Decodable but Not Faithful: Coupling Natural-Language Rationales to Programmatic Verifiers

Vatsal Ananthula [1] **Adarsh Kumarappan** [2]

Abstract

Language models can generate plausible rationales for their predictions, but these explanations may not faithfully represent the model's internal reasoning. We propose *verifier-coupled reasoning*, a framework that inserts inline claims into reasoning traces and trains an auxiliary consistency head to predict programmatic verifier outputs from rationale-span hidden states. The central finding is a gap between *decodability* and *faithfulness*: consistency training reliably makes verifier information decodable from rationale representations, but decodability does not guarantee faithful generation. In LeanCheck (formal theorem proving), rationale-only and proof-only pooling achieve perfect directional separation under counterfactual conflict. In KataGo (Go engine), commentary spans encode 10-way winrate buckets at 81% accuracy. Yet in a code setting, the model achieves 98.6% coupling while its generated explanations remain unfaithful: fluent prose with correct structured claims, but describing unrelated algorithms; a controlled pretrained-vs-from-scratch comparison shows the gap is not capacity-driven. Synthetic activation patching confirms causal influence (73–89% vs. 31% baseline), FEVER reveals that evidence-only pooling isolates genuine evidence sensitivity at the cost of raw accuracy, and per-claim analysis shows that consistency loss disproportionately benefits fine-grained claims over binary ones. These results establish that consistency losses are effective diagnostics and representation-shaping tools, but not sufficient conditions for faithful reasoning.

[1]University of California, Berkeley [2]California Institute of Technology. Correspondence to: Vatsal Ananthula <vananthu@berkeley.edu>, Adarsh Kumarappan <adarsh@caltech.edu>.

3rd AI for Math Workshop at the 43rd International Conference on Machine Learning, Seoul, South Korea, 2026.
Code and data: https://github.com/vatsalananthula/verifiedcoupling

1 Introduction

Chain-of-thought and rationale generation have become default tools for making language-model behavior more inspectable, especially on mathematical and scientific reasoning tasks. The appeal is straightforward: if a model exposes intermediate reasoning, a user or verifier can inspect the path rather than only the final answer. Prior work has shown that chain-of-thought can improve task accuracy (Wei et al., 2022), and self-consistency can further improve reasoning by aggregating sampled reasoning paths (Wang et al., 2023b). Yet this transparency is not perfect. A model can write a plausible explanation that rationalizes a decision after the fact, omits the true source of an answer, or uses shortcuts invisible in the generated text. Generated explanations may fail to faithfully represent the model's internal decision process (Turpin et al., 2023), and probing work cautions that decodable information is not automatically evidence of causal use (Hewitt & Liang, 2019).

External verifiers offer an opportunity to address these issues. In mathematical reasoning, theorem provers and symbolic systems can check claims. In factual verification, evidence labels can serve as targets. In games and scientific domains, engines or simulators can compute dense state evaluations. Process supervision shows that step-level feedback can improve mathematical reasoning beyond outcome-only supervision (Lightman et al., 2023), and recent RL with verifiable rewards uses automatically checkable outcomes to train stronger reasoning policies (DeepSeek-AI, 2025). The usual use of these tools is outcome-level: check the final answer, rerank candidates, or train a reward model. That leaves a gap between *verified claims* and *natural-language rationales*. The model may produce a correct claim while the explanation is unsupported, or encode verifier information internally while failing to verbalize it coherently.

This paper studies a mechanism for narrowing this gap by making the rationale representation directly accountable to the verifier. Rather than only checking the final claim, we train the hidden states of the rationale span to predict the verifier's output. This creates a measurable intermediate condition: *decodability*, meaning that verifier information can be decoded from rationale states. Decodability is a necessary condition for faithfulness—a rationale whose hidden

states carry no verifier-relevant information cannot faithfully explain the claim—but the critical question is whether it is sufficient. Does a model whose representations encode verifier information also *generate* explanations that faithfully reflect it?

Concretely, the model emits a sequence containing a natural-language rationale and an explicit inline claim. A programmatic verifier supplies a target label or scalar. An auxiliary consistency head—a linear classifier over mean-pooled rationale hidden states—predicts that target. The resulting consistency loss is added to the standard language-modeling objective. We then escalate diagnostics from decodability (consistency-head accuracy) through counterfactual following (do swapped spans change predictions?) to causal influence (does patching rationale activations shift claim logits?) and generation quality (does the model produce usable explanations?).

The central finding is that decodability and faithfulness come apart. In controlled and formal-verifier settings, consistency training makes verifier information reliably decodable and even causally influential under activation patching. But in a code-explanation setting, the model achieves 98.6% representation coupling while its generated prose, though fluent, describes unrelated algorithms; a controlled comparison on a pretrained GPT-2 reproduces the same fluent-but-unfaithful failure mode, showing the gap is structural rather than a capacity artifact. This separation of representation and generation is the paper's main result, and it has implications for any approach that uses probe accuracy as evidence of faithful reasoning.

We test across five settings: synthetic (controlled validation and causal intervention), LeanCheck (Lean theorem prover), KataGo (Go engine), code coupling (where the failure mode is most visible), and FEVER (real-language evidence sensitivity). Our contributions are:

- **Verifier-coupled reasoning**: a sequence format and span-level consistency loss for coupling natural-language rationales to programmatic verifier claims, applicable to theorem provers, game engines, code oracles, and evidence labels.
- **A diagnostic ladder from decodability to faithfulness**: consistency-head accuracy, counterfactual span swaps, activation patching, per-claim analysis, and generation-quality evaluation, each level revealing information the previous level misses.
- **A characterization of the decodability–faithfulness gap**: across five settings, representation coupling reaches near-perfect levels while generated explanations can remain unusable, establishing that SFT-style consistency losses are effective diagnostics but not sufficient for faithful reasoning.

