reported it as far larger, and that figure is retracted (§3.7).
- U_{inj} denotes the subspace spanned by the S1/S3 frozen injection/carry deltas — the directions the frozen branch mechanism can actually *write* into the residual stream — with $P_{U_{\text{inj}}}$ the orthogonal projector onto that span.
- d_{out} denotes the verifier-success outcome direction: the hidden-state direction separating verifier-correct from verifier-incorrect continuations.
- The **projection fraction** of the outcome direction into the injection span is

$$\pi_{\text{inj}}(d_{\text{out}}) \;=\; \frac{\|P_{U_{\text{inj}}}\, d_{\text{out}}\|^2}{\|d_{\text{out}}\|^2} \;\in\; [0,1],$$

  read against the random-direction baseline k/D where $k = \dim U_{\text{inj}}$ and D is the ambient feature dimension (Section 8).

We also distinguish a **candidate group** (a set of alternative continuations to be compared or selected among) from an **oracle-present group** (a set for which verifier/gold correctness labels are available for evaluation), and use "verifier-correct" for continuations a task-specific checker accepts and "gold" for reference answers.

2.3 The cache substrate

Ouro-RLTT's key/value cache is organized by both loop iteration and layer. We index cache state by a flattened slot $\text{slot} = u \cdot 48 + \ell$ for loop iteration $u \in \{0,1,2,3\}$ and layer $\ell \in \{0,\dots,47\}$, giving $4 \times 48 = 192$ cache slots per position. This organization is what makes the executable branch/carry substrate of Section 7 possible: a branch is a distinct trajectory through these 192 slots, and forks, carries, prunes, and reordering all operate over this indexed cache. We defer implementation detail to Section 7 and Appendix F, and note here only that the cache structure — loop × layer × position — is the object the readouts are computed over and the object the branch machinery manipulates. The two halves of the paper, readout and control, read and write the same substrate.

2.4 What we do and do not assume

We assume only that (i) the model's loop trajectory is a meaningful object to read, which the readout results justify post hoc, and (ii) the frozen backbone's behavior is stable and reproducible, which we verify by bit-exact identity checks on

the branch machinery (Section 7). We do **not** assume that readable information is causally used by the model, that the model can report on its states, or that any readout direction is a control direction — indeed, the central negative result of the paper is that the last of these fails.

2.5 Operational terms

Several terms carry specific, consistent meanings throughout the paper; we collect them here for readers outside this project's vocabulary.

- **Readout / tap:** an external probe trained on frozen hidden states to predict a property of the model's computation or of candidate continuations. Readouts never modify the model.
- **Process-quality signal:** readable information about the *quality* of computation — likely success, stability, preference, content quality, branch survivability, or generated-branch correctness — as opposed to the model's current answer.
- **Branch:** an alternative continuation, or hidden-state trajectory, derived from the same prompt and computation via a fork at a chosen boundary.
- **Carry:** preserving a branch's own key/value cache state across subsequent computation, rather than restarting the branch from text alone.
- **Survivability:** whether a branch should remain in the search pool because it may still lead to a correct (oracle) continuation.
- **Selection:** forced choice of a single final branch or answer under commitment.
- **Steering:** direct intervention on hidden states along a learned or readout-derived direction during generation.
- **Readout vs. control (actionability):** *readout* is the ability to read a signal externally; *control / actionability* is the ability of an intervention or branch policy to reliably improve final behavior. The paper's central finding is that the first does not, in the frozen model, imply the second.

PART II

Readout

3. Relational Preference Structure

This section establishes the readout side's foundation: preference-relevant structure is linearly decodable from Ouro's loop states, and more accurately from comparisons than from individual representations. Every load-bearing current quantitative claim here is reported under the corrected evaluation protocol stated in §3.7; historical or diagnostic quantities are marked explicitly — several published and draft-stage figures on this topic were distorted by evaluation artifacts, and the corrected effects are real and modest. Readers interested in *how* the original numbers went wrong, and in the general lessons, should read the companion methodology paper; readers interested primarily in this paper's main results may read §3.7's protocol statement and skip to Section 5.

3.1 Setup and the two probes

We train probes to predict human preference labels (from HH-RLHF preference pairs; Bai et al., 2022) from frozen hidden features. Two probe families are compared under an identical protocol:

- a **relational** probe on the pairwise difference $\Delta h = h_A - h_B$, scoring $s(A, B) = w^\top(h_A - h_B)$ with no bias term, so that $s(B, A) = -s(A, B)$ exactly; and
- a **pointwise** probe on a single pooled representation h_A, asked to classify one candidate as chosen or rejected without seeing its partner.

Both use bias-free L-BFGS logistic readouts over mean-pooled representations from four post-final-norm loop boundaries (feature width $4 \times 2048 = 8{,}192$), on 40,000 HH source pairs under a strict **pair-disjoint** split (32,000 train / 8,000 evaluation source pairs). The pair-disjointness is essential and is the subject of §3.7.

3.2 Preference is decodable, and relational decoding is more accurate

On Ouro-2.6B-Thinking, held out on 8,000 unseen source pairs:

Probe	Held-out accuracy	95% CI
Relational (pairwise difference)	**0.5653**	—
Pointwise, final boundary	0.5462	[0.5411, 0.5513]
Pointwise, all four boundaries	0.5418	[0.5366, 0.5471]

Because both probes were evaluated on the *identical* held-out pairs, the correct comparison is paired. The relational probe's advantage over the matched pointwise classifier is $\Delta = +0.0234$**, 95% CI [+0.0132, +0.0334]** — significant, and modest. The same conclusion holds under a stricter framing: the pointwise classifier's scores can themselves be used to *rank* the two candidates in a pair, which yields 0.5554 pairwise accuracy; the dedicated relational probe still beats that, by +0.0099 (95% CI [+0.0004, +0.0195]).

Two claims follow, and one prior claim does not.

Supported: preference-relevant structure is linearly decodable from Ouro's loop states, and it is decoded more accurately relationally than pointwise. This is why every comparison tap in this work operates on Δh.

Not supported: that preference is *unavailable* pointwise. Earlier versions of this work — and the prior project paper (Kirin, 2026a) — reported a pointwise linear accuracy of 21.75%, below chance, and concluded that the absolute channel is not usably present. That figure was produced under a leaked split (§3.7) and does not survive correction: the clean pointwise accuracy is 0.5418, significantly *above* chance. We withdraw the strong claim. Preference is accessible both pointwise and relationally; the relational route is simply better, by about two points.

We also note what these magnitudes are not. At 0.54–0.57, none of these probes is competitive with a trained reward model on this data (typically 0.72–0.75; Lambert et al., 2024). The claim here is about the *organization* of preference information in a frozen model's hidden states — that it is relationally structured — not about achieving competitive preference prediction.

3.3 A high-accuracy fixed-order evaluator, and the first audit

A higher-capacity nonlinear evaluator — attention pooling over the trajectory, per-loop differences, a trained difference-LayerNorm, a GRU across loops, and a nonlinear scorer — trained to compare two candidates in a **fixed** presentation order (chosen always first) reaches roughly **95% test accuracy** (95.2% = 8,141/8,552 in canonical order). Kirin (2026a) reported this evaluator and read the number positively: as evidence that a nonlinear model surpasses the linear probe. It does not. A fixed-order pairwise evaluator can score highly by learning a presentation-order prior rather than relational discrimination, and this one did.

On the **full 8,552-pair HH-RLHF test set**, the evaluator's fixed-order accuracy is **0.9479** (95% CI [0.9431, 0.9525]) — reproducing the historical 0.9519, which lies inside this interval — but its **strict antisymmetrized accuracy**, the fraction of pairs on which the order-independent component of its score has the correct sign, is **0.6392** (95% CI [0.6291, 0.6493]). Roughly a third of the apparent accuracy was the ordering prior: a large learned first-position offset (the symmetric score component is on average 1.50× the antisymmetric one; 75.3% of pairs are scored "prefer the first argument" in both orders), not a degenerate constant — content still matters (normal/flipped score correlation −0.925), but the offset overwhelms the sign for most pairs. The mechanism's full decomposition, its pooling and normalization ablations, and a synthetic reconstruction of the effect are given in the companion methodology paper; the audit artifact is summarized in Appendix B.

Antisymmetrization is therefore a mandatory audit for any fixed-order pairwise evaluator: fixed-order accuracy is not, by itself, a measure of relational discrimination. We report the historical 95.2% only as fixed-order, discovery-stage accuracy.

3.4 What survives, and how the results now order

Under honest, held-out evaluation the three preference numbers order as follows:

Reader	Clean accuracy	Protocol
Nonlinear evaluator, strict antisymmetrized	**0.6392**	full 8,552-pair disjoint test set
Linear relational probe	0.5653	40k pairs, pair-disjoint
Linear pointwise probe	0.5418	40k pairs, pair-disjoint

The nonlinear evaluator's *relational* component (0.639) is the strongest clean preference readout in the project — the additional capacity does buy genuine relational discrimination, once the order artifact is removed. This reverses a claim made in an earlier version of this work, which asserted that the linear probe (then reported at 84.5%) *outperformed* the

antisymmetrized nonlinear evaluator. That inversion was an artifact of the leaked linear number; on clean splits, the ordering runs the other way.

This also revises the prior paper. Kirin (2026a) is correct that preference is encoded relationally rather than absolutely — the paired relational-over-pointwise advantage survives on clean data — but the magnitudes it reports (84.5% relational, 21.75% pointwise, and a 95.2% nonlinear headline) are all artifacts of the split and ordering defects documented here — two inflated, one deflated below chance. The corrected contrast is 0.5653 versus 0.5418, not 0.845 versus 0.2175. The direction of the prior paper's central finding stands; its size does not.

3.5 Training increases the linear preference signal, modestly

An earlier version of this work claimed a training-stage *localization*: that a fixed evaluator read Ouro-2.6B-Thinking at 95.2% and Ouro-RLTT at 95.0% but collapsed to 24% on the base model, and that reasoning fine-tuning therefore *installs* the readable signal. **That claim is retracted.** The historical base=24% cell has no surviving artifact and does not reproduce: the same saved evaluator, applied to the pinned base checkpoint under controlled preprocessing, returns ~95% canonical accuracy on base as well (95.0 / 95.0 / 94.5 across base / Thinking / RLTT, with flat antisymmetrized accuracy 58.0 / 59.5 / 60.0). There is no base-specific collapse. The likely cause of the original figure — a mismatched checkpoint, extraction locus, orientation convention, or transcription error — cannot be established without the artifact, and we do not speculate further.

The question the retraction leaves open — *does training change the readable preference signal at all?* — is answerable, and we answer it properly. Applying the clean pair-disjoint linear protocol of §3.1 to all three backbones on the identical 8,000 held-out pairs:

Backbone	Held-out accuracy	95% CI
Base Ouro-2.6B	0.5553	[0.5444, 0.5663]
Ouro-2.6B-Thinking	0.5653	[0.5543, 0.5760]
Ouro-RLTT	**0.5698**	[0.5586, 0.5804]

The marginal intervals overlap, but that is the wrong test: all three probes were evaluated on the *same* held-out pairs by construction, so the comparison is paired and the correct interval is a bootstrap over per-pair differences, which removes the pair-difficulty variance shared across backbones.

Comparison	Δ	95% paired CI	p	Significant (Bonferroni $\alpha = 0.0167$)
RLTT − Base	**+0.0145**	[+0.0043, +0.0248]	0.0046	**yes**
Thinking − Base	+0.0100	[−0.0001, +0.0201]	0.0546	no
RLTT − Thinking	+0.0045	[−0.0015, +0.0106]	0.1604	no

What this supports: the full training pipeline increases the linearly-decodable preference signal. RLTT reads higher than base by 1.45 points, and that ordering survives correction for multiple comparisons.

What it does not support: any claim that reasoning SFT *specifically* installs the signal. The Thinking-vs-base comparison is borderline and fails at 95% ($p = 0.055$); the RLTT-vs-Thinking comparison is indistinguishable from noise ($p = 0.16$). We therefore attribute the effect to the training pipeline as a whole, not to a stage within it. And crucially, the signal is **present in the base model** (0.5553, well above chance): training enriches a preference direction that pretraining already provides. The earlier framing — that the loop provides the substrate and reasoning training installs the content — is not supported and has been removed throughout.

The effect is small. A 1.45-point gain on a 55-point base is a real ordering, not a large one, and we present it as such.

3.6 The math-transfer shock that started the broader program

The transition from the prior relational-preference paper to the present work began as an empirical surprise rather than a preplanned theory of domain transfer. After training the HH-RLHF evaluator, we applied it, unchanged, to select among Ouro's candidate continuations on 100 Hendrycks MATH problems (Hendrycks et al., 2021) under a fixed harness. This was, in effect, a "wrong-domain" application: the evaluator had not been trained on math, proof search, answer checking, or verifier correctness, and a narrow preference-reader interpretation would predict either noise or a weak style/preference bias.

Instead, the evaluator's selected continuation contained the correct final answer on **47 of 100** tasks. This figure is a floor. On most of the remaining 53 tasks the evaluator did not choose a wrong answer; Ouro simply never produced

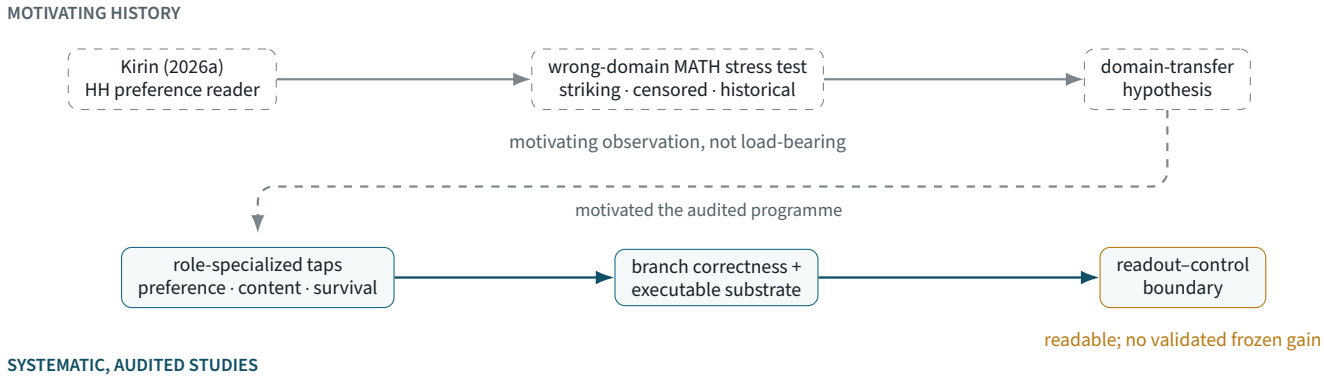
a parseable one — it continued generating without committing to an answer — so those tasks are censored rather than failed, and the true selection accuracy given a scorable candidate is unknown and higher. Under the *same* harness, single-shot Ouro reached a correct answer roughly **2.7**× less often. Because the evaluator-guided selection and the single-shot baseline run under the identical harness and share Ouro's tendency to over-generate, this censoring affects the compared quantities alike, and the 2.7× is a fair same-harness comparison of how often each recovers a correct answer. The exact artifact for this historical run is unarchived; we flag the precise denominator and baseline definition for live-repo pinning and do not treat the multiplier as load-bearing evidence.

The result does not isolate mathematical understanding: part of the effect is best-of-selection simply having more chances to reach a parseable answer, and part may reflect the evaluator preferring complete over unfinished candidates. And it is distinct from — and was followed by — a sharper confound. When we tried to *build* on the result with broader math pilots, generated attempts were often verbose and truncated by token limits, truncation correlated with correctness, and a trained selector could learn "not truncated" as a proxy for "correct" rather than mathematical correctness itself; those pilots were demoted in favor of clean GSM8K with exact numeric parsing (§5, Appendix I). The clean origin result and the later confound are two different events: the same-harness 2.7× ratio is robust to the truncation-leakage problem that killed the broad pilots, which is why it survived as the motivating observation when they did not.

The important point for this paper is not the precise historical multiplier but that a preference-trained hidden-state reader transferred at all to a domain whose labels were not preference labels — surprising enough, and confounded enough, to motivate the auditable domain-transfer, tap, and pre-answer studies that follow. That was the first reason to suspect the evaluator was not merely memorizing HH-specific preference artifacts, but reading a broader latent geometry of candidate quality.

In hindsight, this surprise is the hinge between Kirin (2026a) and the present paper. It motivated the move from a single, high-capacity preference evaluator to a family of smaller, role-specialized taps. The right follow-up was not to declare the HH evaluator a universal reward model; the antisymmetry audit in this section is exactly why that would be too strong. The right follow-up was to ask which parts of the hidden trajectory support which kinds of quality judgments: preference, content relevance, survivability, generated-branch correctness, and pre-answer success. Section 4 is the systematic version of that question.

The path from the prior paper to the present one, and the structure it induced, is summarized below.



Project lineage. An unexpected out-of-domain transfer result (§3.6) reframed a single preference evaluator as a reader of a broader process-quality geometry, motivating role-specialized taps (§4), the branching substrate (§7), and the negative control results (§8).

3.7 Evaluation integrity: the corrected protocol

Every load-bearing current quantitative result in this paper is reported under a protocol adopted after a project-wide audit — historical or diagnostic quantities with incomplete provenance (e.g. the §3.6 math-transfer figures) are explicitly marked as such and are not load-bearing. The audit found five distorted figures — four inflated, one deflated below chance; three in the prior published paper (corrected in the erratum to arXiv:2604.09870), two in earlier drafts of this one — produced by two distinct and mutually invisible mechanisms. This subsection states the protocol and the corrections that affect this paper's results; the mechanisms' full anatomy, the phenomenology of how they hid, synthetic demonstrations, and reusable audit tooling are the companion methodology paper (Kirin, 2026b).

Mechanism 1 — source-item leakage. When a dataset is built by constructing multiple rows from each source item (± difference orientations, chosen/rejected singletons, candidate families, branches per task), splitting those constructed

rows independently leaks the source item across the train/test boundary. Held-out behavior becomes dependent on recognition of source items seen during training; depending on construction geometry, that dependence can inflate accuracy or systematically invert predictions below chance. Notably, exact antisymmetry — a structural property we had relied on as a safeguard — *accelerates* this leak rather than preventing it: a probe that memorizes $w^\top \Delta_i$ from a training row scores its exact negation correctly at evaluation *because* it is antisymmetric. The artifact need not look suspiciously good: the same defect produced both an inflated 0.845 and a below-chance 0.2175 that an earlier paper interpreted as a finding ("inverted polarity").

Mechanism 2 — presentation-order exploitation. A pairwise evaluator trained and evaluated with its candidates in a fixed order can score highly by learning "prefer the first argument." Its data are properly split — no split check can see this; what crosses the boundary is a label, through the presentation (§3.3).

The protocol, used for every load-bearing current quantitative claim in this paper:

1. **Split on source items, never constructed rows** (the HH pair, the task), and enforce an explicit integrity check that counts source items crossing the boundary and refuses to run at anything other than zero.
2. **Antisymmetrize every fixed-order pairwise evaluation**: score both presentation orders and evaluate the order-independent component. Flip-test correlation alone cannot catch an order prior (§3.3).
3. **Power the clean evaluation adequately**: a source-disjoint split with $p \gg n$ returns chance whether or not signal exists (our first pair-disjoint audit, at 800 training pairs against 8,192 features, read 51% on every backbone and was uninformative until rerun at 32,000 pairs).
4. **Audit call sites, not results**: a row-splitting helper applied to constructed data is a bug in a function, not in a result — enumerate every place it touches constructed rows and audit all of them at once, before deciding which numbers to trust.

Corrections affecting this paper's results, each re-verified under a zero-crossing split:

Result	Mechanism	Reported	Corrected
Relational linear probe (§3.2)	orientation rows leaked	0.845	**0.5653**
Pointwise linear probe (§3.2)	pair partners leaked (74% of eval rows)	0.2175 (below chance)	**0.5418**
Fixed-order evaluator (§3.3)	canonical-ordering prior	0.952	**0.6392** (antisymmetrized)
CoreContent v2 (§4.5)	195 task IDs crossing	0.6691	**0.6310**
Branch survival (§6.2)	8 task IDs crossing	0.9848	**0.9697**

Where a genuine signal remained, it survived correction, although the direction and magnitude of the correction varied substantially; one separate training-stage claim did not reproduce and is retracted. Source-item leakage generalizes to evaluations constructed from multiple rows, variants, or candidates of a shared source. Presentation-order exploitation generalizes to pairwise scorers trained, evaluated, or selected under a canonical candidate order. The structural protection is not care but the integrity checks above.

4. Role-Specialized Hidden-State Readouts

Section 3 treated preference as a single relational signal. The second readout finding is that the readable content of the hidden trajectory is **not one scalar quality score** but a set of related yet distinguishable signals, each recoverable by its own low-capacity tap, and each localized to particular layers and loop iterations. We refer to these as *role-specialized readouts*.

4.1 Tiny taps on a frozen backbone

Each readout is produced by a small head — on the order of a few million parameters — trained on top of the frozen Ouro-RLTT trajectory; the backbone is never updated. That such small heads suffice is itself part of the claim: the information is present in the hidden geometry in a form a low-capacity reader can extract, rather than requiring a large external model to manufacture.

The move from the original evaluator to taps was therefore both scientific and methodological. The original HH evaluator was large enough to reveal that useful structure existed, and its GRU-over-loops design was useful for asking whether the refinement trajectory carried information. But the same capacity also made it able to exploit presentation-order regularities under fixed-order training. The tap program deliberately moved in the opposite direction: small heads, explicit pairwise differences, bias-free scoring, and swap-safe antisymmetry by construction. This made the readouts less expressive but easier to audit. A tap that succeeds under these constraints is stronger evidence that the information is

present in the hidden-state geometry rather than manufactured by the external evaluator. Tap architectures and training details are given in Appendix C.

4.2 A decomposition into roles

Across training targets we find distinct, separately-decodable readouts spanning at least:

- **preference** — which of two candidates is preferred (Section 3);
- **content quality** — task-relevant quality of a single continuation's reasoning/content, as distinct from mere preference ordering;
- **branch survivability** — whether a branch is likely to persist rather than be pruned under the scaffold's own dynamics (§6.2);
- **generated-branch correctness** — whether a generated continuation is verifier-correct (Section 6).

These are related but not identical: a tap trained for one role does not transparently solve another, and the roles localize differently across the loop×layer grid. Two of these readouts are carried by tap families built for different pipeline stages by deliberately different methods — a **DualAnchor** family that prunes and retains branches *during* the looped branch/prune search, and a **CoreContent** family that ranks candidates *within* an already-handed-off survivor set. How each was constructed, and why the difference between them is a difference of stage rather than of "aspect," is the subject of §4.5; detailed metrics and the S3B2 relationship are in Appendix E.

4.3 Localization across loops and layers

The readouts are not uniformly distributed across the trajectory. Preference and comparison signals concentrate at particular tapped layers and loop iterations rather than appearing equally at every layer of the recurrent backbone. The canonical 24/36/47 basis was chosen from the earlier locus work, not from an architectural prior or from the final headline results. The project first ran pairwise locus and loop ablations on the HH evaluator (v2–v4), then normalization and bias-decomposition passes showed that the useful signal was relational and mid/late rather than tied to a single absolute state. Later all-layer and cached probes across coding, reasoning, and logic (v7), followed by evaluator-placement and multi-tap ensemble experiments (v8–v9), narrowed the useful region to mid-to-late decoder layers. The v10 Thinking-vs-RLTT loop-geometry pass then made layers **24, 36, and 47** the stable compact basis used by the later architecture-looped and DualAnchor baselines.

Operationally, the three layers serve different points along the same refinement trajectory. Layer 24 acts as a mid-depth representation before final consolidation, layer 36 as a late integration point, and layer 47 as the terminal/pre-output boundary where loop-to-loop spread and comparison signal were most consistently useful. We therefore use the 3 layers × 4 loops × 2048-dimensional feature basis as the default readout substrate, rather than storing all 48 layers at all loop steps. In the branch-survival line this basis was further turned into an explicit per-loop schedule, `L1_24 -> L1_36 -> L1_47 -> ... -> L4_24 -> L4_36 -> terminal L4_47`, which is why the same three loci recur in both the offline taps and the live branch/carry scaffold. Layer/loop localization per role is summarized in Appendix D and the extraction detail in Appendix A; role-specific variants may use mean pooling, final-loop L4 features, L1/L4 fusion, or full loop-concatenation.

4.4 Domain structure and the specialist finding

Do the readouts generalize across task domains, or does each domain need its own tap? We answer this on a deterministic **task-disjoint** split (zero task IDs crossing the boundary), with held-out sets ranging from 360 coding groups to 4,748 alignment groups and task-clustered bootstrap intervals.

Tap trained on	Evaluated on	Top-1	Pairwise
Code	coding	**0.9528**	**0.9650**
HH	coding	0.6944	0.8727
HH	alignment	**0.6902**	0.6831
Code	alignment	0.5609	0.5538
Reasoning	reasoning	0.7671	0.8870
Balanced (all-core)	reasoning	**0.7613**	0.8827
Random-20 HH subset	alignment	0.6038	0.5968

Four things follow. **Code specialization is strong**: a code-trained tap reads coding at 0.9528 against a general HH tap's 0.6944 — the largest specialization effect in the study. **Alignment specialization is real but smaller**: HH-trained 0.6902 versus code-trained 0.5609. **Reasoning does not need a specialist**: a balanced all-core generalist (0.7613 / 0.8827) essentially matches the reasoning specialist (0.7671 / 0.8870), so one general head suffices where the distinction is not

adversarial. And **data scale matters independently of domain**: a tap trained on a random 20-pair HH subset reads alignment at 0.6038 against the full-HH tap's 0.6902, a gap attributable to training-set size rather than domain mismatch.

The pattern is therefore not "specialists always win." It is that specialization pays where the within-domain distinctions are hard (code, and to a lesser extent alignment) and is unnecessary where a general quality axis suffices (reasoning). The readouts are related but distinct, and no single universal quality head dominates every domain.

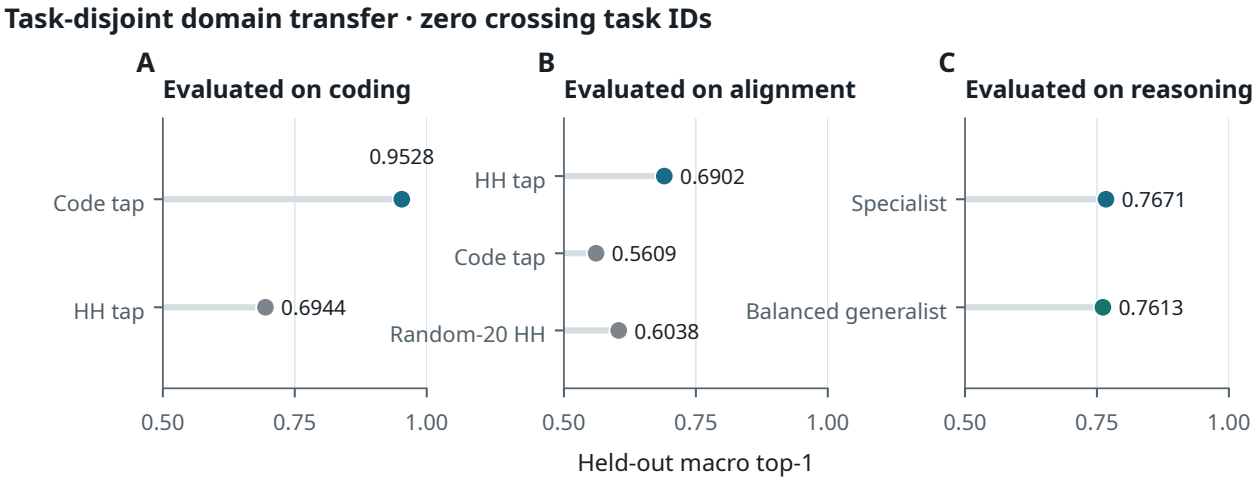


Figure 1. Clean task-disjoint domain transfer (zero task IDs crossing). Specialization pays where distinctions are hard (coding: 0.9528 vs 0.6944), is real but smaller for alignment, and is unnecessary for reasoning, where a balanced generalist matches the specialist.

These numbers replace an earlier, contaminated domain-transfer study. The prior figures — including a reported 0.986 pairwise on reasoning — came from evaluations with tens of tournaments and from splits that predate the task-disjoint discipline of §3.7; at least one of those datasets had task IDs appearing on both sides of the split. We withdrew them and re-ran the study cleanly rather than report them with a caveat. The qualitative direction survived; the magnitudes did not, and the clean study is both more conservative and more informative than the one it replaces.

A domain that resisted, and what it revealed. Not every domain yielded. A repair pass on science did not bring its tap to parity, and the failure decomposed informatively. Source-specific repair partially cleared it: MMLU **anatomy** reached partial readiness (held-out positive-oracle 0.333, parse 1.0), while **chemistry, physics, and SciQ stayed excluded**, with parse rates collapsing to 0.0. The problem was not "science" as a domain but specific sources — and anatomy's gain remains fragile (3 held-out tasks).

The diagnostic that explains it is worth reporting on its own. A **convergence-hair** probe — originally built as a branch-*merging* mechanism, demoted to diagnostic when it could not clear its safety bar (§7.5) — tracks whether a task's branches are spreading apart or collapsing toward a common continuation. On chemistry and anatomy it fires: the branches converge, and they converge to a *no-good* branch (`CHEM_ANATOMY_NO_GOOD_CONFIRMED`). The failure is therefore not that the tap cannot read quality on these sources — it is that the model does not *generate* a correct branch for the tap to find. **No selector can pick an oracle that is not in the pool.** This is a distinct failure mode from the selection wall of §6 (where the correct branch is present and cannot be committed to), it locates a limit in branch *generation* rather than branch *evaluation*, and it is one more reason the forward-looking work in §12 is training-time rather than a cleverer frozen reader.

4.5 DualAnchor and CoreContent: two constructions, two stages

Two of the role-specialized readouts, DualAnchor and CoreContent, deserve fuller treatment, because they were built by opposite methods for different points in the branch pipeline, and the contrast is itself informative. They are not two attempts at the same tap; they occupy different pipeline stages, and each was constructed in the way its stage demanded.

DualAnchor — built by transplant, for branch survival. DualAnchor is the family that operates *during* the looped branch/prune search: at each loop/layer stage it scores the current candidates, prunes the weak ones, and passes survivors forward, with the goal of *retaining* branches that still contain a correct continuation. It was not trained from scratch. It reuses the two strongest existing content/action directions in the model — the `MIX_CODE_REASONING` and `MIX_OBJECTIVE_ALL` readouts — and grafts branch-validity signal onto them by weight-space transplant, so that two taps carry content *and* branch-viability information at once. This was a deliberate architectural choice: rather than maintaining separate content taps, separate branch taps, and separate bridge taps indefinitely, the program folded content and branch-validity into a

single **dual-anchored** pair. The design reached its current form through a long incremental line (old-anchored transplant → two-tap selector → fresh-domain and HH-RLHF comparisons → layer-native re-hosting at 24/36/47 → branch-gap repair → an architecture- looped survival test across all four loops), and its defining property is asymmetric: survival is strong (stage oracle retention 0.9697 task-disjoint, terminal oracle retained 1.0000) while forced terminal commitment is not established, which is exactly the survival-without-selection pattern developed in §6.

CoreContent — built by data, for terminal ranking. CoreContent occupies the *other* stage: it ranks candidates *within* the survivor set that DualAnchor hands off, choosing a final answer rather than managing the search. Mechanically it is the same digest-and-compare engine as the relational preference evaluator of Section 3 — a frozen forward pass, mean-pooled loop states, and an antisymmetric linear tap scoring $\text{layernorm}(\text{state}_i - \text{state}_j) \cdot w$ — generalized from HH preference pairs to five content domains (alignment, reasoning, math, coding, logic). Its construction story is the inverse of DualAnchor's. The first version's hand-crafted content taps *lost* to a broad-objective baseline (`mixedhead_MIX_HH_OBJECTIVE`); the diagnosis was not that the tap architecture was wrong but that the per-domain training data was starved (coding had 30 groups, reasoning 5). The fix was data, not design: the v2 refit expanded the starved domains by factors of 27–520×, re-extracted frozen features, and refit the same small taps — at which point a crafted content tap beat the broad-objective baseline on held-out data. Under a **strictly task-disjoint** split (the corrected protocol of §3.7; zero task IDs crossing the boundary), the selected tap reaches held-out macro top-1 **0.6310**, against the broad-objective baseline's **0.5525**. This corrects a previously reported 0.6691, which was measured on a stored split in which 195 task IDs crossed the train/held-out boundary; removing that contamination costs 3.8 points and the readout survives it. One scope note is essential and easy to blur: this 0.6310 establishes task-disjoint candidate ranking on *CoreContent's own five-domain evaluation set* — it is CoreContent's designed *role* to rank within the survivor set DualAnchor hands off, but the clean number was **not** measured on actual DualAnchor survivor pools. Terminal ranking on real survivors is a separate, underpowered question whose earlier "same-survivors" comparison was withdrawn (§6.5) and remains unquantified; we report 0.6310 as candidate ranking, not as validated survivor-set selection. CoreContent is thus the Section 3 method applied to content quality, rescued by scale rather than by a new architecture — and, unlike the linear preference probe, it survives its own leakage audit substantially intact.

Why the difference matters. The two families are best understood not as reading "different aspects of a signal" but as solving the two halves of a search: *keep the right branches alive* (DualAnchor, survival) and *pick the right one at the end* (CoreContent, terminal ranking). That both were needed — and that neither subsumes the other — is part of the paper's larger finding that a readable signal decomposes by role and by pipeline position rather than collapsing into one universal quality score. The construction contrast (transplant onto existing directions vs. data-driven refit of fresh taps) also records a practical lesson: which method works depends on the stage, and a starved data regime can masquerade as an architectural failure. Detailed lineages, expansion figures, and the honest limitations of the CoreContent result — including a constructed-negative inflation and a genuine relevance ceiling — are in Appendix E.

4.6 Is the loop necessary? A non-looped architecture control

The readouts of this section are all measured on a looped model, which leaves open a question the paper must answer before attributing anything to recurrence: **is a looped architecture necessary for quality-relevant information to be linearly readable at all?** We test this directly by repeating the CoreContent-v2 protocol on a conventional, non-looped transformer.

The control backbone is MiniCPM-2B-sft-bf16: a standard decoder transformer with **40 physically distinct blocks** (no recurrent block reuse), hidden size 2,304, explicitly SFT-trained. We extract frozen features at two physical layers (24 and 36) with mask-valid mean pooling, run the identical CoreContent probe grid, and evaluate under a deterministic **task-disjoint** split (zero task IDs crossing the boundary).

Backbone	Held-out macro top-1	Held-out macro pairwise
Ouro-RLTT (looped), corrected task-disjoint	**0.6310**	—
MiniCPM-2B-sft (non-looped), task-disjoint	**0.5680**	0.7237

Per-domain, the non-looped control reads coding at 0.912, alignment 0.693, reasoning 0.520, math 0.389, and logic 0.326 — a domain profile broadly similar in shape to Ouro's, with coding unusually easy and logic hard for both.

What this establishes: a conventional non-looped SFT transformer contains linearly readable candidate-quality information that generalizes across strictly task-disjoint data. **A looped architecture is not necessary for this class of readout to exist.** This is a genuine constraint on the paper's framing, and it is worth stating in the strongest available terms rather than hedging: the readout side of this work is not, on current evidence, a property of recurrence.

What it does not establish. The 6.3-point gap between Ouro (0.631) and the control (0.568) is *not* an architecture comparison. The two models differ in backbone family, pretraining corpus, tuning objective, width, feature dimension,

parameter budget, and tap geometry; any of these could account for the difference. We therefore do **not** attribute the gap to looping, and we do not claim the looped architecture is irrelevant either — only that its contribution is unmeasured. A causal architecture study would require matched looped and non-looped models trained on the same data, objective, initialization regime, and compute budget, which we have not run.

The consequence for the paper's argument is a narrowing, and it is worth being explicit about where the loop still does work. The *readouts* (this section, §3, §5) are not shown to require recurrence. The *branching substrate* (§7) is a different matter: the loop is what gives injected branches iterative depth to diverge across recurrent steps, which a single forward pass does not provide (§7.6). And the readout–control boundary (§8) is a claim about the frozen Ouro model specifically. What we can no longer say — and an earlier draft did say — is that looping is what makes process-quality information readable.

4.7 What this establishes: the taps work

The readouts of this section are not marginal effects. Collected in one place, on their own targets and under their own held-out protocols:

Readout	Target	Result
DualAnchor (survival)	keep oracle-containing branches alive through the loop	stage oracle retention **0.9697** (task-disjoint); terminal retention **1.0000**; causal: L47 ablation → 0.0417
CoreContent v2 (terminal ranking)	rank candidate groups across five domains; designed for terminal survivor-set ranking	macro top-1 **0.6310** task-disjoint (baseline 0.5525); coding **0.8956**
S3B2 (generated-branch correctness)	is this generated branch verifier-correct?	AUROC **0.7755**, pairwise **0.7338**
Pre-answer (§5)	will this in-progress computation succeed?	+0.066 AUROC over shortcuts, CI [+0.021, +0.112]

Three of these readouts have substantial effect sizes, and one is supported causally. DualAnchor's layer-47 channel is not merely correlated with branch survival: ablating it collapses oracle retention from 1.0000 to **0.0417** (§E.2.1). That is an intervention, not a probe. It establishes that the channel — the layer-47 locus the tap reads — is load-bearing for retention; it does not by itself prove that the tap's exact learned scalar is the causal variable, only that the locus it reads is one the branch dynamics depend on.

Audit status, stated precisely. The split-protocol failure of §3.7 was not confined to the preference probes. A systematic task-disjoint re-audit of this section's results found a fifth instance of the same error — the branch-survival evaluation had eight task IDs crossing the train/held-out boundary, and 26 of its 48 evaluation tasks were training-side. We re-ran everything that could be re-run. The current status:

- **Re-verified under a zero-crossing task-disjoint split.** CoreContent: 0.6691 → **0.6310** (−3.8 pts). Branch survival: 0.9848 → **0.9697** (−1.5 pts), terminal retention 1.0000 unchanged. Domain transfer: re-run from scratch (§4.4), with the contaminated figures withdrawn rather than caveated. In each case the effect survived decontamination with a modest loss — which is what a real signal does.
- **Established by intervention, not by any split.** Ablating the layer-47 channel collapses oracle retention from 1.0000 to **0.0417**. A leaked split cannot manufacture an ablation effect; this is the strongest single piece of evidence that the locus a tap reads is one the model's own dynamics depend on (that the layer-47 locus is load-bearing, not that the tap's exact scalar is the causal variable).
- **Withdrawn, not repaired.** The terminal-selection figures (§6.5) did not survive. The clean re-run leaves only two reward-diverse tasks — too few to establish a selection effect in either direction — and an "integrated" comparison that had been reported as pairwise accuracy on real survivor pools turned out to be macro top-1 on different candidate groups. Those numbers are retracted and no replacement is claimed.
- **Never exposed to the defect.** The S3B2 detection figures were produced under a leave-one-task-out split grouped by `task_id` with an explicit leakage check that passed (16 groups, 160 candidates; `s3b2_generated_branch_correctness_expanded_2026-06-17`), so they were never exposed to the row-level construction that caused the §3 leak. The N=8 selection slice remains underpowered (§6.4), but the detection protocol itself is source-disjoint by construction.

This matters for the argument that follows. The control results of Part IV are interesting precisely because the readouts are strong: a scaffold that could not tell good branches from bad would fail to steer for boring reasons. The taps can identify which branches contain a correct answer (0.9697 retention, task-disjoint), rank candidates by quality (0.6310), detect generated-branch correctness (AUROC 0.7755), and predict the model's own success before it answers (§5). No frozen intervention we tested produced a validated capability gain from those signals.

5. Strict Pre-Answer Success Prediction

The readouts of Sections 3 and 4 read the model's hidden trajectory as it computes a candidate — which continuation is better, how good its content is, whether a branch is worth keeping. This section makes the sharpest available move: it asks whether the hidden state predicts the success of a computation that has **not yet produced an answer at all**. The pre-answer timing is what forecloses the obvious deflation — the probe cannot be reading a finished artifact, because no artifact exists yet — and that makes this the cleanest instance of the property the paper names, and the anchor of the proto-introspection framing.

5.1 Setup and the strict pre-answer cut

We evaluate on GSM8K grade-school math problems (Cobbe et al., 2021): **170 tasks**, **680 examples** total. For each, the model generates a solution trajectory, and a checker labels the final answer verifier-correct or not. The prediction task is: from hidden states extracted **before the answer is produced**, predict whether the eventual answer will be correct.

Everything turns on the **strict pre-answer cut**, so we state precisely what it excludes. A probe with access to the answer token or the numeric gold value would be reading the conclusion, not the process, and the result would be worthless — the finding would reduce to "a model that has written the right answer knows it has written the right answer."

The cut is defined and enforced in code, not applied as post-hoc filtering. Features are taken from the model's loop states over the *pre-answer* span of the trajectory only. Three things are excluded by construction: (i) the **answer region** — every token from the point at which the solution begins committing its final numeric answer onward; (ii) the **gold value** — the reference answer never enters the feature extraction path in any form, so the probe cannot be matching against it; and (iii) the **correctness label**, which is produced by an external checker *after* generation and is used only as a training target for the probe, never as an input. What remains is the trajectory of a computation that has not yet resolved.

This is what makes the result interpretable as *pre-answer*, and it is the assumption we most expect to be challenged. It has been re-verified in code as part of the audit programme described in §3.7, and the raw per-example features, predictions, and labels are preserved (`within_domain_recapture.pt`) so the cut can be inspected directly rather than taken on trust. Full extraction details in Appendix I.

5.2 Hidden states add information beyond shortcuts

A probe on pre-answer hidden features reaches **AUROC 0.745** (95% CI [0.707, 0.783]) for predicting eventual correctness. That number alone proves little: two trivial shortcuts carry some of the same information — solution **length** alone reaches AUROC 0.687, and token **log-probability** (confidence) alone reaches 0.569. The question that matters is whether the hidden state adds anything beyond "longer and more confident solutions succeed more often." It does.

Combining the two shortcuts gives a length+logprob composite at **AUROC 0.731**. The claim rests on what hidden features add *to that composite*: AUROC rises to **0.797**, an increment of **+0.066**. The hidden state contributes information the shortcuts do not contain.

Because the 680 examples are nested within 170 tasks and are therefore *not* independent, the interval must be estimated by resampling **tasks**, not examples; an i.i.d. bootstrap over examples would be anti-conservatively narrow. Under a paired **task-clustered** bootstrap (10,000 draws; each draw resamples 170 task IDs with replacement and retains all four candidates per sampled task), the 95% percentile interval on the increment is **[+0.021, +0.112]** (bootstrap mean +0.0658, SD 0.0235; zero one-class draws), which **excludes zero**. The result is not driven by any single task: leave-one-task-out re-estimation moves the increment only within [+0.056, +0.071], a maximum absolute change of 0.0098 from the full-data value. (A candidate-level bootstrap gives the narrower [+0.032, +0.100], as expected; we report it only as a diagnostic and do not use it for the claim.) Hidden states carry pre-answer information about eventual success that is **not** reducible to length or confidence.

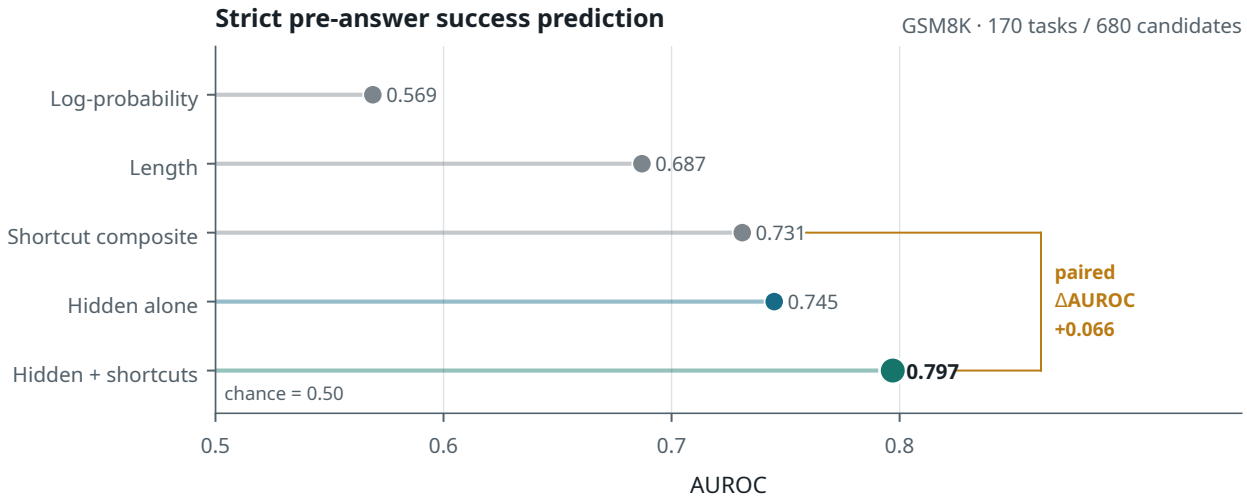


Figure 2. The paper's primary result. Length and log-probability are genuine predictors of eventual success (gray); hidden features add information beyond their composite (+0.066 AUROC, task-clustered 95% CI [+0.021, +0.112]), read from a computation whose answer does not yet exist.

> **Provenance.** The raw per-example predictions, features, and labels are preserved (`artifacts/reports/proto_introspection/within_domain_recapture.pt`: 170 tasks, 4 samples each, 680 examples, 407 positive / 273 negative), and the task-clustered interval above was recomputed directly from them (seed 20260710) rather than inherited. The *original* June interval ([+0.017, +0.114]) was described in its report as a task/group bootstrap, but the preserved analysis code does not contain the paired clustered-delta routine and the original draws were not saved, so its exact execution path is not code-auditable; the interval reported here supersedes it and independently verifies the significance claim.

5.3 Scope

The effect is incremental and it is measured on one domain. Both facts bound the claim, and neither undermines it.

Incremental is the right frame, not a weakness: length and log-probability are genuine predictors of success, they are included as explicit controls, and the hidden state's contribution is what it adds beyond them. A probe that merely re-read confidence would show no increment. This one does, with an interval excluding zero under the correct clustered test.

What that increment consists of is open. Candidate process-level correlates include the consistency of the intermediate calculation across loop iterations, the sharpness or stability of the evolving representation, and the convergence behavior of the loop trajectory. Separating these — and separating them from residual decoding correlates such as token-position variance not fully absorbed by the length control — is future work. This paper establishes that a shortcut-independent pre-answer signal exists, not which feature of the computation carries it.

The single-domain limitation is the paper's most significant, and it is not for want of trying: two candidate second domains were tested and rejected at preflight for reasons about the datasets rather than the effect (SVAMP front-loads its answers; Hendrycks MATH degenerates into a length predictor once truncation is handled — §11). Until a second powered domain exists, the proto-introspection framing rests on this result.

Finally, this result says nothing about whether the model *uses* the signal internally — a question we do not test. Part IV shows only that the frozen interventions *we* built on it produced no validated capability gain. The claim here is narrower and cleaner: the information is present, externally readable, and available before the answer exists.

6. Generated-Branch Correctness and the Commitment Gap

Section 5 established that hidden states read the quality of the model's own computation. This section is the hinge on which the paper turns from readout to control. It shows that even when a quality signal is clearly *readable*, converting it into a **commitment** is a separate and much harder problem: generated-branch correctness is decodable from hidden features, and branch *survival* — keeping the correct branch alive in the pool — works very well, yet no mechanism in this

project has yet been shown to convert either into reliable forced *commitment*. Readable, and even retainable, is not yet reliably selectable. The rest of the paper is about why that gap exists and how deep it goes.

6.1 The task

We generate pools of candidate branches for reasoning tasks, label each branch verifier-correct or not, and ask two questions of a hidden-feature reader (the S3B / S3B2 setting; Appendix E). First, **detection**: can the reader decode whether a given generated branch is correct? Second, **selection**: forced to commit to a single branch from a pool, can the reader reliably pick a correct one, against a properly matched random baseline?

6.2 Survival works: the branch-retention scaffold

Branch *survival* — keeping a correct continuation alive in the pool while pruning weak branches — is the one part of this pipeline that works well, and it is verified under a zero-crossing task-disjoint split.

Metric	Clean (task-disjoint) result
Stage oracle retention (DualAnchor)	**0.9697**
Terminal oracle retention	**1.0000**
Scaffold top-4 retention	**1.0000** (52 groups / 26 tasks)

These figures correct an earlier contaminated evaluation. The original 0.9848 stage-retention number was measured on a split with eight task IDs crossing the train/held-out boundary, and 26 of its 48 evaluation tasks were training-side; a clean re-run with zero crossings gives 0.9697 — a 1.5-point drop, the signature of a real effect surviving decontamination rather than an artifact dissolving (§3.7 documents the class of error, and §4.7 the audit tiers).

The survival claim does not rest on held-out accuracy alone. It is established causally: **ablating the layer-47 channel the survival tap reads collapses oracle retention from 1.0000 to 0.0417**. A leaked split cannot manufacture an ablation. The scaffold is reading something the branch dynamics genuinely depend on.

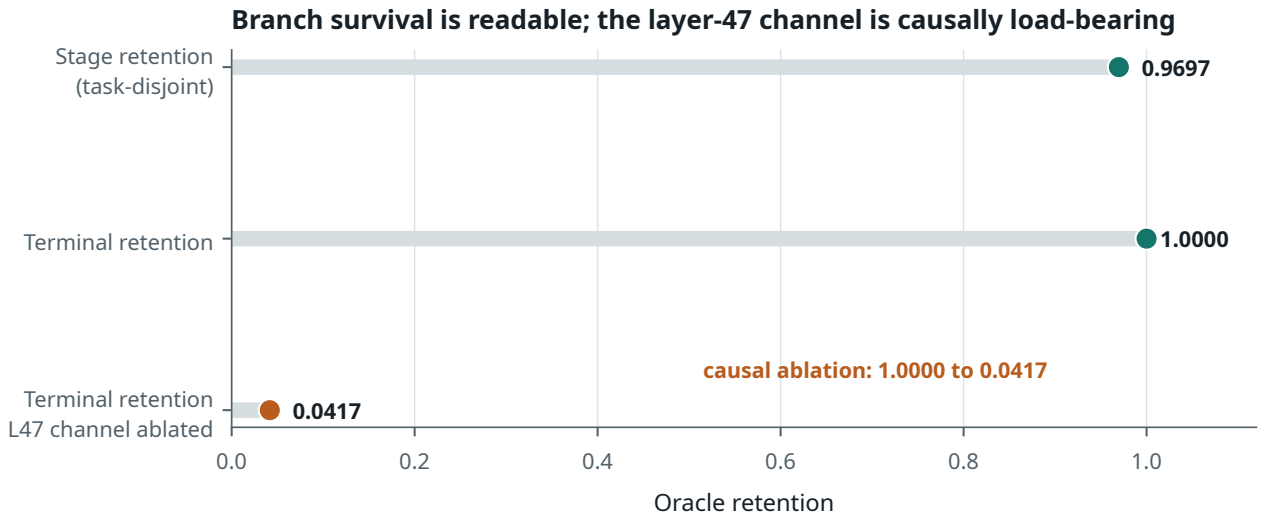


Figure 3. Survival works under a zero-crossing task-disjoint split, and the layer-47 locus is causally load-bearing for retention: ablating the channel the tap reads collapses retention to 0.0417. This does not identify the tap's exact scalar as the causal variable.

What survival does *not* deliver is commitment, and the rest of this section is about that gap.

6.3 Correctness is readable

Detection succeeds. An L2-regularized logistic reader over hidden features reaches **AUROC 0.7515** with pairwise accuracy **0.6835**; an expanded hidden-ridge reader reaches **AUROC 0.7755** with pairwise accuracy **0.7338**.[1] Metadata-only

[1]The S3B2 detection figures (AUROC/pairwise) are pinned by the pending-items live-repo pass (`artifacts/reports/paper_verification/pending_items_resolution_20260703_214531.*`). The **0.5833** figure sometimes reported alongside them is **not** a matched S3B2 control, contrary to an earlier draft note: it is the macro-average over three oracle-present domains from the S3B1 corrected transfer, $(0.5_{\text{math}} + 0.75_{\text{reasoning}} + 0.5_{\text{logic}})/3 = 0.5833$ (`utilities/tests/manual/mpn_s3b1_loop_pool_transfer.py`), aggregated differently from the pool-weighted selection fraction in §6.4. It is retained only for continuity with prior tables and is explicitly not used as the matched baseline for the selection claim.

controls (features derived from surface properties rather than hidden states) remain weak, so the signal is in the hidden geometry, not in incidental artifacts. Generated-branch correctness is genuinely, non-trivially decodable — a fourth role-specialized readout to add to those of Section 4.

6.4 Selection is not established

Selection does not follow. The oracle-present selection task comprises **N = 8 task groups** (drawn from a full pool of 160 candidates across 16 groups; groups with no correct branch cannot contribute an oracle-conditioned selection score). Under forced top-1 choice, the expanded reader selects a correct branch in **5 of 8 groups (0.625)**, and the weaker reader matches it — the substantially higher detection AUROC (0.7755) buys no improvement in forced-choice selection.

The baseline requires care, and two earlier figures were wrong. The eight groups contain ten candidates each, with correct-branch counts $(7, 1, 3, 2, 4, 2, 8, 2)$. Uniform random choice within each group therefore succeeds with matched expectation **0.3625** (2.9 of 8 groups) — *not* the 0.5833 that earlier tables imported from the aggregation-mismatched S3B1 macro, and not the 0.625 a subsequent draft mistakenly asserted. The selector's 5/8 does **exceed** the matched-random point expectation. But the exact Poisson-binomial probability of at least five successes under random choice is $p = 0.087$, which does not clear significance at the 0.05 level: with eight groups, the test is underpowered.

The conclusion is therefore neither "selection works" nor "selection is at chance," but that **reliable forced selection is not established on this slice**: correctness is clearly decodable (pairwise ≈ 0.73, AUROC ≈ 0.78), the selector points in the right direction, and eight groups are too few to conclude anything firmer. High-margin abstention does not rescue forced top-1 either.

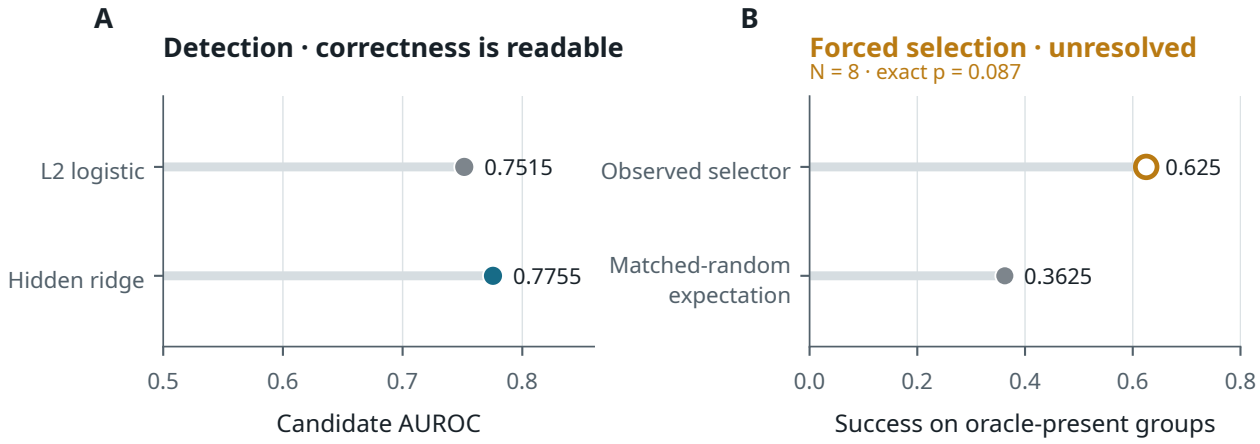


Figure 4. The commitment gap in its sharpest local form: generated-branch correctness is decodable (AUROC 0.7755, left), yet forced top-1 selection on the N=8 oracle-present groups exceeds the matched-random expectation without reaching significance (exact p=0.087, right).

We report this slice because it is the setting in which correctness is most directly decodable, and because its detection/selection gap motivated the larger survivor-set experiments. Those experiments (§6.5) were themselves audited and their quantitative claims withdrawn, so the honest position across both is the same: **no adequately-powered clean evaluation in this project establishes either a working terminal selector or a quantified selection deficit.** What survives is a qualitative bottleneck supported by convergent evidence (§6.5), not a headline number.[2]

6.5 The terminal-selection bottleneck: a finding without adequate quantitative support

Across every selection mechanism in this project the same pattern recurs: **the survivor set often contains a correct branch, but reliable forced commitment has not been established.** Survival is solved (§6.2: 0.9697 stage retention, 1.0000 terminal retention, verified clean). Detection is solved (§6.3: AUROC 0.7755). Forced terminal commitment remains unresolved.

We must be precise about the evidential status of that claim, because a task-disjoint audit removed most of its quantitative support and we report the audit rather than the numbers it invalidated.

[2]The matched-random baseline is analytic, not an artifact lookup: with per-group correct counts $(7, 1, 3, 2, 4, 2, 8, 2)$ out of ten candidates each, the expected random hit rate is $\frac{1}{8}\sum_i c_i/10 = 0.3625$, and the exact Poisson-binomial $P(\geq 5 \text{ hits}) = 0.0869$ (the probability of exactly five is 0.0717). Two prior figures are hereby corrected: the 0.5833 (an S3B1 three-domain macro, aggregated differently) and a later erroneous claim that matched random equalled the selector's 0.625.

What was withdrawn. An earlier version of this section rested on three figures, all now retracted. The best-survivor-versus-final-reward gap (0.9453 vs 0.6672) came from the contaminated split described in §6.2. The integrated comparison — "CoreContent pairwise 0.658 versus DualAnchor forced top-1 0.379 on the same survivors" — fails on three counts: the 0.658 was CoreContent *macro top-1*, not pairwise accuracy; it was measured on CoreContent candidate groups rather than on real branch-survivor pools, contrary to how it was described; and it predates the corrected CoreContent split. On the clean actual-survivor subset the comparison does not merely shrink, it *reverses* descriptively (CoreContent top-1 0.7778 against DualAnchor forced top-1 0.8889) — and that reversal is itself underpowered, so it establishes nothing in either direction. We do not report any of these numbers as evidence.

What the clean data shows. The zero-crossing re-run leaves nine clean DualAnchor tasks, of which only two have reward-diverse candidates — too few to establish a terminal-selection effect in either direction. On that remainder, forced top-1 oracle retention is 0.8889 and forced top-1 reward is ≈0.0000 against a best-available terminal reward of 0.0222: a gap of +0.0222 on two informative tasks, which is not evidence of anything. The honest statement is that **no adequately-powered clean evaluation in this project demonstrates a working terminal selector, and none demonstrates a quantified selection deficit either.**

Why we still report the bottleneck as a finding. Three independent lines converge on it, none of which depends on the withdrawn numbers. First, the S3B2 slice (§6.4): correctness is decodable at AUROC 0.78 and forced selection is not established at N=8. Second, the arbiter lineage (Appendix E.3): every terminal arbiter built in this project — listwise softmax, tie-aware listwise rank, merged weight taps, domain-gated fallback — closed with a weak or `NO_IMPROVEMENT` verdict, and the program's locked policy is *confidence-gated top-1 with defer*, i.e. an explicit decision not to force commitment. Third, and most directly, the control results of Part IV: no tested frozen intervention has produced a validated capability gain. The project's adopted defer policy is consistent with that unresolved status.

A measurement trap worth stating, since it recurs. Aggregate forced top-1 scores flatter themselves on tie-heavy tasks, where many branches are equally good and any choice scores well. The clean re-run makes this concrete in the sharpest possible way: of nine clean tasks, only two are reward-diverse. Any terminal-selection metric computed over the full set is dominated by tasks where selection cannot be wrong. This is the same class of trap as the canonical-ordering artifact of §3.3 — an aggregate that looks strong because the hard cases are diluted — and it is why we decline to report a headline selection number rather than reporting an inflated one.

The connection to Section 3, stated as hypothesis. Forced top-1 is an *absolute, listwise* commitment, and §3.2 finds that preference is decoded more accurately relationally (0.5653) than pointwise (0.5418). That a model ranks pairs better than it scores singletons is a plausible mechanism for why pairwise comparison works while forced absolute commitment does not — and it is consistent with every listwise arbiter in the lineage failing to improve. We flag it as a hypothesis the current data motivates but does not test.

The scoped conclusion: **survival is solved, detection is solved, and commitment is unsolved and currently unquantified.** That is a weaker claim than the one this section previously made, and it is the one the clean data supports.

6.6 Why this is the hinge

This section is where the paper turns, and the turn survives the audit even though several of its numbers did not.

The chain is: the model's hidden states let an external reader **keep the correct branch alive** (0.9697 retention, verified clean, and causally established by ablation) and **recognize a correct branch when they see one** (AUROC 0.7755). No mechanism in this project has yet been shown — not a listwise arbiter, not a tie-aware ranker, not a merged tap, not the S3B2 selector — to convert either capability into reliable **commitment**. Readable, and even retainable, is not yet reliably selectable.

That is the readout–control boundary in its first appearance, and it appears in the most favorable setting available: an external reader, handed correctness labels to train on, asked only to choose among candidates the model already produced, with no demand on the frozen model whatsoever. If conversion has not been shown to work even when the reader is given everything, it is unlikely to be easier when the demand is harder — when the frozen model must itself be steered, or branched, to produce gains. Sections 7 and 8 test exactly that: first by building machinery through which those signals *could* be acted on, validated by bit-exact identity, and then by showing that no frozen intervention through that machinery produced a validated capability gain under the conditions tested.

---

7. Internal Branch/Carry/Prune Machinery

Before asking whether the readable signals can be *acted on*, we build the machinery that would make acting possible — and that machinery turns out to be the part of this project no audit touched.

This section describes an executable internal branching substrate for Ouro-RLTT: autoregressive, branch-specific key/value-cache carry across a 192-slot recurrent cache, validated through a six-level correctness ladder; a bit-exact suffix-recompute splice that cuts up to 88% of per-branch compute; and a live fork/carry/prune/reorder scaffold. Two things make it more than apparatus. First, it is validated by **exact identity and negative controls**, not by output plausibility — which is what makes the negative results of Part IV credible rather than attributable to a broken scaffold. Second, exact-equivalent branch-specific cache manipulation inside a *looped* transformer is not a port of standard incremental decoding, and we present it as a systems contribution usable independently of the introspection question this paper studies. We claim no capability gain here: the substrate is validated for *correctness and executability*, and Part IV is where we show that correctness is not enough.

7.1 What makes branch-carry hard in a looped model

The problem is not obviously hard until one tries it, so it is worth stating what the substrate must do that a standard incremental decoder does not.

Ouro's cache (`UniversalTransformerCache`) is indexed by **192 distinct slots per position** — slot $= u \cdot 48 + \ell$ for loop $u \in \{0..3\}$ and layer $\ell \in \{0..47\}$. Prefill populates all 192; each decode step appends one token to *every* slot; a batch reorder must permute the batch dimension of every populated slot. A **branch** is a distinct trajectory through this structure, and carrying one autoregressively means keeping its own `past_key_values`, `cache_position`, `attention_mask`, `position_ids`, `generated_ids`, and lineage aligned across every decode step, for every one of the 192 slots, without leaking into any sibling branch.

This is a different and strictly harder problem than the prompt-only layer carry used by our offline probes (which run with `use_cache=False` and can afford to recompute). Generation-time, branch-specific KV carry is what a *live* scaffold requires, and it is where the correctness burden actually lives: a single misaligned `position_ids` row or an incorrectly reordered slot produces plausible-looking text and silently invalid branches. We therefore validated it as a ladder of increasingly demanding correctness properties rather than by inspecting outputs.

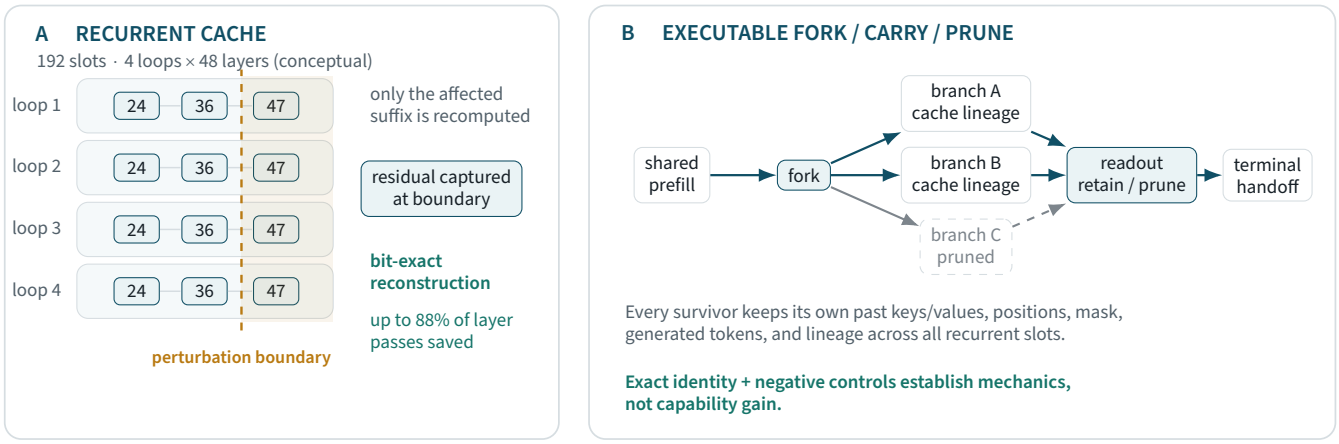


Figure 5. Left: the 192-slot recurrent cache (4 loops × 48 layers per position), with the tapped readout layers marked. Right: the live scaffold — one shared prefill, branch-specific cache lineages forked at a loop/layer boundary, carried, pruned, and handed off to terminal ranking.

7.2 A validation ladder, not a smoke test

Level	Property established
L0 cached decode	matches full recompute (prefill bit-exact; decode within bf16 drift)
L1 token-boundary fork	$K = 2/4/8$ independent branch caches, **no cross-branch contamination**
L2 batched branches	batched ≡ independent ≡ full recompute
L3 prune / reorder	survivor subsets and order changes ($8 \to 4 \to 2$, $8 \to 3$, $4 \to 1$) keep lineage aligned
L4 current-token perturb	injection at layers 24/36/47, loop-targeted, carries correctly via the branch cache
L5 prompt-internal perturb	branch-specific cache required — **negative control**: withholding it diverges to RMS ≈ 3.0

All six pass. (A seventh level, L6, tested a first-attempt partial splice: its slot-boundary logic was valid but it delivered no compute saving, and it was superseded by the v2 splice of §7.3.) Two properties are worth drawing out. **Correctness is established by exact identity, not similarity**: a zero-perturbation fork reproduces the reference **bit-exactly at prefill (RMS 0)**, with only small bf16 drift during cached decode (RMS ≈ 0.05–0.2, max-abs < 1.0) — the expected numerical signature of cached-versus-recomputed key/value paths, not an error in the branch logic. And **the negative control matters as much as the positive**: forcing a branch to proceed *without* its proper cache lineage diverges to RMS ≈ 3.0, which is what tells us the carried cache is load-bearing rather than incidental. Correct carry gives exact reproduction; absent carry gives large divergence. A mechanism that only ever produced plausible text would have passed neither test.

Batched decode required one non-obvious fix: left-padding a batch of branches breaks unless each row is given explicit `position_ids` (RoPE's relativity makes a single-prompt left-pad shift harmless, but a *batched* one is not). We record it because it is exactly the class of bug that produces silently wrong branches rather than crashes.

7.3 The compute-saving splice, and the obstacle it had to clear

Naively, evaluating a perturbed branch means re-prefilling the whole prompt for that branch: K branches, K full prefills. The obstacle to doing better is specific and easy to miss. **The KV cache stores keys and values, but not the inter-layer residual stream.** A perturbed branch therefore cannot simply resume from the shared cache — the residual it would need to continue from was never stored, and reconstructing it appears to require the forward pass one is trying to avoid.

A first attempt (v1) validated the slot-boundary logic — the copy-affected cache reproduced the full cache bit-exactly — but delivered **no compute saving at all** (`PARTIAL_SPLICE_DIAGNOSTIC_ONLY`): knowing *which* slots change does not help if you must still re-prefill to obtain the residual they depend on. The saving required a second idea.

The solution is to capture, during a single shared-prefix prefill, the residual hidden state *at the perturbation boundary* (the output of loop u, layer ℓ). For an additive boundary perturbation the perturbed residual is then $H_{\text{boundary}} + \delta$ — reconstructible **with no forward pass at all** — and only the *suffix* need be recomputed: the remaining layers of loop u, then loops $u + 1$ onward. The implementation is test-only orchestration over the model's layers, rotary embeddings, norm, and head: no weight edits, no permanent model surgery.

Establishing which cache slots this actually touches required an empirical result we did not anticipate: perturbing a layer's *output* leaves that layer's own boundary slot unaffected, so the first affected slot is $(u,\ \ell + 1)$, and the changed set is exactly the `downstream_only` prediction. Over-sharing an affected slot diverges — the negative control that pins the boundary.

The result is **bit-exact and cheap**. The spliced branch cache matches a full perturbed-prompt reference across all 192 slots, with prefill logits at RMS 0 and the continuation identical token-for-token — validated single-branch, multi-branch ($K = 2/4$, independent storage, no contamination), batched, and under prune/reorder. The savings grow with fork depth: recomputing only the affected suffix saves **13% / 38% / 63% / 88%** of per-branch layer passes for fork boundaries at loops 0–3, and **32% / 47% / 55%** at $K = 2/4/8$ branches (fork at loop 2, layer 24). Savings amortize over $K \geq 2$; at $K = 1$ there is nothing to share.

We position this carefully against a well-developed systems literature. Branch-forking and prefix-sharing over KV caches are standard: PagedAttention (Kwon et al., 2023) forks with copy-on-write pages, SGLang's RadixAttention (Zheng et al., 2024) shares prefixes across a radix tree of branches, and SpecInfer (Miao et al., 2024) verifies a token tree over a shared cache with tree attention. Reuse *across recurrent steps* is also known: depth-recurrent models attend to KV entries generated by the same projections at every iteration (Geiping et al., 2025; Zhu et al., 2025). What we have not found in that literature is the combination specific to this substrate — branch-specific carry plus a **residual-capture suffix splice** that reconstructs a branch perturbed *mid-computation* (at an interior loop/layer boundary) bit-exactly over the loop×layer recurrent cache, without the re-prefill that a stored-K/V-only cache would otherwise force. It is not a port of standard incremental decoding, and it is usable independently of the readout–control question this paper studies — which is why

we present it as a contribution in its own right rather than as apparatus. We do not claim a performance advantage over these systems; the comparison is one of mechanism, and the substrate's role here is as a validated instrument, not a faster search.

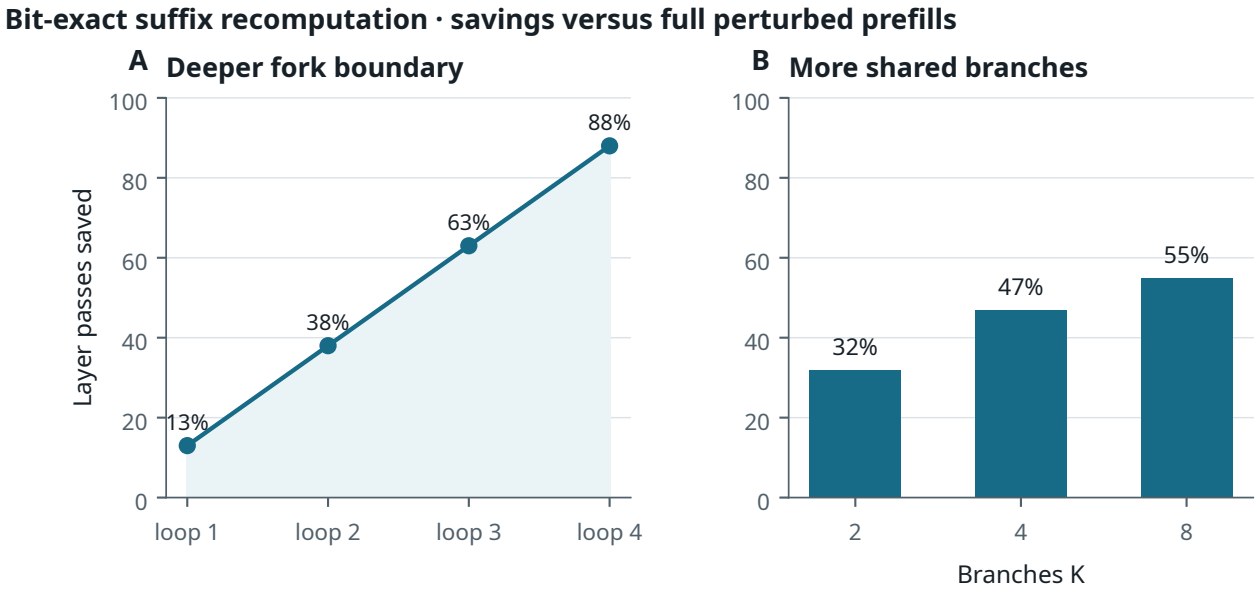


Figure 6. Compute saved by the bit-exact suffix-recompute splice, relative to K full perturbed prefills: by fork boundary (left — deeper forks recompute less) and by branch count at a fixed boundary (right). Savings amortize over $K \geq 2$.

7.4 The live scaffold

These pieces compose into a live **branch/carry/prune/loop-back** scaffold: the system forks candidate branches at chosen boundaries, carries their caches forward, scores and prunes survivors, reorders the live set, and can loop back to earlier boundaries to re-explore. This is the apparatus required to *act* on process-quality readouts at run time — to branch where a readout is uncertain, prune branches a readout deems weak, and commit where a readout is confident. Having built it, we can ask the paper's central question sharply: given that the readouts exist (Part II) and the machinery to act on them exists (this section), does frozen intervention through that machinery actually help? Part IV answers no.

7.5 A mechanism we built and did not keep: convergence hairs

Not every component of the scaffold survived. One is worth reporting because the reason it failed is informative, and because it is the kind of thing usually deleted from a paper rather than described.

The idea. Branches that fork from a shared prefix often reconverge — they drift apart, then settle back onto substantially the same continuation. If that can be *detected* mid-generation, the redundant branches can be **merged**, freeing search budget for genuinely distinct alternatives. We built this: a **convergence-hair** probe reading hidden states at layers 30 and 42, with a policy layer that would hard-merge branches judged to have converged.

The bar, set in advance. A merge policy is only safe if it does not silently discard the branch that would have been correct. We therefore required a policy to clear three conditions simultaneously before it could be used as a hard merge: terminal oracle retained ≥ 0.98; false-merge rate ≤ 0.05; and survivor reduction ≥ 0.10 — the last because a merge that saves nothing is not worth its risk.

No policy cleared it. The best-performing merge retained the terminal oracle on **0.9583** of tasks (below the 0.98 floor) while reducing survivors by only **0.0247** (against the 0.10 requirement). It was simultaneously too *aggressive* — losing oracle branches — and too *timid* — barely shrinking the pool. Conservative variants were safer and merged even less, which made them pointless. The verdict was to keep the probe as a **soft diagnostic** and never merge on it: `DUALANCHOR_CONVERGENCE_HAIRS_RS_STATUS = SCIENCE_BRANCH_GENERATION_WEAK`, hard merge not cleared.

Why this is not a footnote. First, it is a concrete instance of the paper's central asymmetry appearing *inside the substrate itself*: convergence was **readable** — the hairs found real diagnostic structure — and it was not **actionable**, because acting on the reading cost more oracle than it saved compute. Readable is not usable, one level down from where Part IV finds it.

Second, the discarded mechanism turned out to be a good instrument. In its demoted role the hair probe diagnosed the science-domain failure of §4.4: on chemistry and anatomy the branches converge, and they converge *to a no-good branch* — which is why no selector could rescue those tasks, and why the limit there is branch **generation** rather than branch evaluation. The component we could not use for control became the one that told us where the pipeline was actually broken. We report it because "we built X, X did not clear a pre-registered bar, and X was useful for something else" is a more honest and more useful thing to write down than a scaffold description with the failures pruned out.

7.6 Why a looped backbone: branches need iterative depth to diverge

Neither the branch machinery nor the readouts are, in principle, unique to looped transformers. A standard (non-looped) transformer could also fork at a token boundary and carry branch-specific key-value state, and process-quality signals could in principle be read from a standard model's activations. What the looped architecture supplies is not the *ability* to branch but the *time for branches to diverge.* Because the same weights are applied across four loop iterations, a perturbation injected at a loop/layer boundary propagates and differentiates over the subsequent iterations: a branch co-develops across the loop trajectory, and can be re-perturbed and pruned across iterations rather than being fixed at the moment of the fork. The four iterations give branches a shared-weight refinement budget — depth *in time* — within which to genuinely separate.

A standard transformer diverges too — a branch there develops through its remaining physical layers, across subsequent autoregressive token steps, and through ordinary cached decoding. What it lacks is the *specific* budget the loop supplies: repeated application of the *same* block stack at the *same* token position, i.e. recurrent-depth stages at which a branch can be re-perturbed and pruned at corresponding points of a shared-weight refinement. A non-looped model's within-position computation is traversed once; the loop adds passes over the same position that a branch can co-develop across. This is the sense in which our substrate is neither best-of-N nor Coconut: best-of-N draws independent one-shot samples with no shared evolving state, and Coconut superposes candidate steps within a single continuous thread; our branches are distinct trajectories that diverge along the loop dimension, which only a looped (or otherwise depth-recurrent) backbone provides; the representational advantages of such depth-recurrence are analyzed by Saunshi et al. (2025).

The scope of this argument is narrow. It is a claim about the *mechanism's* suitability, not a capability claim: the divergence itself is mechanically real (zero-perturbation forks stay identical at prefill while perturbed and no-carry branches diverge measurably; §7.2), but Part IV shows that in the *frozen* model this genuine divergence does not translate into reachability gains over matched sampling. The loop gives branches additional recurrent-depth opportunities to diverge; it does not, absent training, make that divergence useful for control. That gap is exactly the readout–control boundary, and the iterative depth argument is why we locate the fix in training-time integration (Section 12) rather than in a different frozen search procedure.

PART IV

Control

8. Frozen Branching, Steering, and the Readout–Control Boundary

Parts II and III put two things on the table: readable process-quality signals, and an executable substrate through which those signals could in principle be acted upon. This section establishes the paper's load-bearing control result: the tested frozen interventions do not produce a validated capability gain from those signals, with the evidence levels separated below.

Whose failure this is. One clarification governs everything below, because the alternative phrasing is tempting and wrong. We do not show that "the model cannot use its own signal." The model is not consulting anything: it does not know the taps exist, and it plays no part in reading or acting on them. In every experiment here, *we* are the agent — an external probe reads the hidden states, and an external policy (a steering hook, a fork, a selector) attempts to act on what was read. The claim is therefore precise and narrower than the anthropomorphic version: **no frozen intervention we constructed produced a validated capability gain from the readable signal.** Whether the model itself makes internal use of process-quality information is a question this paper does not test and cannot answer.

One established negative and two conversion attempts that do not establish a gain under the tested conditions characterize the boundary:

Level	Intervention	Finding
Directional (§8.1)	write a readable direction into the residual stream	*established negative*: across seven methods the direction that reads success is not the direction that causes it (the *geometry*)
Branch-level (§8.2)	fork, carry, and generate a perturbed branch	*bounded screen* (four tasks): injected branches are real and divergent, but do not outperform K-matched ordinary sampling — the positive interpretation is removed, though the screen is too small to estimate a general deficit
Selective (§6, §8.3)	given the correct branch is in the pool, commit to one	*unresolved*: reliable conversion from detection to commitment has not been demonstrated, and the clean evaluations are underpowered

These are not three views of one finding; they concern different mechanisms, which is part of why the boundary is robust. We then test the simplest geometric explanation that would unify them and find that the audit does not support it (§8.4). We call the resulting separation the **readout–control boundary**, and show it is *empirical*: not the consequence of a clean linear-algebraic obstruction, but a robust pattern — an established directional negative plus two conversion routes that yield no gain under the conditions tested — that survives the obvious attempts to explain it away.

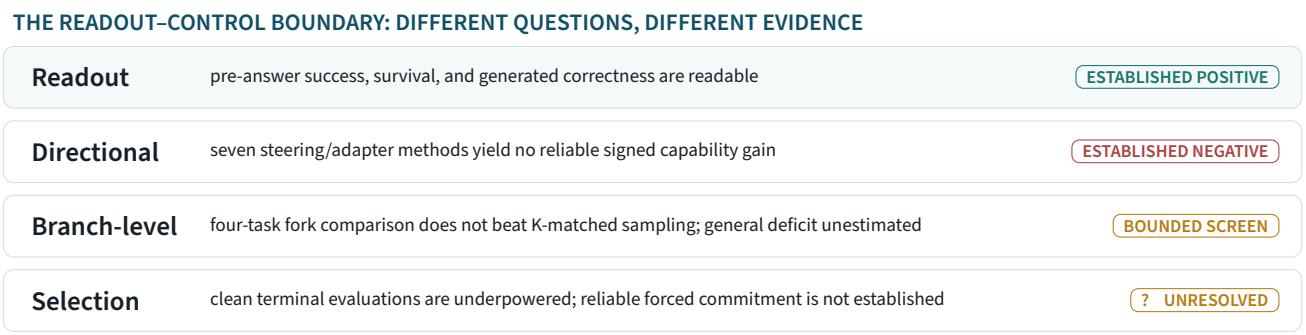


Figure 7. Evidence status of the readout–control boundary. Process-quality readouts are established positive; directional steering is an established negative; the branch-level comparison is a bounded four-task screen; and terminal selection remains unresolved under the clean, underpowered evaluations. The bounded LoRA run and two-null audit are diagnostic rather than capability results.

8.1 Directional control fails: the write surface works, the direction does not

The most local intervention adds a readable direction back into the residual stream during generation and asks whether it steers behavior. The result is a clean dissociation, and getting the dissociation right matters more than the failure itself: **the write path is mechanically valid, and the directions are wrong.**

The write surface is validated, not assumed. Before concluding that steering fails, we established that steering is mechanically possible. Decoder-layer hooks are clean: zero-magnitude writes reproduce no-hook generation *exactly*; perturbation size scales predictably; perturbations propagate to later hidden states and to logits (surviving on the order of 32 tokens); and no CUDA/NaN/Inf instability appears within the safe envelope ($\alpha \leq 0.02$ effective RMS). Whatever fails here, it is not the plumbing — an important distinction, because "we tried steering and it didn't work" is otherwise indistinguishable from a broken hook.

Seven methods, one verdict. Each tested route produced an *unsigned* effect: outputs move, but not in the intended direction.

Method	Verdict
Raw readout direction (NoNorm)	`UNSIGNED_EFFECT`
Empirical success-mean difference	`EMPIRICAL_UNSIGNED_ONLY`
RMS-calibrated static direction	`RMS_UNSIGNED_ONLY`
Local outcome-score gradient probe	`GRADIENT_NO_BETTER_THAN_RANDOM`

Method	Verdict
Classifier-derived adapter	no reliable held-out control
Teacher-forced causal adapter	`LOCAL_LOGIT_CONTROL_ONLY` — improves logit margin under teacher forcing, `TEACHER_FORCED_ONLY` in free generation
Sequence-level (REINFORCE) adapter	`SEQUENCE_REWARD_IMPROVES` in training, `NO_ADAPTER_SPECIFIC_TRANSFER` held-out, and **`WORSE_THAN_RANDOM`** against a random-direction control

The last row deserves emphasis, because it is a stronger negative than "no gain." An adapter trained directly on sequence-level reward *did* improve that reward during training — and then, on held-out generation, performed **worse than a random direction of matched magnitude**. It did not merely fail to find a control direction; it confidently found the wrong one, and optimizing harder made it worse. The stopping verdict is `NO_FROZEN_BACKBONE_WRITE_PATH` / `FROZEN_BACKBONE_INFERENCE_STEERING_STATUS = CLOSED_UNDER_TESTED_METHODS`.

Why: three geometries that do not coincide. The direction that *reads* success, the direction along which successful and unsuccessful trajectories empirically *differ*, and the direction a learned adapter uses to exert local control are mutually near-orthogonal:

Direction pair	Cosine
Adapter control proxy vs. raw readout	−0.00055
Adapter control proxy vs. empirical success-mean difference	−0.00429
Raw readout vs. empirical success-mean difference	+0.10100

Compactly: *readout geometry ≠ empirical-success geometry ≠ local logit-control geometry.* The model's hidden states tell an external reader which trajectories are promising; the hooks can write into those states; and the direction that carries the reading is not the direction that causes the outcome. Two independently trained adapters converged on nearly the same direction (cosine +0.951) — they agree with *each other* and disagree with the signal they were built to exploit, which is what one expects if both are descending into the same non-steering local-control basin rather than discovering the outcome direction.

Two scope notes. This closure holds under the tested safe-α envelope and tested optimizers; it does not prove that *no* training method can ever steer Ouro, and Section 12 is about the training-time route it motivates. And this local linear diagnostic is *not* the global subspace-misalignment explanation tested but not supported in §8.4 — it is the narrower observation that the obvious linear write directions are not already usable control directions. Full method configurations in Appendix H.

8.2 Branch-level screen: the frozen-fork comparison under matched sampling

One level up, we fork rather than nudge: inject a perturbation at a loop/layer boundary, carry the branch forward through its own cache (Section 7), and let it generate an independent continuation. Mechanically this works — the substrate's bit-exact identity and lineage guarantees hold. In the bounded four-task comparison, injected branches did not outperform K-matched plain sampling. **Greedy** forks yield **zero** new-correct answers relative to the unforked baseline (divergence and reconvergence only). **Sampled** forks do occasionally reach new correct answers — the tempting result to report — but a **K-matched plain-sampling deconfound** dissolves it. Under matched conditions (4 tasks, 12 plain samples per task, temperature 0.7, top-p 0.95, 96-token sampling budget), plain sampling reaches an oracle of **0.750** while the sampled fork reaches **0.611** — the fork is −0.139 *below* matched sampling, not above it. The apparent gains are attributable to the sampling randomness the fork happens to introduce, not to the injection; drawing the same number of ordinary temperature samples does at least as well. A separate check confirms the branches are not cosmetic. **Hook-origin branches persist geometrically**: a perturbation injected mid-trajectory is still detectable in the hidden states at layer 47, and such branches *do* sometimes change the downstream outcome. The injection is producing genuinely distinct trajectories, not decorative noise that washes out — which is what makes the null informative rather than vacuous. What the frozen model lacks is not divergence but a way to *aim* it: the branches are real, they go somewhere different, and nothing in the frozen system can tell in advance which of them to prefer.

The scope of this claim is bounded by its size, and we state it as such. Four source tasks are a screen, not a powered null: the comparison removes the evidence for a frozen-fork *gain* (the sampled-fork advantage is explained by matched sampling, and greedy forks add no new correct solutions), but it is too small to estimate a general deficit. We record it as a bounded frozen-fork screen (Appendix G): the mechanism is valid, and in this four-task comparison injected branches do not outperform K-matched sampling.

This deconfound is also the direct answer to the objection that the branch/carry substrate is merely "best-of-N with extra steps." The K-matched comparison *is* best-of-N with oracle selection, and it is the baseline the frozen substrate is measured against — not a competitor the paper sidesteps. The honest conclusion is not that the branch machinery is useless, but that frozen inference-time branching does not yet beat the corresponding sampling baseline; whether trained branch control can is the question of Section 12.

A bounded training probe weighs against the "a little training would fix it" rejoinder, without closing it. A 300-step bf16 LoRA (30.3M parameters, 1.12% of the model, loss 0.34; deliberately not run to convergence) measurably changes behavior — branch **diversity** rises by +0.45 (2.17 → 2.62), coding **parse** improves 0.72 → 0.94, and math **oracle@K** improves 0.75 → 0.92 — yet net **reachability** is flat: macro positive-oracle@K moves 0.708 → 0.688, because logic (0.83 → 0.75) and reasoning (0.92 → 0.67) regress even as math and coding gain (math parse was already saturated at 1.0 → 1.0). Light training moves the surface statistics a readout cares about — diversity, parse quality — without moving net outcome reachability. Frozen readout does not confer control, and one shallow training pass does not either — a bounded probe against the trivial rejoinder, not a closure over light training generally.

8.3 Selection-level control is not established

The third level grants the reader the correctness signal and asks only that it *choose.* This is the S3B2 result of Section 6, which belongs equally here: generated-branch correctness is decodable (AUROC 0.7755) yet reliable forced top-1 selection is not established (5/8 on N=8 groups; matched random 0.3625, exact $P(\geq 5) = 0.087$ — the selector points the right way but the test is underpowered; §6.4), and abstention does not rescue it. The survivor-set experiments of §6.5 point the same way, though their quantitative claims were withdrawn under audit; what converges is the qualitative bottleneck, not a magnitude. Even with labels in hand and no demand on the frozen model, readability does not become reliable selection.

8.4 A simple geometric explanation, tested but not supported

The three intervention levels invite one tidy explanation. If the frozen branch mechanism can only write into a subspace U_{inj}, and the outcome direction d_{out} lies largely *outside* it, the frozen null would follow immediately — no intervention could push the computation along the direction that matters, however well that direction can be read. This would make "a signal that cannot be acted through" a literal geometric statement, and we tested it directly rather than asserting it.

Using exact-protocol regenerated S1/S3 frozen injection/carry deltas (regenerated from the protocol, not replayed from historical saved tensors — a caveat we preserve), we measured $\pi_{\text{inj}}(d_{\text{out}}) = 0.0183$. Taken alone this looks decisive: the outcome direction places 98.2% of its energy outside the writable span. It is not decisive, because a projection fraction is uninterpretable without the span's rank. The injection span has rank $k = 344$ in a $D = 24{,}576$-dimensional feature space, so a uniformly random direction already projects $k/D = 0.0140$ of its energy into the span in expectation. The observed 0.0183 is 1.31× that baseline — *above* random-subspace chance, not below it; against an empirical random-direction null (one million matched draws) the observed projection sits at the 99.99th percentile. The naive reading — outcome direction lies unusually *outside* the injection span — is therefore not supported by this null.

The appropriate null for the actual question is stricter: comparing d_{out} not to random directions but to **outcome-shaped shuffled-label** directions (directions with the same correlational structure as a real outcome axis but no true verifier information), the observed projection (0.0183) falls *below* their mean ($\approx$ 0.0227; global shuffled mean 0.0230, domain-stratified 0.0227), at the low end of the shuffled-label null. The pending-items pass pins the null-audit draw counts (one million random-direction draws and ten thousand shuffled-label draws for the shuffled controls), so we keep this phrasing as a low-end/null-tail observation rather than turning it into a load-bearing significance claim. This points weakly in the *opposite* direction from the random-null comparison: relative to generic label-correlated variance, the true success direction is *under*-represented in the injection span.

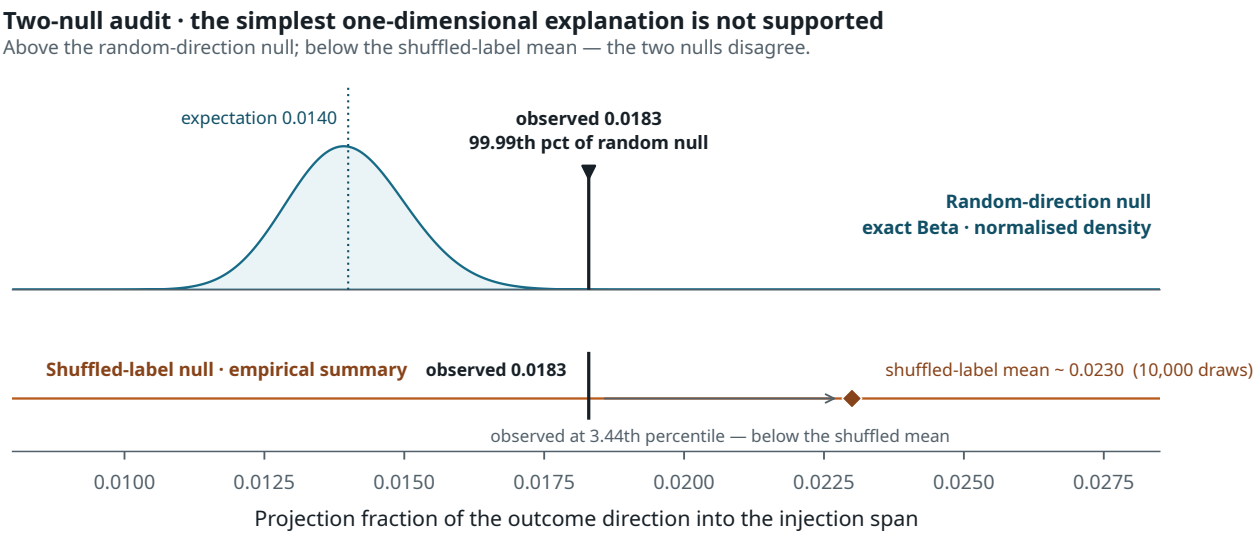


Figure 8. The two-null audit does not support the simplest one-dimensional span-misalignment explanation: the observed projection (0.0183) sits above the random-direction null and below the shuffled-label outcome-shaped null (weakly pointing the other way).

The two nulls disagree; the effect sizes are small (the gap between observed and shuffled-mean is ≈ 0.0044 of total energy); the estimate rests on a single one-dimensional projection from one regenerated delta bundle; and the injection span is effectively low-dimensional (participation ratio ≈ 5.06 despite nominal rank 344). A separate control confirms the injection span is statistically distinct from the natural sampling span — frozen forking is not merely resampling — but that answers a different question. We therefore do **not** treat this audit as evidence that the frozen null is explained by subspace misalignment. It enters the paper as a *simple explanation not supported by this audit*, not as support for the framing (Appendix J).

Crucially, this audit sharpens the boundary without proving the converse. It does not support reducing the observed control pattern to a one-dimensional linear span mismatch; broader subspace misalignment remains untested.

8.5 Synthesis: the boundary is empirical, not geometric

Directional steering is an established negative. The bounded four-task branch screen removes evidence for a frozen-fork gain but cannot estimate a general deficit. Terminal selection remains unresolved because the clean evaluations are underpowered: correctness is decodable, but no selector has yet been shown to convert detection into reliable commitment. A bounded training pass moves diversity without moving net reachability. The simplest one-dimensional span-misalignment explanation is not supported by the rank-corrected audit. The readout–control boundary is therefore an **empirical** pattern across multiple intervention types, not a demonstrated geometric obstruction.

Three simple explanations for the observed pattern were tested, which is what makes the boundary a result rather than an absence of one. First, it is not a mechanical implementation bug: zero-perturbation forks reproduce the reference at prefill and the suffix-recompute splice is bit-exact, so the branch/carry machinery preserves the intended computation to the relevant tolerance (§7.2–7.3). Second, it is not a sampling artifact: the K-matched deconfound shows the apparent sampled-fork gains are explained by sampling, and deterministic frozen forks produce no new correct solutions (§8.2). Third, it is not cleanly a linear-subspace mismatch: the rank-corrected projection audit returned conflicting random-direction and outcome-shaped shuffled-label nulls, supporting neither the "outcome outside the writable span" story nor its converse (§8.4).

We state the open mechanism plainly rather than paper over it. With the implementation and sampling confounds addressed, and the simplest geometric account not supported, the remaining bottleneck plausibly involves some combination of nonlinear propagation of perturbations through the loop, perturbation magnitude relative to the model's operating regime, decoding dynamics, loop-level instability, terminal selection policy, and — most importantly for what follows — the absence of any training-time objective that ties writable branch directions to verifier outcomes. The frozen model was never trained to make its injectable branches outcome-distinct, and nothing here suggests it would exhibit that alignment by accident. This motivates training-time branch-tournament integration, developed as future work (Section 12); we make no claim that such training has been run or that it will succeed, only that the frozen results locate the problem precisely enough to specify it.

9. Operational Proto-Introspection

We have deferred the paper's loaded term to this point on purpose. The reader has now seen the evidence: relational preference structure, role-specialized readouts, pre-answer success prediction, a correctness signal that is decodable but that no selector has yet been shown to convert into reliable commitment, an executable branching substrate, and a three-level frozen control boundary. Only now do we define the term those results collectively motivate, and defend it against the connotations it invites.

9.1 Definition

We say a hidden state is **operationally proto-introspective** if it contains externally readable information about the quality, stability, uncertainty, likely success or failure, or branch viability of the model's **own ongoing computation**, before external final judgment.

Three features of this definition do deliberate work. It is **operational**: it is a statement about what an external reader can recover from the state, not about what the model experiences or can say. It is about the model's **own ongoing computation**: the read object is the model's in-progress process trajectory, not the quality of some external artifact, which is why the pre-answer result (Section 5) is the definition's empirical anchor — the signal is available *before* the computation concludes and cannot be a reading of the finished answer. And it is **proto-**: a precursor to, and weaker than, the introspection the self-report literature studies, marking the gap rather than eliding it.

9.2 Why this is not merely probing

A natural deflation is that we have trained probes and dressed the results in psychological language. The grounds for rejecting it are specific. The signal is **pre-answer** (about ongoing computation, not finished output — the property that rules out the trivial reading, and the one §5 establishes), **role-specialized** (a structured set of distinguishable readouts — survivability, content quality, generated-branch correctness — not one competence scalar), **operationally consumed** (computed over the same cache a live branching scaffold manipulates, not analyzed offline), **causally load-bearing in at least one case** (ablating DualAnchor's layer-47 channel collapses oracle retention from 1.0 to 0.042 — the tap reads something the branch dynamics depend on), and **coupled to a control boundary** (no tested frozen intervention has produced a validated gain from the very signals that read success). No single one of these forces the interpretation, but their conjunction is more specific than "a probe works": it is a claim about *what kind* of information the model's process trajectory exposes and *where* it lives. Probing is the method; the object of study — externally readable process-quality structure in a model's own ongoing computation — is the contribution.

9.3 Why this is not self-report introspection

This is the distinction that most needs stating, because the field's term of art points elsewhere (Section 1.1). The self-report paradigm (Binder et al. 2024; Lindsey 2026; Comşa & Shanahan 2025) asks whether a model can **report** on its internal states and whether that report is causally grounded, and evaluates the report against criteria such as accuracy, grounding, internality, and metacognitive representation. Those criteria are defined over the *report*. We make no report claim at all: we never ask the model about its states, and our evidence is entirely about what an *external* reader recovers from *naturally-arising* activations. We therefore do not — and do not attempt to — satisfy the self-report criteria; our property is orthogonal to them, which is precisely why "proto-" is the honest prefix rather than "weak" or "partial." Where the self-report literature manipulates a representation and elicits a grounded report, we leave the representation untouched and elicit no report. The kinship is that both concern a model's relationship to its own internal states; the distance is that one is about articulable self-knowledge and ours is about external readability of process quality.

9.4 What we explicitly do not claim

We claim none of the following, and no result in the paper should be read as implying any of them: that the model is **conscious**, has **subjective experience**, or is **self-aware**; that the model **reports** on its states or possesses self-report introspection; that the model exercises **autonomous control** over its computation; or — importantly — that the model **uses** the readable signal internally. On the last point the paper's own evidence is the strongest disclaimer: the readout–control boundary (Section 8) shows that no frozen intervention we built produced a validated capability gain from these signals, so we are in no position to claim it does so on its own. Operational proto-introspection is a readout-side property.

It says the information is there and externally readable; it says nothing about the model having, using, or being able to talk about it.

9.5 Why looped models, and why the boundary matters

A scope note first, on what carries the framing and what does not.

The readouts qualify because of *what they read*, and this is easy to state imprecisely. The tap inputs contain hidden states rather than token IDs or decoded text. They are not reward models scoring finished text: they read the model's **hidden trajectory** — the loop states the model produces while computing — and nothing else. A tap therefore reads the model's own computation, from the model's own states, which is precisely the object the definition in §9.1 names.

Two of these readouts are additionally *forward-looking*, and they are the framing's strongest support because their timing forecloses the obvious deflation. The strict pre-answer probe (§5) reads a trajectory whose answer **does not yet exist** and predicts whether it will be correct; no part of the outcome is available to it, in code. The branch-survivability tap (§4, §6.2) reads an in-flight branch's states and predicts whether that branch will still contain a correct continuation — again, before the branch resolves, and, in DualAnchor's case, causally: ablating the channel it reads collapses oracle retention from 1.0 to 0.042. Both are readouts of unresolved computation, and both precede any external judgment of that computation. That is the property this paper names.

The remaining readouts — content ranking, generated-branch correctness — read the trajectories of computations that have already produced a candidate. They are process readouts on the model's own states, not text scorers, but they read a *completed* computation rather than one still in flight, and we treat them as supporting rather than anchoring evidence.

What the framing does **not** rest on is the branching substrate. The fork/carry/prune machinery of Section 7 is our construction, not the model's: it consumes readouts, it does not produce them, and Section 8 shows no frozen intervention through it produced a validated capability gain. Building a scaffold that reads a model's states is not evidence that the model introspects. The readouts are the evidence; the substrate is what allowed us to test whether reading becomes acting; no tested use of it produced a validated capability gain.

The evidence base is therefore a family of process-quality readouts on the model's own hidden states, with two forward-looking members, of which the strict pre-answer result is the cleanest and the only one measured on a single powered domain. If that result failed to replicate in a second domain, the framing would weaken substantially — though the branch-survivability readout, and the control boundary, would stand.

The property is natural to look for in looped transformers because repeated latent computation produces an internal trajectory — a sequence of intermediate states refining toward an answer — that a single-pass model does not expose in the same way. (We mean this precisely: the loop provides the trajectory *natively along depth*, by re-applying the same weights across iterations, whereas feedback approaches such as Coconut (Hao et al., 2024) can *induce* a latent trajectory in a non-looped model along the *sequence*, by spending token positions. The looped route is what makes the trajectory available at every position without consuming the output budget, and — per §7.6 — is also what gives injected branches the iterative depth to diverge.) The looped substrate is what makes "ongoing computation" a concrete, readable object.

This paper retracts a claim made here previously. A prior draft argued that the loop was necessary but not sufficient — that reasoning fine-tuning *installs* the readable content. The controlled replication of §3.5 does not support that: the linearly-readable relational direction is present across the whole Ouro lineage, base model included. What the loop supplies is the *trajectory* — an internal sequence of intermediate states, available at every position, with the iterative depth that lets injected branches diverge (§7.6) — not, on the current evidence, the readability of preference structure itself. Whether the *process-quality* signals that this paper's other readouts target (branch survivability, generated-branch correctness, pre-answer success) are similarly present in an untrained base model is an open question we have not tested; only the HH preference direction was replicated across backbones. Operational proto-introspection, as demonstrated here, is a property of a reasoning-trained looped model; which of its two adjectives is load-bearing remains unsettled.

One distinction keeps this consistent with the base-model literature. The Ouro authors argue their reasoning-trained latent states are *causally faithful* — perturbing an intermediate state changes the output (§7.2 of Zhu et al., 2025). Our readout–control boundary does not contradict this: causal faithfulness is a claim about *sensitivity* (the states are load-bearing, which is precisely why a readable signal exists at all), whereas our null is about *steerability* (the readable signal cannot be used by any frozen intervention we tested to move outcomes in a chosen direction). Our own machinery shows both: injected perturbations do change continuations (a no-carry branch diverges to RMS ≈ 3.0; §7.2), yet that sensitivity does not become controllable gain over sampling (§8). Sensitivity without steerability is, in fact, a compact restatement of the readout–control boundary.

And the control boundary is not a disappointing coda but a load-bearing part of the interpretation: it keeps the claim honest. A readout-only paper could be accused of over-reading a probe; a paper that *also* builds the machinery to act on the signal and finds no validated gain from the tested frozen action has, in effect, bounded its own claim from above. We even sought the clean geometric story that would have strengthened the frame — outcome direction outside the writable

span — and reported honestly that the audit did not support it (Section 8.4). The result is a claim scoped by its own negative evidence: readable, not usable; present, not exercised.

10. Synthesis

The paper's chain is short and each link is independently supported. Hidden states in a frozen looped transformer expose readable, role-specialized process-quality signals, and they are strong: branch survivability at 0.9697 oracle retention task-disjoint (with a causal ablation), content ranking at 0.6310, generated-branch correctness at AUROC 0.7755, and — most directly — prediction of the model's own eventual success before the answer exists, beyond surface shortcuts (Sections 3–6). A real executable branch/carry/prune substrate exists through which those signals could be acted on, validated by bit-exact identity (Section 7). On the control side, directional steering is an established negative; the bounded four-task branch screen removes evidence for a frozen-fork gain but cannot estimate a general deficit; and terminal selection remains unresolved because the clean evaluations are underpowered. A bounded training probe also does not establish a gain. The rank-corrected two-null audit does not support the simplest one-dimensional span-misalignment explanation for this pattern (Section 8).

The conjunction is the contribution. Each half alone would be unremarkable: readable probes are common, and the failure of frozen steering is unsurprising in isolation. What is not common is establishing both in the same system, with the signal demonstrably present, the machinery demonstrably correct, and the obvious confound (sampling) demonstrably controlled — so that the absence of a validated conversion cannot be dismissed as a weak probe, a broken scaffold, or a sampling artifact. The through-line is an asymmetry that had to be *earned*: **readable does not automatically become selectable, and selectable does not automatically become controllable.** We interpret it as weak operational proto-introspection plus a readout–control boundary (Section 9) — the model's own ongoing computation is externally legible, while no frozen intervention we tested produced a validated capability gain.

This also suggests a use for the taps beyond diagnosis. If the readouts are treated cautiously, they become candidate *latent reward models*: small functions over hidden trajectories that can provide intermediate credit assignment for preference, content quality, survivability, or branch correctness before a final answer exists. The control failures in Sections 6–8 are the reason this remains future work rather than a result: a readable tap is not automatically a safe or sufficient reward signal, and must be anchored to external verifiers and audited for reward hacking.

10.1 Controls, failures, and surviving claims

Because this paper's central claim is an asymmetry rather than a single positive score, the negative results and controls are part of the evidence rather than afterthoughts. Table 3 summarizes the main checks that changed, limited, or falsified a stronger interpretation. All entries are controls or failures reported elsewhere in the paper or appendices; rows with final live-repo path pinning still pending are marked as such rather than upgraded to final provenance.

Claim pressure-tested	Control or failure mode	Observed result	Surviving claim	Status
HH hidden states contain preference structure	Strict antisymmetrization of the fixed-order evaluator, **full 8,552-pair test set**	Fixed-order 0.9479 [0.9431, 0.9525]; strict antisym **0.6392** [0.6291, 0.6493]; symmetric/antisym magnitude ratio 1.50×	Preference structure is real; the fixed-order 95.2% is order-inflated and is discovery-stage only	**VERIFIED** on the full test set
Preference signal is relational, not pointwise	Matched pointwise probe on the identical pair-disjoint split	Relational linear 0.5653 vs pointwise linear 0.5418; paired Δ +0.0234 [+0.0132, +0.0334]	Relational decoding is more accurate, but preference IS available pointwise (0.5418 > chance) — the strong 'unavailable pointwise' claim is withdrawn	**VERIFIED clean**; the historical 21.75% was leak-inflated

Claim pressure-tested	Control or failure mode	Observed result	Surviving claim	Status
Readable relational signal is installed by reasoning training	Controlled cross-backbone replication (reconstructed *historical* probe protocol + original evaluator, identical config across backbones)	Reconstructed historical probe (row-level orientation split, 80 ordered rows / 74 source pairs — leak-style by design, for cross-backbone comparison only; its 0.8375 absolute value is **not** comparable to the clean §3.5 numbers) reads base/Thinking/RLTT identically, swap consistency 1.0; original evaluator gives ~95% canonical on base too (95.0/95.0/94.5) and flat antisym (58.0/59.5/60.0); historical base=24% not reproduced, no artifact	**Retracted.** Localization unestablished; the linear preference direction is present across the whole lineage including base — the clean cross-backbone values are §3.5's 0.5553/0.5653/0.5698	Failed replication reported in §3.5
Domain transfer is uniform	Science repair; source-specific breakdown; convergence-hair diagnostic	MMLU anatomy partially cleared (n=3); chemistry/physics/SciQ excluded, parse → 0.0; branches converge to a *no-good* branch (`CHEM_ANATOMY_NO_GOOD_CONFIRMED`)	Transfer is role/domain structured; the science limit is in branch **generation**, not evaluation — no selector can pick an oracle absent from the pool	**VERIFIED** (§4.4, §7.5)
Specialist taps always beat generalists	**Task-disjoint domain-transfer study** (zero crossing IDs; 360–4,748 held-out groups; task-clustered CIs)	Code→coding **0.9528** vs HH→coding 0.6944; HH→alignment **0.6902** vs code→alignment 0.5609; reasoning specialist 0.7671 ≈ balanced generalist **0.7613**; random-20 HH→alignment 0.6038	Specialization pays where distinctions are hard (code, alignment) and is unnecessary where a general quality axis suffices (reasoning); training-set scale matters independently of domain	**VERIFIED clean** (§4.4); supersedes a contaminated small-N study
Pre-answer success signal is just a shortcut	Length, log-probability, and composite controls; task-clustered bootstrap	Hidden+shortcuts AUROC 0.797 vs shortcut composite 0.731; incremental +0.066, **task-clustered** CI [+0.021, +0.112], excludes zero; leave-one-task-out [+0.056, +0.071]	Hidden states add pre-answer information beyond simple shortcuts	**VERIFIED** under the correct clustered test; primary evidence
Generated correctness can be turned into top-1 choice	Forced selection after S3B2 refit; exact Poisson-binomial matched-random baseline	Hidden ridge AUROC 0.7755 / pairwise 0.7338; sel@oracle 5/8 = 0.625 on N=8 groups vs matched random 0.3625, exact P(≥5) = 0.087	Selector exceeds the matched-random point expectation but N=8 is underpowered: reliable forced selection is **not established**	Detection **pinned**; baseline computed exactly (two earlier figures — 0.5833 and 0.625 — corrected)
Survival scaffold solves terminal arbitration	Task-disjoint re-run of the branch-survival evaluation (zero crossing task IDs)	Survival verified clean (stage retention 0.9697, terminal 1.0000); terminal-selection figures **withdrawn** — the clean remainder has only 2 reward-diverse tasks	Survival works; terminal commitment is unsolved **and currently unquantified**	Original evaluation had 8 crossing task IDs; re-run reported in §6.2, §6.5

Claim pressure-tested	Control or failure mode	Observed result	Surviving claim	Status
Branching gains reflect internal fork control	K-matched plain sampling deconfound	Sampled fork gains are explained by matched sampling; greedy/deterministic fork does not add new corrects	No gain in the bounded four-task comparison; a general deficit is unestimated	**Bounded negative screen**; exact params recorded in artifact report
Branch/carry mechanics are invalid	Zero-perturbation and suffix-splice checks	Prefill fork is bit-exact; cached decode has small bf16 drift; suffix recompute splice is bit-exact across 192 slots	The substrate is mechanically real, but mechanics do not imply control	Verified by ledger; full detail in App F
Steering can use readout directions as control vectors	Seven steering/adapters methods	No reliable signed capability gain; local adapter directions can be near-orthogonal without solving global control	Readout is not reliable steering control	Artifact-backed
Light training fixes branch reachability	Bounded 300-step LoRA probe	Coding parse improves 0.72→0.94 and math oracle@K 0.75→0.92, but macro reachability 0.708→0.688 due to logic/reasoning regressions	Light training changes behavior/diversity but does not solve the boundary	Verified by ledger
Frozen null is explained by simple linear orthogonality	Rank-corrected two-null subspace audit	Observed projection 0.0183 is above random-direction chance but low vs shuffled-label outcome-shaped controls	The geometric explanation is ambiguous and not load-bearing	**Pinned**; random-direction and shuffled-label draw counts resolved

The table is intentionally conservative. Several rows are negative: the stronger story fails, and the claim is narrowed. This is the pattern that makes the paper more than a collection of probes. The readout results survive shortcut, antisymmetry, transfer, and selection controls, while the control results survive matched sampling, steering, and simple subspace-explanation audits. What remains is not "the model can control itself,' ' but the narrower and better-supported statement that the model's looped hidden states are externally legible in ways frozen inference-time control cannot yet exploit.

10.2 Evidence-status map

For a reader who wants each major result's standing in one place:

Status	Results
Established positive	Strict GSM8K pre-answer increment (+0.066, task-clustered CI excludes zero); task-disjoint branch survival (0.9697; L47 channel causally load-bearing); generated-correctness detection (AUROC 0.7755, grouped split); corrected content ranking (0.6310 vs 0.5525); mechanical cache equivalence and the bit-exact splice
Established negative	No reliable signed steering under the seven tested methods
Bounded negative screen / positive interpretation removed	In the four-task comparison, frozen injected branches do not beat K-matched sampling and greedy forks add no new-correct answers; the screen is too small to estimate a general deficit
Unresolved	Terminal selection (underpowered in every clean evaluation); the mechanism of the readout–control boundary; whether training-time integration crosses it
Withdrawn / corrected	Leaked preference magnitudes (0.845, 0.2175); the "preference unavailable pointwise" claim; the fixed-order 95.2% as a relational result; training-stage localization (base=24%); contaminated terminal-selection and domain-transfer magnitudes
Diagnostic-only	Convergence hairs; the one-dimensional subspace audit; bounded-LoRA surface changes; the historical MATH-origin observation

11. Limitations

We state limitations directly; several are already integrated at the point of each claim, and we consolidate them here rather than confess them at the end.

- **A single powered pre-answer domain.** The central introspective result (Section 5) rests on GSM8K alone (170 tasks). This is the paper's most significant empirical limitation. A second powered pre-answer domain is the highest-value experiment we have not run, and its absence bounds how far the proto-introspection claim should currently be trusted.
- **The 95.2% fixed-order evaluator figure is not a relational result.** It was substantially reliant on a canonical-ordering prior, collapsing to **0.6392** strict antisymmetrized accuracy on the full 8,552-pair test set (§3.3). We report it only as fixed-order, discovery-stage accuracy. The strongest *clean* preference readout is the antisymmetrized nonlinear evaluator at 0.6392; the linear relational probe reads 0.5653.
- **Terminal selection is unsolved.** Generated-branch correctness is decodable (AUROC $\approx$ 0.78), and branch *survival* works well (0.9697 oracle retention, task-disjoint), but forced terminal commitment does not: the clean task-disjoint re-run leaves only two reward-diverse tasks, too few to quantify the gap in either direction (§6.5). The earlier figures supporting a quantified selection deficit were contaminated and are withdrawn. The selection wall is a genuine open problem, and it is now an *unquantified* one.
- **The orthogonality audit is not load-bearing.** The exact-protocol subspace audit (Section 8.4) is ambiguous under a two-null analysis and does not explain the frozen null. It does not rule out broader subspace misalignment. The readout–control boundary is empirical; we do not have a mechanistic account of it.
- **Frozen results only.** All control findings are for the frozen backbone under tested methods and magnitudes. We have run no training-time integration (no S3A/S3C), and make no claim about what such training would yield.
- **No capability, control, report, or consciousness claims.** We establish no capability gain from frozen branching, no autonomous control, no self-report, and nothing about subjective experience or awareness. The scope is a readout-side property and its boundary.
- **Retracted claims, and two systematic causes.** Five figures reported in this project — including three in the prior published paper (Kirin, 2026a) — were distorted (four inflated, one deflated below chance): four by source-item leakage across the train/test split, and one (the 95.2% fixed-order evaluator) by a presentation-order prior that no split check would catch. One further claim did not reproduce at all. The relational linear probe (84.5% $\rightarrow$ 0.5653), the pointwise linear probe (21.75% $\rightarrow$ 0.5418), CoreContent v2 (0.6691 $\rightarrow$ 0.6310), and branch survival (0.9848 $\rightarrow$ 0.9697) were each corrected by splitting on *source items* rather than constructed rows; the fixed-order evaluator's 95.2% was separately inflated by a canonical-ordering prior ($\rightarrow$ 0.6392 antisymmetrized); and the training-stage localization (base 24%) has no surviving artifact and does not reproduce. The terminal-selection magnitudes that had rested on the contaminated branch-survival split are withdrawn without replacement (§6.5). We retract the strong claim that preference is *unavailable* pointwise (it is decodable at 0.5418, above chance) and the claim that reasoning fine-tuning *installs* the readable signal. The corrective protocol is stated in §3.7; the full audit anatomy is the companion methodology paper, and the correction of record for the prior paper is the erratum to arXiv:2604.09870. What survives is a smaller, cleaner set of results: preference is decoded more accurately relationally than pointwise (+0.0234), the antisymmetrized nonlinear evaluator reads 0.6392 on the full test set, and the full training pipeline modestly raises the linear signal (+0.0145, RLTT vs base).
- **The readouts do not require a looped architecture.** A non-looped SFT transformer reads candidate quality at 0.5680 under a task-disjoint split (§4.6). The readout side of this work is therefore not a property of recurrence. We also do not claim looping is irrelevant: the Ouro-vs-control comparison is not architecture-controlled (different family, corpus, objective, width, tap geometry), so the 6.3-point gap is unattributed. A causal architecture study would need matched looped/non-looped models trained identically.
- **The corrected CoreContent coding figure is mutant-only.** The corrected task-disjoint coding value (0.8956) is measured against deterministic mutants; a corrected task-disjoint *relevance* evaluation has not been run, because the relevance negatives were generated inside the old splits and were not regenerated. The stored-split finding that the coding tap is a corruption detector rather than a relevance judge (0.94 vs 0.58) is retained as qualitative, not re-quantified.
- **Two candidate second domains were rejected at preflight.** The single-domain limitation above is not for want of trying: SVAMP proved unusable because its answers are front-loaded (parser and label checks disagreed), and Hendrycks MATH, while supplying genuine long reasoning, produced ~21/22 correct among parseable generations — most failures were non-commitment or truncation, so dropping truncations removes the negative class while labelling truncation as failure degenerates the task into a length predictor. We document this as a negative decision rather than an unrun experiment; a suitable second domain needs a different protocol, not simply more compute.

- **Provenance items.** The strict pre-answer result, the full-set antisymmetry audit, the S1 K-matched decimals, the two-adapter convergence cosine, and the steering seven-method closure are now verified against live-repo artifacts. The math-transfer origin figures (§3.6) remain unarchived and are retained only as non-load-bearing origin motivation.

What would falsify or strengthen this framing. We state the conditions under which the paper's claims should be revised, since a framing worth defending should be one whose failure modes are namable. The proto-introspection framing would be *weakened* if a second powered pre-answer domain removed the hidden-state incremental gain; if S3B2 generated-branch correctness collapsed under grouped or task-held-out splits; or if the strict pre-answer effect failed to replicate under an independent implementation. (Two candidate weakeners have already been tested and did *not* materialize: the full 8,552-pair antisymmetrization audit confirmed rather than eliminated the relational preference result, and the task-clustered bootstrap confirmed rather than dissolved the pre-answer increment.) It would be *strengthened* by a second powered pre-answer domain that preserved the incremental gain; by a trained branch-control objective that converts readout into actionability (crossing the boundary rather than describing it); or by replication of the readout and boundary in a different looped or depth-recurrent architecture. A properly pair-split antisymmetric replication across the Ouro lineage — larger than the one reported in §3.5, and with the original probe weights preserved — would settle the training-stage question that this version leaves open.

12. Future Work

The frozen boundary specifies its own next step. Because control fails not for lack of readable signal but plausibly for lack of a training-time objective tying writable branch directions to outcomes, the natural intervention is **training-time branch-tournament integration** (S3A): training the model so that its injectable branch directions become outcome-distinct, aligning the branch-control manifold with the already-readable outcome geometry. We frame this as the motivated next experiment, not a result. External evidence makes the hypothesis concrete: Coconut (Hao et al., 2024) is a clean case in which latent-space exploration became *usable* precisely because it was trained into the model rather than bolted onto frozen inference — consistent with the boundary we observe, where reading and acting on analogous signal in a frozen model confers no gain. Coconut demonstrates only the positive half (trained latent exploration is usable); the frozen-fails half is supplied by our own results, not by Hao et al., who trained latent reasoning from the start and did not test a frozen variant. The two halves together — their trained success and our frozen null — are the pattern S3A is designed to reproduce for internal branch control.

12.1 Taps as latent reward models

A natural next use for the tap family is **reward learning over latent computation** rather than only over completed text. Standard reward models usually score final responses, or occasionally explicit process traces written in tokens (Lightman et al., 2023). The readouts in this paper score a different object: hidden-state trajectories, pairwise candidate differences, and branch states inside the looped computation. This makes them plausible auxiliary rewards for branch-tournament RLTT. A DualAnchor-like tap can reward retaining branches that still contain an oracle continuation; a CoreContent-like tap can reward content-quality improvements; an S3B-style tap can reward generated-branch correctness; and a strict pre-answer success tap can provide early credit before the answer string reveals the target value. In this interpretation, the taps do not replace task verifiers or preference labels. They densify them, turning sparse final supervision into intermediate signals over the latent trajectory that produced the answer.

This use case is also where the readout–control boundary becomes practically important. The same results that make taps attractive as reward features make them dangerous as unanchored objectives. S3B2 shows that a correctness signal can be decodable while forced selection remains weak; the frozen branching and steering results show that externally readable directions are not automatically usable as control directions; and the orthogonality audit shows that a simple linear geometric story is not enough to explain the boundary. A training system that uses taps as rewards should therefore keep external verifiers as the final authority, use grouped heldout tasks to check generalization, calibrate tap margins before converting them into reward, and include explicit reward-hacking controls such as shuffled labels, metadata-only baselines, adversarial branch pools, and ablations where the tap reward is withheld. The intended proposal is not "optimize the tap and trust it," but rather **verifier-anchored latent reward shaping**: use taps to assign credit inside the model's computation while retaining external correctness, preference, or safety evaluations as the arbiter.

In the context of alignment and monitoring, this also makes the taps useful as *diagnostic reward models.* A tap can mark where in the latent trajectory a branch becomes low-quality, unstable, self-contradictory, or likely to fail, even when the final answer is still recoverable. That opens a path to process-level datasets built from hidden trajectories rather than only text completions: reward the trajectory for preserving correct branches, penalize premature collapse or false pruning, and train the model to make branch states more separable before final decoding. The present paper stops short of that training step. Its contribution is to show that such latent reward features are readable and structured, and that frozen

inference-time control is not enough to exploit them.

12.2 Other uses for taps: monitors, routers, and data engines

The reward-learning use case is the central one, but it is not the only plausible role for taps. More generally, a tap is a small, auditable sensor over a latent trajectory. That makes it useful anywhere a system needs to decide what to do with an ongoing computation before the final answer is available. One use is **adaptive compute allocation**: a pre-answer success tap could decide whether to stop, continue looping, branch, ask for a verifier, or defer to a stronger model. Another is **candidate routing**: survivability and correctness taps can cheaply filter large branch pools before expensive external verification. A third is **active data selection**: branches with high disagreement between a tap and an external verifier are precisely the examples that should be added to the next training or calibration round.

A further use is **hidden-state monitoring**. Textual chain-of-thought monitoring is useful when reasoning is externalized, but recent safety work emphasizes that monitorability is fragile: relevant computation may be absent from the visible chain, compressed into latent state, or deliberately made hard to read. Hidden-state taps are not a replacement for chain-of-thought monitors, but they are a complementary instrument: they can look for process-quality, inconsistency, branch collapse, or likely failure in the latent computation itself. This connects the present work to representation-engineering and activation- monitoring methods, where simple directions or population-level representations have been used to detect or influence high-level properties such as truthfulness, refusal, and safety-relevant behaviors. The important difference is role and timing: our taps are trained on looped process trajectories and are intended to read the quality of an unfolding computation, not only a static representation of a completed prompt.

A related use is **AI-control monitoring**. In control protocols for untrusted agents, monitors decide when to defer, audit, resample, or block actions. A hidden-state tap could provide an additional monitoring channel: not "what did the model say?" but "what did the computation look like while producing it?" This could be useful for detecting low-confidence success, suspiciously unstable branches, or cases where the final answer looks clean but the latent trajectory resembles known failure modes. The same adversarial caveat applies even more strongly here than in reward learning. If a model can observe or optimize against the tap, the tap can become a target. Any safety use would need heldout taps, randomized monitor ensembles, adversarial evaluations, and protocols that deny the untrusted policy direct access to the monitor internals.

Finally, taps can serve as **scientific measurement tools**. They let us ask when a capability becomes readable during training, which loop/layer loci carry it, whether a signal transfers across domains, and whether a proposed intervention changes the internal trajectory before it changes the output. In this paper, that measurement role is already visible in the controls-and-failures ledger (§10.1). Future work could turn it into a standard diagnostic suite: before claiming that a new latent-reasoning method improves reasoning, measure whether its hidden states become more separable for survivability, correctness, and pre-answer success, and whether those signals remain calibrated under distribution shift.

12.3 Taps and recursive improvement

Taps are also relevant to recursive self-improvement, but only in a limited, verifier-anchored sense. A recursively improving system needs to choose among proposed data mixtures, prompts, branch policies, training objectives, verifier designs, or model edits. Final-output evaluation alone is expensive and sparse; a tap can provide a cheap first-pass critic over the internal computation produced by each proposal. In such a loop, taps could filter proposals, assign dense credit to promising latent trajectories, flag internal degradation before it shows up in aggregate benchmarks, and prioritize which candidate changes deserve external evaluation.

This is not a claim that taps enable autonomous RSI. The results of this paper argue against that interpretation. Readable process-quality signals do not become frozen control, and a system trained to maximize tap scores directly would be vulnerable to Goodharting the tap. The safe RSI-adjacent role is therefore evaluator-layer support: taps can make improvement loops more sample-efficient by identifying which internal trajectories are worth checking, while external verifiers, heldout tasks, adversarial branch pools, and tap-withheld audits remain the authority. In short, taps could be part of the critic and monitoring layer of a recursive-improvement system; they are not the self-improvement engine.

12.4 The experiment queue, in priority order

Ordered by how much each would change the paper's standing rather than its polish:

1. **A second powered pre-answer domain.** The only outstanding item that alters the scientific weight of the primary result rather than its presentation. Two candidates were tested and rejected at preflight for reasons about the datasets, not the effect (SVAMP front-loads its answers; Hendrycks MATH degenerates into a length predictor once truncation is handled — §11), so this needs a *different protocol*, not more compute. Until it exists, the proto-introspection framing rests on one domain.

2. **An adequately-powered terminal-selection evaluation.** The clean task-disjoint re-run left only two reward-diverse tasks (§6.5), so the terminal bottleneck is currently a finding without a magnitude. A larger reward-diverse pool would either quantify the deficit or overturn it; both are useful, and we currently cannot distinguish them.
3. **Training-time branch integration (S3A).** The route the frozen boundary motivates (§12 opening). A selector at scale, with a training-time objective binding writable branch directions to verifier outcomes, is the direct test of whether the boundary is crossable at all.
4. **A properly pair-split antisymmetric replication across the Ouro lineage**, with the probe weights preserved, to settle the training-stage question §3.5 leaves open (the observed RLTT − Base effect is real but small, and no single stage is attributable).
5. **A subspace-vs-subspace orthogonality audit** (principal angles / CCA) rather than the single 1-D projection of §8.4, with confirmation that the outcome direction used is the one the taps actually read.
6. **Read/write robustness gates** to characterize the steering envelope more finely than the tested safe-α band, and a **minimal independent replication** of the pre-answer and boundary results.

We name the longer-horizon integrated system (Jormungandr) only as a direction, with no capability claim attached.

13. Conclusion

We asked whether the intermediate states of a frozen looped transformer carry readable information about the quality of the model's own ongoing computation, and whether that readability confers control. The answers are yes and no.

Hidden states expose role-specialized process-quality signals that low-capacity taps recover from frozen representations and that a live branching scaffold consumes: branch survivability at 0.9697 oracle retention under a task-disjoint split (with the layer-47 locus causally load-bearing — ablating that channel collapses retention to 0.0417), content ranking at 0.6310 across five domains, generated-branch correctness at AUROC 0.7755, and — the paper's primary result — a strict pre-answer probe that predicts the model's own eventual success on GSM8K before the answer exists, adding significant information beyond length and log-probability shortcuts, with an interval that excludes zero under the correct clustered test and no single task driving it. A same-class candidate-quality readout also appears in a **non-looped** SFT transformer (0.568 task-disjoint), showing that this form of process-quality readability does not require recurrence. We report that against our own framing's interest. We then built the machinery through which such signals could be used, validated by bit-exact identity, and found that no frozen intervention we tested produced a validated capability gain: directional steering is an established unsigned negative; in a bounded four-task screen, injected branches do not outperform matched sampling; no selector has yet been shown to convert detection into commitment even where the correct branch is demonstrably present; and the simplest one-dimensional geometric explanation for the pattern is not supported by a two-null audit. The branch-level and selection evaluations are underpowered, so those two levels remove the positive interpretation without establishing a broad null; the directional negative is the one established across many methods. We name the readable-but-not-usable property operational proto-introspection, defined narrowly against the self-report introspection literature rather than as an instance of it, and take the readout–control boundary — not any capability, control, or awareness claim — to be the paper's central contribution.

Every load-bearing current quantitative claim in this paper is reported under the audited protocol (§3.7): source-item-disjoint splits with zero-crossing integrity checks, and antisymmetrized evaluation for fixed-order pairwise scorers. Historical or diagnostic quantities with incomplete provenance are explicitly marked and are not load-bearing. The audit that produced that protocol — two evaluation traps, five corrected figures, three of them in the prior published paper — is reported in full as a companion methodology paper, and the prior paper's correction of record is the erratum to arXiv:2604.09870. The corrected story is less dramatic, and the surviving effects are real.

The most important open question is whether training-time integration can cross a boundary that no frozen intervention we built could. The most important open caveat is that the pre-answer result stands on a single powered domain.

APPENDICES

Complete reproducibility record

Repository: `github.com/VykosMolt/ouro_project`; reproducibility commit and per-result artifact paths in Appendix K. Where a figure has been superseded by a corrected re-run, the appendix reports the corrected value and marks the original withdrawn rather than deleting it.

Appendix A — Hidden-state extraction and feature formats

Feature construction uses hidden trajectories from the frozen looped backbone. The canonical feature basis taps layers **24, 36, and 47** across four loop states L1–L4 at width 2048, giving $D = 3 \times 4 \times 2048 = 24{,}576$ when concatenated. Role-specific variants use final-loop L4, mean-over-loops, L1/L4 fusion, or full loop-concatenation; the main text reports which variant is used where. The 24/36/47 basis is the general-purpose default; one role is an exception — the locked CoreContent terminal selector prunes layer 47 as dead weight and uses a 2-channel 24+36 tap (Appendix E.2.2), which slightly outperforms the 3-channel version on real negatives. Layer 47 remains load-bearing for DualAnchor's looped survival (its perturbation is not diagnostic-only there; Appendix E.2.1), so the basis is not uniformly reducible — 47 matters for survival and not for terminal content ranking.

The 24/36/47 basis comes from the historical locus program summarized in `evaluator-locus-summary.md`. That program began with pairwise locus and loop ablations, passed through normalization/bias decomposition and all-layer cached probes, and only then settled on 24/36/47 after the v10 Thinking-vs-RLTT loop-geometry analysis. The goal was to preserve a small mid/late/final trajectory: a mid-depth anchor (24), a late integration anchor (36), and the terminal/pre-output boundary (47). Full all-layer extraction would have been much more expensive and would have made every downstream audit harder to repeat; the three-locus basis retained the useful loop-localized readout signal while keeping tiny taps and repeated controls tractable. In the DualAnchor architecture-looped branch-selection line, the same loci were promoted from a static feature basis into a twelve-stage schedule over the four loop iterations: `L1_24 -> L1_36 -> L1_47 -> L2_24 -> L2_36 -> L2_47 -> L3_24 -> L3_36 -> L3_47 -> L4_24 -> L4_36 -> terminal L4_47`.

For the original evaluator, hidden states were captured by a forward hook on the Ouro model body rather than relying on generic `output_hidden_states` plumbing, because the local Ouro wrapper exposes the loop trajectory through model-specific outputs. The HH-RLHF data, the extraction procedure, and the ~5M-parameter evaluator head follow the setup of Kirin (2026a); we reuse it unchanged except for the role-specific feature variants introduced here. Supporting: `interfaces-and-tools.md`, `evaluator-locus-summary.md`, `chronological-evaluator-summary.md`, Kirin (2026a, arXiv:2604.09870), hidden-state extraction scripts.

Appendix B — Pairwise evaluator and HH flip/antisymmetry audit

Evaluator architecture. Fixed-order pairwise evaluator: token attention pooling per loop state, normalized candidate differences, projection to 512 dimensions, a two-layer GRU across the four loop states, and a nonlinear scorer (full detail in Appendix C). The historically selected epoch-2 checkpoint preserves the fixed-order accuracy peak: 83.3% → 95.2% → 62.4% across epochs 1/2/5 (Kirin 2026a). The audit shows that this peak combines genuine relational signal with a maximal transient presentation-order prior and must not be described as a clean generalization peak; later epochs overfit.

Full-test-set antisymmetry audit (8,552 pairs). The evaluator's canonical-order accuracy reproduces (0.9479, 95% CI [0.9431, 0.9525]; the historical 8,141/8,552 = 0.9519 lies inside this interval), but its **strict antisymmetrized accuracy is 0.6392** (95% CI [0.6291, 0.6493]).

Measurement	Full 8,552-pair result
Fixed-order (canonical) accuracy	0.9479
Swapped-direction accuracy	0.1954
Strict antisymmetrized accuracy	**0.6392**
Strict sign-flip rate	0.2475
Normal/flipped score correlation	−0.9247
Both orders prefer the first argument	75.25%
Symmetric (order) component, mean	+1.2649
Symmetric / antisymmetric magnitude ratio	1.50×

All 8,552 test indices present exactly once; no duplicate, missing, or non-finite rows. The evaluator is *not* degenerate — swapped scores remain strongly anti-correlated with canonical ones — but the positive first-position offset dominates the sign for most pairs. Pooling and normalization ablations (§3.3) confirm that attention pooling is not the cause and that the trained difference-LayerNorm *restrains* rather than creates the order effect.

Linear probes. The L-BFGS difference probe scores $w^\top(h_A - h_B)$ with no bias, hence exact antisymmetry. Clean pair-disjoint result: **0.5653**. The historical 84.5% was inflated by orientation-row leakage (§3.7) and is retracted.

Swap-protocol training-metric deflation. Documented in Kirin (2026a): the antisymmetry-enforcement protocol masked the pairwise model's capability across seven consecutive runs, with the deflated training metric inversely correlated with test performance.

Supporting: `flip-test-interpretation.md`, `chronological-evaluator-summary.md`, full-HH audit artifacts (`full_hh_antisymmetry.json`, `full_hh_antisymmetry_rows.csv`), Kirin (2026a, arXiv:2604.09870).

Appendix C — Tap architectures and training details

The paper uses the term *tap* for small readout heads trained on frozen hidden-state features. This is deliberately narrower than the original evaluator. The Kirin evaluator used token attention pooling for each loop state, normalized candidate differences, projected them to a 512-dimensional hidden space, ran a two-layer unidirectional GRU over the four loop states, concatenated the GRU output with the final-loop projection, and passed the result through a nonlinear scorer. That architecture was useful as a discovery tool because it could integrate information across the loop trajectory, but subsequent locus and swap-control work showed that the GRU was repeatedly weak or mildly counterproductive relative to simpler exact-antisymmetric heads. It remains a control/escalation path, not the default tap architecture.

The simplest comparison taps are `AntisymLinear` and `AntisymLinearNoNorm`. Both form a pairwise difference

$$\Delta h = h_A - h_B,$$

and then apply a bias-free linear readout so that swapping candidates flips the score sign. The normalized variant scores

$$s_{\mathrm{LN}}(A, B) = w^\top \mathrm{LN}(h_A - h_B),$$

whereas the NoNorm variant scores

$$s_{\mathrm{raw}}(A, B) = w^\top (h_A - h_B).$$

Both are structurally antisymmetric: $s(B, A) = -s(A, B)$ up to numerical tolerance and the exact details of the symmetric preprocessing. The difference is interpretive. `AntisymLinear` suppresses raw scale and norm effects and asks whether the *direction* of the pairwise difference carries the label; it is the safer comparator for hard near-miss distinctions where magnitude can be a shortcut. `AntisymLinearNoNorm` preserves raw magnitude and often behaves more like a scalar utility readout; it is useful when quality is relatively transitive or when easy-prune/objective distinctions are carried by feature scale. We keep both families as complementary probes rather than treating either as universally superior.

This distinction is part of the evaluator-to-tap transition. The large GRU evaluator first revealed the existence of a loop-trajectory signal, but its capacity and fixed-order training made strict antisymmetry auditing mandatory. The tap family was designed so that the comparison rule itself is auditable: if a head says A beats B, the same head must say B loses to A. Success under that constraint is therefore stronger evidence for readable hidden geometry than success by a large order-sensitive evaluator. Tap head sizes, optimizer/effective-batch settings (EBS = 32), epoch selection and overfitting behavior, DualAnchor and CoreContent head definitions, and pointwise-vs- pairwise probe setups (the clean pair-disjoint pointwise result is 0.5418; the historical 21.75% was pair-leaked, §3.7) are tracked in the supporting training scripts and ledgers. Supporting: tap training scripts, `chronological-evaluator-summary.md`, `evaluator_pairwise.py`, Kirin (2026a, arXiv:2604.09870).

Appendix D — Domain transfer and role-separation tables

Clean (task-disjoint) domain-transfer study. Deterministic split, **zero task IDs crossing** the boundary; held-out sets from 360 coding groups to 4,748 alignment groups; task-clustered bootstrap intervals. This is the reportable study; it supersedes the contaminated one below.

Tap trained on	Evaluated on	Top-1	Pairwise	Reading
Code	coding	**0.9528**	**0.9650**	strong in-domain code specialization
HH (general)	coding	0.6944	0.8727	general head transfers, substantially weaker
HH (general)	alignment	**0.6902**	0.6831	alignment specialization is real
Code	alignment	0.5609	0.5538	code specialization does not replace preference
Reasoning	reasoning	0.7671	0.8870	reasoning specialist
Balanced (all-core)	reasoning	**0.7613**	0.8827	generalist matches the specialist — no reasoning tap needed

Tap trained on	Evaluated on	Top-1	Pairwise	Reading
Random-20 HH subset	alignment	0.6038	0.5968	data-scale effect, independent of domain

Summary: specialization pays where the within-domain distinction is hard (code, and to a lesser degree alignment) and is unnecessary where a general quality axis suffices (reasoning). Training-set *scale* is a separate axis from domain match.

Superseded (contaminated) study — retained for transparency, not for use. An earlier domain-transfer table reported code-trained 0.875/0.833 vs HH 0.750/0.600 on strict-clean code; HH all-200 0.855 vs code-trained 0.535; reasoning 0.960/**0.986**; HH→code transfer 0.500/0.571; and random-20 means 0.655/0.640. Those evaluations were small-N (tournament counts in the tens; the 0.986 rests on 25 tournaments) and pre-dated the task-disjoint discipline of §3.7 — at least one branch dataset had task IDs appearing on both sides of the split. **They are withdrawn.** The qualitative direction survived the clean re-run; the magnitudes did not. We list them here so that readers of earlier drafts can locate what changed.

Science: partial repair, source-specific failure. Source-specific repair partially cleared MMLU **anatomy** (held-out positive-oracle 0.333, parse 1.0, n = 3 tasks) while **chemistry, physics, and SciQ** stayed excluded, with parse rates collapsing to 0.0. The **convergence-hair** diagnostic (§7.5) locates the cause: on chemistry and anatomy the branches converge to a *no-good* branch (`CHEM_ANATOMY_NO_GOOD_CONFIRMED`) — the model does not generate a correct branch for a tap to find, so the limit is in branch *generation*, not branch *evaluation*. Reasoning, not science, became the headline scope.

Layer/loop localization. Canonical layers 24/36/47 with L1–L4 loop variants; layer 47 carries the largest loop spread while 24/36 are more loop-converged (§4.3, Appendix A).

Supporting: task-disjoint domain-transfer re-run (2026-07-13); `domain-transfer-ledger.md`, `core-domain-tap-audit.md`, `science-reasoning-repair.md`, `evaluator-locus-summary.md`, `chronological-evaluator-summary.md`.

Appendix E — DualAnchor / CoreContent / S3B details

E.1 Pre-DualAnchor survival scaffold

Before DualAnchor, the branch-survival line passed through fixed-composite and selection-only scaffold experiments. These are included because they show the same survival/selection split before the later DualAnchor naming and architecture-looped baseline.

Scaffold / policy	Main retention result	Failure mode / caveat	Status
`fixed_composite_conservative_top4`	oracle retention 0.931; false-prune 0.069; avg survivors 3.873	**measured pre-audit**, on the same split family later found to have crossing task IDs; treat as historical, not as a clean estimate	`SURVIVAL_READY` (verdict stands; magnitude unaudited)
old-context/coding subset	retention 1.000; coding false-prune 0.000	subset-specific; pre-audit; not a general selector	supporting diagnostic
selection-only Phase 2 prototype	fixed top4 oracle retention 0.9514; false-prune 0.0486; avg survivors 3.9109	terminal-reward figures **withdrawn** (contaminated split; see E.3)	`SURVIVAL_READY_FINAL_ARBITER_WEAK` (verdict stands; magnitudes do not)

These results are the pre-DualAnchor ancestor of the later selection wall. They support the claim that survival and terminal arbitration separated early: top-k pruning could retain oracle branches at high rates, while the final arbiter remained weak. Supporting: `branch-generation-and-survival.md`, `current-state.md`.

E.2 DualAnchor, CoreContent, and S3B2

E.2.1 DualAnchor — genesis and construction. DualAnchor is the branch-survival / retention family (scores, prunes, and forwards survivors during the looped branch/prune search), not a terminal correctness selector. It is the two tap identities `MIX_CODE_REASONING` and `MIX_OBJECTIVE_ALL` — the strongest existing content/action readouts — with branch-validity grafted on by weight-space transplant, so that two "dual-anchored" taps carry content and branch-viability together rather than being split across separate content, branch, and bridge taps. The design reached its locked form through an incremental lineage, each step with its own diagnostic verdict:

Step	Verdict	What it established
old-anchored branch-valid taps v1	`OLD_ANCHORED_BRANCH_TAP_USEFUL`	weight-space transplant adds branch/bridge gain without dropping content/code performance
two-tap branch selector v1	`TWO_TAP_BRANCH_SELECTOR_READY`	the two taps score cached branch-survival groups with high retention (heldout retention 0.9825, false-prune 0.0175)
fresh-dataset comparison v1	`TWO_TAP_MATCHES_OR_BEATS_OLD`	on fresh same-content domains, new two-tap ≈ beats old (0.8818 vs 0.8800; 192 tasks / 742 candidates / 550 pairs)
HH-RLHF comparison v1	`TWO_TAP_MATCHES_OR_BEATS_ON_HH`	on 512 HH pairs, modestly beats old (0.6152 vs 0.5977)
layer-native two-tap v1	`DOMAIN_READY_BRANCH_GAP`	native 24/36/47 taps preserve domain behavior but retain branch gaps
branch-gap repair v1	`DUALANCHOR_BRANCH_GAP_REDUCED`	repair narrows the gap without a clean fixed-bundle pass
architecture-looped v3	`READY_WITH_TERMINAL_DEFER`	all-loop repeated branch/prune survival works; terminal forced commitment is the weak point (magnitudes later withdrawn, E.3)

Architecture-looped v3 diagnostic (48 tasks; 24 reasoning / 24 science; 3,454 rows): stage oracle retention **0.9697** (task-disjoint, zero crossing task IDs; supersedes a contaminated 0.9848 measured with 8 crossing IDs and 26/48 training-side tasks), terminal oracle retained **1.0000**, scaffold top-4 retention **1.0000** (52 groups / 26 tasks). Terminal-selection figures from the contaminated evaluation (forced top-1 oracle 0.9167, reward-diverse 0.6364, rewards 0.2625/0.3167) are **withdrawn**; the clean remainder has only 2 reward-diverse tasks of 9. An L47 ablation confirms earlier-loop L47 perturbation is load-bearing, not diagnostic-only: disabling it collapses oracle retention from 1.0000 to 0.0417. The locked policy is confidence-gated top-1 with defer / top-k handoff rather than unconditional forced top-1 — the survival-without-terminal-selection pattern of §6.

E.2.2 CoreContent — genesis and construction. CoreContent is the content / terminal-selection family: it ranks candidates *within* the handed-off survivor set, using the same digest-and-compare engine as the Section 3 preference evaluator (frozen forward, mean-pooled loop states, antisymmetric linear tap over layernorm($\text{state}_i - \text{state}_j$)), generalized from HH pairs to five domains. Its construction is the inverse of DualAnchor's — data-driven, not transplant:

- **v1 (tap crafting):** every crafted content tap *lost* to the broad-objective baseline `mixedhead_MIX_HH_OBJECTIVE`. Diagnosed cause: starved per-domain data (reward-diverse groups: coding 30, reasoning 5, math 66, logic 80, alignment 200).
- **v2 (dataset expansion + refit, 2026-06-04/06):** the starved domains were expanded 27–520× (reward-diverse groups: coding 1,733, reasoning 2,600, math 3,200, logic 2,199, alignment ~25,993; sources include MBPP/APPS/HumanEval, GSM8K/Hendrycks/SVAMP, LogiQA, ARC/OpenBookQA/CommonsenseQA/ StrategyQA, HH / UltraFeedback / SHP / PKU), features re-extracted (64 shards, 4.87 GB, leakage found and fixed), and the same small taps refit. A crafted tap then beat the baseline on untouched held-out data:

Corrected (task-disjoint) result — the reportable figure. Under a deterministic task-disjoint split (seed 20260711; 23,054 train / 2,948 validation / 2,831 held-out tasks; **zero** task IDs crossing splits):

Quantity	Value
Corrected held-out macro top-1	**0.6310**
Broad-objective baseline (`mixedhead_MIX_HH_OBJECTIVE`)	0.5525
Historical stored-split figure (superseded)	0.6691
Task IDs crossing splits, stored / corrected	195 / 0

Corrected per-domain held-out top-1: coding **0.8956** (n=182), math **0.6409** (298), reasoning **0.5985** (269), alignment **0.6143** (2,564), logic **0.4057** (212).

Important scope note on the corrected coding figure. The corrected coding value (0.8956) is **mutant-only**: every corrected held-out coding group contains a canonical solution plus deterministic mutants, and none contains wrong-problem candidates. A corrected task-disjoint *relevance* evaluation has **not been run** — the relevance negatives were generated inside the old stored splits and were not regenerated after task-disjoint reassignment (evaluating the corrected policy against that stale cache gives 0.5165, but every corrected held-out target draws at least one donor from outside

its split, so the number is not publication-clean). The 0.8956 must therefore **not** be read against the historical 0.583 as a matched comparison.

Historical stored-split diagnostics (superseded; retained for the qualitative finding). The following per-domain table was measured on the contaminated stored split and is reported only because it establishes a qualitative property that the corrected run does not re-measure:

Domain / negative type	CoreContent v2	Mixed baseline	Δ
coding constructed mutants	0.935	0.613	+0.32
coding real wrong-problem relevance	0.583	0.502	+0.08
math constructed perturbations	0.657	0.586	+0.07
logic real distractors	0.460	0.353	+0.11
reasoning real distractors	0.681	0.677	+0.00
alignment real preference pairs	0.612	0.534	+0.08

Honest limitations. On the stored split, the macro headline was roughly *half* a constructed-negative artifact: restricted to real-negative domains (reasoning/logic/alignment) the edge over the baseline was **+0.063**, with CIs barely disjoint. The coding tap learned *corruption detection*, not prompt relevance: ~0.94 against syntactic/semantic mutants but **0.58** against real, compiling solutions to *other* problems, and retraining it *with* wrong-problem negatives did **not** close the gap (0.593 → 0.583), indicating prompt-code relevance is not linearly accessible in the pooled features — an intrinsic ceiling, not a data artifact. This corruption-detector-not-relevance-judge characterization is the qualitative finding we retain; the corrected split does not refute it, but neither does it currently re-quantify it, and we say so rather than implying otherwise. Finally, layer 47 proved **dead weight for this role**: a 2-channel 24+36 tap equalled or beat the 3-channel tap and was slightly better on real negatives, so the locked content selector is `CoreContent_v2_blockwise_pruned_24_36`. At branch-selection time the realistic candidate pool resembles the wrong-problem regime more than the mutant regime, so the mutant-heavy figures should be read as an upper bound on deployed selection quality.

Non-looped control (Appendix E.2.2b). The identical CoreContent protocol on MiniCPM-2B-sft-bf16 (40 distinct blocks, no loops; revision `4ec16344`…, weight SHA-256 `0b0c993a`…; layers 24/36, mask-valid mean pooling, task-disjoint split, 0 crossing IDs) gives held-out macro top-1 **0.5680** and macro pairwise **0.7237** (per-domain top-1: coding 0.9121, alignment 0.6927, reasoning 0.5204, math 0.3893, logic 0.3255). Establishes: readable candidate-quality structure does not require a looped architecture. Does not establish: any causal attribution of the Ouro-vs-control gap to looping (the models differ in family, width, corpus, objective, and tap geometry). See §4.6.

E.2.3 S3B2 (generated-branch correctness refit). L2 logistic AUROC **0.7515** / pairwise **0.6835**; expanded hidden-ridge AUROC **0.7755** / pairwise **0.7338**; metadata-only controls weak; high-margin abstention non-rescuing. Selection: **5/8 = 0.625** on N = 8 oracle-present task groups (full pool 160 candidates / 16 groups). The **0.5833** figure sometimes cited as the matched baseline is *not* a matched control — it is the S3B1 three-domain macro $(0.5+0.75+0.5)/3$; a pool-weighted matched-random over the same eight groups (correct counts 7,1,3,2,4,2,8,2 out of ten candidates each) gives matched random **0.3625**, with exact Poisson-binomial $P(\geq 5) = 0.087$ — so the selector exceeds the random point expectation but N=8 is underpowered and reliable selection is not established (§6.4). Supporting: `artifacts/reports/paper_verification/pending_items_resolution_20260703_214531.*`, `dualanchor-architecture-baseline.md`, `dualanchor-tap-evolution.md`, `content-selection-taps.md`, `corecontent-dataset-expansion-v2.md`, `core-domain-tap-audit.md`, `terminal-selection-and-arbiters.md`, `current-state.md`.

E.3 Terminal selection and the final arbiter

Terminal selection — choosing one final output from the survivor set — was the persistent weak point, and several arbiter designs were tried before the program settled on defer rather than a solved selector. The lineage and its verdicts:

Stage	Status	Interpretation
selection-only prototype v1	`SURVIVAL_READY_FINAL_ARBITER_WEAK`	good branches survived; final selection lagged best survivor
final arbiter v1 (`listwise_softmax`)	`FINAL_ARBITER_WEAK_BUT_USEFUL`	beat majority and fixed-top-1, missed the 0.75 target
final arbiter v1.1 (`tie_aware_rank_listwise`)	`NO_IMPROVEMENT`	rank-heavy / tie-aware training did not clear readiness
merged weight taps v1	`FINAL_ARBITER_IMPROVES_ONLY`	weight-space merged tap helped the final choice but did not replace the selector
merged tap integration v1.1	`DOMAIN_FALLBACK_USEFUL_BUT_REASONING_LIMITED`	domain-gated fallback improved CV macro; reasoning remained the blocker

Stage	Status	Interpretation
DualAnchor architecture-looped v3	READY_WITH_TERMINAL_DEFER	survival ready; terminal confidence weak on the hard slice

Status of the quantitative figures. The arbiter lineage's *verdicts* stand — every terminal arbiter closed weak or NO_IMPROVEMENT, and the program's locked policy is confidence-gated top-1 with defer. Its *numbers* do not. The selection-only prototype figures (top-4 retention 0.9514, best-selected reward 0.9453, final reward 0.6672), the architecture-looped terminal figures (forced top-1 oracle 0.9167, reward 0.2625 against best 0.3167, reward-diverse 0.6364), and the integrated comparison (reported as CoreContent pairwise 0.6584 against DualAnchor forced top-1 0.3787) were all measured on splits with task IDs crossing the train/held-out boundary, and the integrated comparison additionally mislabeled a macro top-1 as a pairwise accuracy and used CoreContent candidate groups rather than real branch-survivor pools. All are **withdrawn**. A zero-crossing re-run (2026-07-13) verifies survival (stage retention 0.9697, terminal 1.0000, scaffold top-4 1.0000) but leaves only 2 reward-diverse tasks of 9 — too few to quantify terminal selection in either direction. On that remainder, forced top-1 oracle retention is 0.8889, forced top-1 reward ≈0.0000, best terminal reward 0.0222; on the clean actual-survivor subset CoreContent top-1 is 0.7778 against DualAnchor forced top-1 0.8889, a descriptive reversal that is itself underpowered. We report no terminal-selection magnitude.

The confidence work concluded TERMINAL_WEAK_ON_HARD_SLICE / TERMINAL_CONFIDENCE_ONLY: aggregate terminal top-1 looks strong only because many tasks are tie-heavy; on reward-diverse and positive-plus-reward-diverse slices, unconditional top-1 is not reliable. The locked rule at terminal L4_47 is therefore: (1) score terminal candidates pairwise with both DualAnchor taps; (2) evaluate forced top-1 as diagnostic only; (3) collapse only if the confidence gate fires; (4) otherwise keep or defer the terminal survivors. No arbiter in this line is a steering module. The integrated terminal test that had been reported here (CoreContent-v2 0.6584 against DualAnchor forced-top-1 0.3787) is withdrawn for the reasons above and did not replicate on clean survivors. The composed pipeline (DualAnchor survival → CoreContent ranking → survivor handoff) remains the system's architecture; what is no longer claimed is a quantified advantage for either component at the terminal stage. Supporting: `terminal-selection-and-arbiters.md`, `history/bg-run-notes/terminal-arbiters/*`, `history/bg-run-notes/survival-selection/bg_selection_only_phase2_prototype_v1.md`, `branch-training-logic-expansion.md`, `current-state.md`.

Appendix F — Branch/carry/prune and KV-cache implementation

Cache structure. `OuroForCausalLM`, bf16, eager attention; `total_ut_steps = 4`, `num_hidden_layers = 48` → **192 distinct cache slots**, with `cache_slot = current_ut * num_hidden_layers + layer_idx` (slot audit: SLOT_MAPPING_CONFIRMED). Each (loop, layer) owns a distinct KV slot; prefill populates all 192; a decode step appends one token to every slot; `reorder_cache` permutes the batch dimension of every populated slot.

Why this is not prompt-only carry. Prompt-only layer carry (used by the offline probes, which run `use_cache=False`) is a strictly easier problem. Generation-time branch-specific carry requires each branch to maintain its own `past_key_values`, `cache_position`, `attention_mask`, `position_ids`, `generated_ids`, and lineage, aligned across every autoregressive decode step.

Equivalence standard. Prefill is **bit-exact** (RMS 0). Cached decode shows small bf16 drift (RMS ≈ 0.05–0.2, max-abs < 1.0) from cached-versus-recomputed key/value paths; top-1 token sequences match except at model-intrinsic argmax near-ties. The splice (below) is bit-exact end-to-end.

v1 validation ladder (`autoregressive_kv_branch_carry_v1`, 2026-06-01; status PROMPT_INTERNAL_BRANCH_CACHE_VALID):

Level	Result
L0 cached decode	matches full recompute (prefill bit-exact; decode within bf16 drift)
L1 token-boundary fork	K = 2/4/8 independent branch caches; no cross-branch contamination
L2 batched branches	batched ≡ independent ≡ full recompute
L3 prune / reorder survivors	8→4→2, 8→3, 4→1; lineage aligned
L4 current-token perturb	layers 24/36/47, loop-targeted; carries via branch cache
L5 prompt-internal perturb	branch-specific cache required; **negative control** RMS ≈ 3.0 without it
L6 partial splice (v1)	slot-boundary logic valid but **diagnostic only — no compute saving**

Supporting: PADDING_MASK_SAFE (left-padded batched decode requires explicit per-row `position_ids`), DualAnchor integration smoke BRANCH_PRUNE_CARRY_SMOKE_VALID, failure analysis NO_MAJOR_FAILURES.

v2 suffix-recompute splice (`partial_cache_splice_v2`, 2026-06-04; status `PARTIAL_SPLICE_COMPUTE_SAVING_VALID`, upgrading v1's L6 from diagnostic-only):

- *Obstacle.* The KV cache stores K/V but **not** the inter-layer residual stream, so a perturbed branch cache naively forces a full re-prefill.
- *Solution.* During a minimal shared-prefix prefill, additionally capture the residual hidden at the perturbation boundary (output of loop u, layer ℓ). For an additive boundary perturbation, reconstruct $H^{\text{perturbed}}_{\text{boundary}} = H_{\text{boundary}} + \delta$ with **no forward pass**, and recompute only the suffix (loop u, layers $\ell + 1..$; then loops $u + 1..$). Implemented as test-only orchestration over `model.model.layers / rotary_emb / norm / lm_head` — no weight edits, no permanent model surgery.
- *Hook timing (empirical).* Perturbing a layer *output* leaves that layer's boundary slot unaffected; the first affected slot is $(u,\ \ell+1)$, and the changed set equals the `downstream_only` prediction. Over-sharing an affected slot diverges (negative control).
- *Equivalence.* The spliced branch cache is **bit-exact** versus a full perturbed-prompt reference: all 192 slots, prefill logits RMS 0, continuation bit-for-bit. Validated single-branch, multi-branch (K = 2/4, independent storage, no contamination), batched + prune/reorder, and left-padded.
- *Measured compute saving* (baseline = K full perturbed prefills; splice = one shared prefill + K suffix recomputes), per-branch layer passes saved: loop 0 **13%**, loop 1 **38%**, loop 2 **63%**, loop 3 **88%**. K-scaling at (loop 2, layer 24): K=2 **32%**, K=4 **47%**, K=8 **55%**. Amortized over $K \geq 2$.

What cannot be claimed (recorded in the source notes and honored here): production readiness; steering of any kind; any claim beyond the validated levels. The substrate is a correctness result, not a capability result.

Supporting: `kv-cache-branch-carry.md`, `bg_autoregressive_kv_branch_carry_v1.md`, `autoregressive_kv_branch_carry_validation_note.md`, `bg_partial_cache_splice_v2.md`, `branch-generation-and-survival.md`, `phase1-controller-and-routing.md`.

Appendix G — S1 frozen-fork / K-matched sampling deconfound

Protocol. A branch is forked at a chosen loop/layer boundary, carried forward through its own KV cache (Appendix F), and allowed to generate an independent continuation. The question is whether the *injection* adds reachability — i.e. reaches correct answers the unforked baseline does not — or whether any apparent gain is attributable to the sampling randomness the fork happens to introduce.

Deconfound. The comparison is against **K-matched plain sampling**: drawing the same number of ordinary temperature samples and taking the oracle over them. If matched sampling reaches an equal or higher oracle, the fork adds nothing.

Parameter	Value
Tasks	4
Plain samples per task	12
Temperature	0.7
Top-p	0.95
Sampling budget	96 tokens (long reference: 160)

Results.

Condition	Oracle
Plain sampling (K-matched)	**0.750**
Sampled fork	**0.611**
Fork − sampling	**−0.139**
Greedy (deterministic) fork, new-correct answers	**0**

The sampled fork lands *below* matched sampling, and deterministic forking produces no new correct answers at all. Verdict: `FROZEN_FORK_CLOSED__SAMPLING_EXPLAINS_SCREEN__S3_IS_LEVER` — frozen branch injection adds no demonstrated reachability beyond matched stochastic decoding in this bounded test.

Scope. Four tasks is a small screen; the load-bearing claim is the *direction* of the deconfound (fork does not exceed matched sampling, greedy fork adds nothing), not the precise decimal. Supporting: S1 frozen-fork artifacts (2026-06-17), `artifacts/reports/paper_verification/pending_items_resolution_20260703_214531.*`, `current-state.md`.

Appendix H — Steering / adapters closure

Validated write surface. Before any steering claim: decoder-layer hooks are mechanically clean. Zero-magnitude writes reproduce no-hook generation exactly; perturbation size scales predictably; perturbations propagate to later hidden states and to logits (surviving ~32 tokens); no CUDA/NaN/Inf instability inside the safe envelope. Safe-magnitude envelope: $\alpha \leq 0.02$ effective RMS.

Seven methods, all unsigned.

Method	Verdict
Raw readout direction (NoNorm)	`UNSIGNED_EFFECT`
Empirical success-mean difference	`EMPIRICAL_UNSIGNED_ONLY`
RMS-calibrated static direction	`RMS_UNSIGNED_ONLY`
Local outcome-score gradient probe	`GRADIENT_NO_BETTER_THAN_RANDOM`
Classifier-derived adapter	no reliable held-out control
Teacher-forced causal adapter	`ADAPTER_IMPROVES_LOGIT_MARGIN` (teacher-forced) → `TEACHER_FORCED_ONLY` → `LOCAL_LOGIT_CONTROL_ONLY`
Sequence-level (REINFORCE) adapter	`SEQUENCE_REWARD_IMPROVES` (training) → `NO_ADAPTER_SPECIFIC_TRANSFER` (held-out) → **`WORSE_THAN_RANDOM`**

Summary verdicts: `BG_SEQUENCE_LEVEL_ADAPTER_VERDICT = NO_FROZEN_BACKBONE_WRITE_PATH`; `FROZEN_BACKBONE_INFERENCE_STEERING_STATUS = CLOSED_UNDER_TESTED_METHODS`.

Geometry. Readout geometry ≠ empirical-success geometry ≠ local logit-control geometry:

Direction pair	Cosine
Adapter proxy vs. raw NoNorm readout	−0.0005530
Adapter proxy vs. empirical success-mean difference	−0.0042941
Raw NoNorm readout vs. empirical success-mean difference	+0.1010016
Sequence-level adapter vs. teacher-forced adapter (independent training runs)	**+0.9510944**

The two independently trained adapters converge on nearly the same direction while both remaining nearly orthogonal to the readout and empirical-success directions — consistent with both descending into the same non-steering local-control basin rather than discovering the outcome direction.

Scope. The closure holds under the tested safe-α envelope and tested optimizers. It does not prove that no training method can steer Ouro; it closes the simple frozen-backbone inference-time path and motivates the training-time route of §12. What was explicitly *not* established: reliable action steering, a production write path, or a trained steering corridor.

Supporting: `steering-and-adapters.md`, `artifacts/reports/probes/bg_steering_suite_2026-05-18/summary.md`, `artifacts/reports/probes/bg_steering_suite_2026-05-18/analysis.md`, `bg_causal_intervention_adapter_2026-05-18`, `current-state.md`.

Appendix I — Strict pre-answer audit details

Data. GSM8K (Cobbe et al., 2021): **170 tasks**, 4 sampled solutions each, **680 examples**; 407 positive / 273 negative by external checker. Raw per-example features, predictions, and labels are preserved at `artifacts/reports/proto_introspection/within_domain_recapture.pt`, so no regeneration is required to re-audit the cut.

The strict pre-answer cut. Enforced in the extraction code, not by post-hoc filtering. Excluded by construction:

1. **Answer region** — all tokens from the point the solution begins committing its final numeric answer onward. Loop states over this span never enter the feature vector.
2. **Gold value** — the reference answer is not present in the extraction path in any form. The probe has no channel through which to match against it.
3. **Correctness label** — produced by an external checker after generation; used solely as the probe's training target, never as an input feature.

The probe therefore sees only the loop trajectory of a computation that has not yet produced an answer.

Features and controls. Hidden features are pooled loop states over the pre-answer span. Two shortcut controls are computed on the same span: solution **length** and token **log-probability**. The composite control is length + logprob.

Results. Per-feature AUROC: hidden 0.745 [0.707, 0.783]; length 0.687; logprob 0.569; length + logprob **0.731**; hidden + all **0.797**. Incremental gain **+0.066** (0.065790 exactly).

Interval estimation. Because the 680 examples are nested within 170 tasks, an i.i.d. bootstrap over examples is anti-conservative. The reported interval is a **paired task-clustered bootstrap**: each of 10,000 draws (seed 20260710) resamples 170 task IDs with replacement, retains all four candidates per sampled task, computes both AUROCs on that draw, and records their difference. Result: 95% percentile CI **[+0.021, +0.112]**, bootstrap mean +0.0658, SD 0.0235, zero one-class draws — **excludes zero**. A candidate-level bootstrap gives the narrower [+0.032, +0.100] and is reported only as a diagnostic. Leave-one-task-out re-estimation moves the increment within [+0.056, +0.071] (max change 0.0098), so no single task drives the result.

Provenance note. The original June interval ([+0.017, +0.114]) was described in its report as a task/group bootstrap, but the paired clustered-delta routine was not preserved and the original draws were not saved, so its execution path is not code-auditable. The interval above was recomputed from the preserved raw predictions and supersedes it. Supporting: `artifacts/reports/paper_verification/fable_hardening_checks_20260710_175017/preanswer_task_clustered_ci.json`.

Appendix J — Orthogonality / null audit

The hypothesis under test. If the frozen branch mechanism can only write into a subspace U_{inj}, and the verifier-outcome direction d_{out} lies largely outside it, the frozen-control pattern would have a simple geometric explanation. The rank-corrected audit does not support that one-dimensional span-misalignment account; it enters the paper as a tested but unsupported explanation, never as proof of either that account or its converse.

Setup. Exact-protocol regenerated S1/S3 injection/carry deltas (regenerated from the protocol, not replayed from historical tensors — a caveat we preserve). Ambient dimension $D = 24{,}576$; injection-span rank $k = 344$; participation ratio ≈ 5.06, so the span is effectively far lower-dimensional than its nominal rank.

Quantity	Value
Observed projection $\pi_{\text{inj}}(d_{\text{out}})$	**0.018296**
Random-direction baseline k/D	0.013997
Ratio observed / random	**1.307**×
Percentile vs. random-direction null (10^6 draws)	99.9907 ($p_{\text{right}} = 9.3 \times 10^{-5}$)
Shuffled-label null, global mean (10^4 draws)	0.022990 (observed at percentile 3.44)
Shuffled-label null, domain-stratified mean	0.022702 (observed at percentile 0.0)

Why the two nulls disagree. Against *random directions*, the outcome direction projects into the injection span **more** than chance (1.31×, 99.99th percentile) — so the naive reading, "the outcome direction lies unusually outside the writable span," is not supported by this null. Against *outcome-shaped shuffled-label* directions (same correlational structure, no true verifier information), it projects **less** than their mean — pointing weakly the opposite way. The gap between observed and shuffled-mean is ≈ 0.0044 of total energy: small.

Verdict. Ambiguous and fragile. The estimate rests on a single one-dimensional projection from one regenerated delta bundle, and the two nulls point in opposite directions. A separate control confirms the injection span is statistically distinct from the natural sampling span (frozen forking is not merely resampling), but that answers a different question. We therefore do **not** treat subspace misalignment as an explanation of the frozen null. The boundary remains empirical and its mechanism unresolved (§8.4). A proper subspace-vs-subspace analysis (principal angles / CCA) rather than a 1-D projection is listed as future work (§12.4).

Supporting: `s1_s3_exact_injection_orthogonality_null_audit_2026-06-17.*`, helper `tools/s1_s3_exact_injection_null_audit_2026_06_17.py`.

Appendix K — Artifact and reproducibility index

Repository state. Reproducibility is pinned to commit `e4776dd41a85cad699ac36f309b5986ab48bd171`. Canonical project state: `current-state.md`.

Models.

Role	Identifier	Revision / hash
Base	`ByteDance/Ouro-2.6B`	`1ed04250da1a9936042725d302e81c8fa2ab5abd`
Reasoning-SFT	`ByteDance/Ouro-2.6B-Thinking`	`f1edd81e7ac41355db670500ceaf204e0f73af68`
RL-trained (primary)	Ouro-RLTT research weights provided by Jonathan Williams; locally converted checkpoint at `.../models/ouro_rltt_local`	local shard hashes recorded; upstream revision unavailable

Role	Identifier	Revision / hash
Non-looped control (§4.6)	`openbmb/MiniCPM-2B-sft-bf16`	rev `4ec16344ac13e6ef5010aeecaa533369ac8eb53c`; weights SHA-256 `0b0c993ace78c5983373948c636b0e587fcf1ac6f2e0f980bf7d735fe7dc52f8`
Evaluator checkpoint	`artifacts/checkpoints/evaluator/pairwise_epoch2.pt`	SHA-256 `3630c2092eca8db13239f763bc9c212f4b673866e47f811c3095efc57409ec96`

Environment. `transformers==4.54.1` (pinned); PyTorch `2.12.0.dev20260407+cu128`; bf16 forward, fp32 features, eager attention; no quantization. Hardware: NVIDIA RTX 5070 Ti Laptop GPU.

Seeds. Pair selection / split: `20260711`. Task-clustered bootstrap (§5): `20260710`. Domain-transfer task-disjoint split (§4.4): `20260711`. Linear-probe reconstruction: `42`. Bootstraps use 10,000 draws throughout.

Primary artifacts by result.

Result	Artifact
Strict pre-answer GSM8K (§5)	`artifacts/reports/proto_introspection/within_domain_recapture.pt` (raw features, predictions, labels); clustered CI in `.../fable_hardening_checks_20260710_175017/preanswer_task_clustered_ci.json`
Full 8,552-pair antisymmetry audit (§3.3)	`.../remaining_todos_resolution_20260710_183700/full_hh_audit/full_hh_antisymmetry.{json,csv}`
Powered pair-disjoint probes (§3.2, §3.5)	`.../powered_clean_probe_and_corecontent_20260711/hh_{base,thinking,rltt}_40000_pair_disjoint_results.json`; paired bootstrap `powered_hh_paired_bootstrap.json`; pointwise `hh_thinking_40000_pointwise_pair_disjoint_results.json`
CoreContent task-disjoint refit (§4.5)	`.../powered_clean_probe_and_corecontent_20260711/ouro_corecontent_task_disjoint_results.json`
Non-looped control (§4.6)	`.../nonlooped_architecture_control_20260710_235252/` (report + `corecontent_control_results_task_disjoint.json`)
Domain-transfer re-run (§4.4)	`artifacts/reports/probes/paper_v44_task_disjoint_rerun_2026-07-13/tier2_domain_transfer.json`
Branch survival re-run (§6.2)	`.../paper_v44_task_disjoint_rerun_2026-07-13/tier1_branch_survival_selection.json`; integrated survivors `integrated_true_survivors.json`
S3B2 detection/selection (§6.3–6.4)	`.../final_engineering_expansion_2026-06-17/s3b2_generated_branch_correctness_expanded_2026-06-17.md`
KV-cache / splice (§7)	`kv-cache-branch-carry.md`; `bg_autoregressive_kv_branch_carry_v1.md`; `bg_partial_cache_splice_v2.md`
Steering closure (§8.1)	`artifacts/reports/probes/bg_steering_suite_2026-05-18/{summary,analysis}.md`; `bg_causal_intervention_adapter_2026-05-18`
Orthogonality null audit (§8.4)	`s1_s3_exact_injection_orthogonality_null_audit_2026-06-17.*`

Verification tooling. `tools/paper_verification/` — including `powered_hh_pair_disjoint_probe.py`, `powered_hh_pointwise_pair_disjoint_probe.py`, `paired_bootstrap_powered_hh.py`, `rerun_ouro_corecontent_task_disjoint.py`, `nonlooped_transformer_control.py`, and `utilities/tests/manual/rerun_paper_v44_task_disjoint_audits.py`.

Known provenance gaps (stated rather than hidden): the original ~84.5% linear-probe weight vector was not preserved (the cross-backbone replication of §3.5 uses a reconstruction); the historical base-24% localization run has no surviving artifact (§3.5, retracted); the original June pre-answer CI's bootstrap code and draws were not preserved, so the interval reported here is a fresh recomputation from the raw predictions (§5.2, Appendix I); the math-transfer origin figures (§3.6) are unarchived and non-load-bearing; and the base-backbone extraction in the powered probe run completed without writing its shard manifest, though shard and row counts are independently verified.

References

Arditi, Andy, Oscar Obeso, Aaquib Syed, et al. 2024. *Refusal in Language Models Is Mediated by a Single Direction*. https://arxiv.org/abs/2406.11717.

Bai, Yuntao, Andy Jones, Kamal Ndousse, et al. 2022. *Training a Helpful and Harmless Assistant with Reinforcement Learning from Human Feedback*. https://arxiv.org/abs/2204.05862.

Betley, Jan, Xuchan Bao, Martín Soto, Anna Sztyber-Betley, James Chua, and Owain Evans. 2025. *Tell Me about Yourself: LLMs Are Aware of Their Learned Behaviors*. https://arxiv.org/abs/2501.11120.

Binder, Felix J., James Chua, Tomek Korbak, et al. 2024. *Looking Inward: Language Models Can Learn about Themselves by Introspection*. https://arxiv.org/abs/2410.13787.

Cobbe, Karl, Vineet Kosaraju, Mohammad Bavarian, et al. 2021. *Training Verifiers to Solve Math Word Problems*. https://arxiv.org/abs/2110.14168.

Comşa, Iulia, and Murray Shanahan. 2025. *Does It Make Sense to Speak of Introspection in Large Language Models?* https://arxiv.org/abs/2506.05068.

Fu, Tingchen, Yupeng Hou, Julian McAuley, and Rui Yan. 2024. *Unlocking Decoding-Time Controllability: Gradient-Free Multi-Objective Alignment with Contrastive Prompts*. https://arxiv.org/abs/2408.05094.

Geiping, Jonas, Sean McLeish, Neel Jain, et al. 2025. *Scaling up Test-Time Compute with Latent Reasoning: A Recurrent Depth Approach*. https://arxiv.org/abs/2502.05171.

Hao, Shibo, Sainbayar Sukhbaatar, DiJia Su, et al. 2024. *Training Large Language Models to Reason in a Continuous Latent Space*. https://arxiv.org/abs/2412.06769.

Hendrycks, Dan, Collin Burns, Saurav Kadavath, et al. 2021. "Measuring Mathematical Problem Solving with the MATH Dataset." *Advances in Neural Information Processing Systems: Datasets and Benchmarks*. https://arxiv.org/abs/2103.03874.

Kirin, Jan. 2026a. *Relational Preference Encoding in Looped Transformer Internal States*. https://arxiv.org/abs/2604.09870.

Kirin, Jan. 2026b. *Two Evaluation Traps in Constructed-Row Pipelines: Source-Item Leakage, Presentation-Order Shortcuts, and the Audit Protocol That Catches Them*.

Korbak, Tomek, Mikita Balesni, Elizabeth Barnes, et al. 2025. *Chain of Thought Monitorability: A New and Fragile Opportunity for AI Safety*. https://arxiv.org/abs/2507.11473.

Kutasov, Jon, Chloe Loughridge, Yuqi Sun, et al. 2025. *Evaluating Control Protocols for Untrusted AI Agents*. https://arxiv.org/abs/2511.02997.

Kwon, Woosuk, Zhuohan Li, Siyuan Zhuang, et al. 2023. "Efficient Memory Management for Large Language Model Serving with PagedAttention." *Proceedings of the 29th Symposium on Operating Systems Principles*. https://arxiv.org/abs/2309.06180.

Lambert, Nathan, Valentina Pyatkin, Jacob Morrison, et al. 2024. *RewardBench: Evaluating Reward Models for Language Modeling*. https://arxiv.org/abs/2403.13787.

Li, Ji-An, Hua-Dong Xiong, Robert C. Wilson, Marcelo G. Mattar, and Marcus K. Benna. 2025. *Language Models Are Capable of Metacognitive Monitoring and Control of Their Internal Activations*. https://arxiv.org/abs/2505.13763.

Lightman, Hunter, Vineet Kosaraju, Yura Burda, et al. 2023. *Let's Verify Step by Step*. https://arxiv.org/abs/2305.20050.

Lindsey, Jack. 2026. *Emergent Introspective Awareness in Large Language Models*. https://arxiv.org/abs/2601.01828.

Macar, Uzay, Li Yang, Atticus Wang, Peter Wallich, Emmanuel Ameisen, and Jack Lindsey. 2026. *Mechanisms of Introspective Awareness*. https://arxiv.org/abs/2603.21396.

Marks, Samuel, and Max Tegmark. 2023. *The Geometry of Truth: Emergent Linear Structure in Large Language Model Representations of True/False Datasets*. https://arxiv.org/abs/2310.06824.

Miao, Xupeng, Gabriele Oliaro, Zhihao Zhang, et al. 2024. "SpecInfer: Accelerating Generative LLM Serving with Tree-Based Speculative Inference and Verification." *Proceedings of the 29th ACM International Conference on Architectural Support for Programming Languages and Operating Systems*. https://arxiv.org/abs/2305.09781.

Pearson-Vogel, Theia, Martin Vanek, Raymond Douglas, and Jan Kulveit. 2026. *Latent Introspection: Models Can Detect Prior Concept Injections*. https://arxiv.org/abs/2602.20031.

Saunshi, Nikunj, Nishanth Dikkala, Zhiyuan Li, Sanjiv Kumar, and Sashank J. Reddi. 2025. *Reasoning with Latent Thoughts: On the Power of Looped Transformers*. https://arxiv.org/abs/2502.17416.

Song, Siyuan, Harvey Lederman, Jennifer Hu, and Kyle Mahowald. 2025. *Privileged Self-Access Matters for Introspection in AI*. https://arxiv.org/abs/2508.14802.

Williams, Jonathan, and Esin Tureci. 2026. *Prioritize the Process, Not Just the Outcome: Rewarding Latent Thought Trajectories Improves Reasoning in Looped Language Models*. https://arxiv.org/abs/2602.10520.

Yao, Shunyu, Dian Yu, Jeffrey Zhao, et al. 2023. "Tree of Thoughts: Deliberate Problem Solving with Large Language Models." *Advances in Neural Information Processing Systems*. https://arxiv.org/abs/2305.10601.

Zheng, Lianmin, Liangsheng Yin, Zhiqiang Xie, Chuyue Sun, et al. 2024. *SGLang: Efficient Execution of Structured Language Model Programs*. https://arxiv.org/abs/2312.07104.

Zhu, Rui-Jie, Zixuan Wang, Kai Hua, et al. 2025. *Scaling Latent Reasoning via Looped Language Models*. Project page: http://ouro-llm.github.io. https://arxiv.org/abs/2510.25741.

Zou, Andy, Long Phan, Sarah Chen, et al. 2023. *Representation Engineering: A Top-down Approach to AI Transparency*. https://arxiv.org/abs/2310.01405.