2 Related Work

Rationales and chain-of-thought faithfulness. Chain-of-thought prompting demonstrates that intermediate reasoning can improve language-model performance on arithmetic, symbolic, and commonsense tasks (Wei et al., 2022), and self-consistency improves reasoning by marginalizing over sampled reasoning paths (Wang et al., 2023b). However, generated rationales need not reveal the causal basis of a model's answer. Models can rationalize biased answers without mentioning the biasing feature (Turpin et al., 2023), and faithfulness can remain limited even after in-context learning, fine-tuning, or activation editing (Tanneru et al., 2024). Broader analyses of explanation faithfulness emphasize that plausible natural language should not be equated with faithful computation (Jacovi & Goldberg, 2020; Lanham et al., 2023).

Process supervision and verifier-guided generation. Process supervision trains systems to evaluate or supervise intermediate reasoning steps rather than only final answers. Step-level supervision can outperform outcome supervision on mathematical reasoning benchmarks (Lightman et al., 2023), and automated process-supervision methods reduce dependence on human labels through search, rollout, or code verification (Uesato et al., 2022; Cobbe et al., 2021; Wang et al., 2023a; Luo et al., 2024). Verifier-guided reasoning also appears in code and scientific settings, where execution or simulation can provide feedback (Chen et al., 2021; Dong et al., 2024). Our work differs in where the pressure is applied: rather than only scoring completed steps, we train the hidden representation of the natural-language rationale span itself to encode the verifier-derived claim. Lean 4 provides a natural formal-verifier target because it combines an interactive theorem prover with an extensible programming language (de Moura & Ullrich, 2021).

RLHF, RLVR, and verifiable rewards. RLHF aligns models to human preferences by training reward models and optimizing policies (Christiano et al., 2017; Ouyang et al., 2022). More recent reasoning systems increasingly use verifiable rewards from math, code, or structured environments (DeepSeek-AI, 2025; Shao et al., 2024; Lambert et al., 2024). Verifier-coupled reasoning can be viewed as a supervised precursor to RL with verifiable rewards: it creates inline claims that can later be checked and rewarded automatically.

Probing, causal interventions, and activation patching. Our work is most closely related to probing and causal representation evaluation. Probing warns that high probe accuracy may reflect probe capacity rather than genuine encoded structure, motivating controls and selectivity metrics (Hewitt & Liang, 2019; Pimentel et al., 2020). Causal tracing and activation patching provide stronger tests by asking whether replacing or ablating representations changes model outputs

(Meng et al., 2022; Zhang & Nanda, 2023). Truth-probing work shows that factual properties can sometimes be linearly represented and manipulated in activations (Marks & Tegmark, 2023). Our work differs from standard probing in two ways. First, we do not merely *decode* information from pre-existing representations; we *train* the representation to encode verifier-derived targets via the consistency loss, then test whether this trained coupling transfers to generation behavior. Second, we systematically escalate diagnostics: from decodability (consistency-head accuracy) through counterfactual following (do swapped spans change the head's prediction?) to causal influence (does patching rationale activations shift claim logits?) to generation quality (does the model produce coherent explanations?). The code experiment demonstrates why this escalation matters: decodability reaches 98.6% while generation quality is zero, a gap that standard probing alone would not reveal.

3 Background

Programmatic verifiers. A programmatic verifier is any external procedure that can compute a target label, scalar, or accept/reject judgment for a model-produced claim. In this paper the verifier is not a human preference model; it is a checkable computation such as a theorem prover, a game engine, a factual-evidence label, or code execution. This matters because the verifier target can be attached to an inline natural-language claim and used to train or audit the hidden states of the rationale span.

Lean and LeanCheck. Lean is an interactive theorem prover and functional programming language used to state mathematical theorems and mechanically check proofs (de Moura & Ullrich, 2021). A Lean proof either elaborates and is accepted by the kernel, or fails with an error. LeanCheck is our lightweight formal-verifier dataset built around this binary signal: each example contains a theorem, a candidate proof, a short informal rationale, and a verifier-derived label indicating whether Lean accepts the proof. The purpose is not to solve theorem proving end-to-end, but to test whether the selected text span encodes the formal accept/reject signal under controlled counterfactual swaps.

KataGo and Go positions. KataGo is a strong open-source Go engine that combines neural-network evaluation with search in the AlphaGo/AlphaZero tradition (Silver et al., 2016; 2017; Wu, 2019). For a board position it can estimate quantities such as win rate, score lead, ownership, and principal variations. We use KataGo as a dense programmatic verifier: instead of a single binary proof label, it supplies structured numeric or categorical targets for complex Go states. This makes the setting useful for testing whether natural-language commentary can encode verifier-derived information in a high-dimensional domain.

4 Method

The method has three components: a *sequence format* that places a natural-language rationale and an explicit inline claim in the model's output, a *consistency objective* that trains rationale-span hidden states to predict the programmatic verifier's label, and an optional *information-flow constraint* (the BEC attention mask) that structurally prevents the claim head from bypassing the rationale. We then define a *diagnostic ladder* that escalates from decodability through counterfactual following to causal influence. The following subsections define each component; implementation details (models, training) appear at the end of the consistency-objective subsection.

4.1 Verifier-coupled sequence format

For an input problem or state x, a model generates a rationale r and an inline claim c in the format `[CONTEXT] x [RAT] r [CLAIM] c [EOS]`. A programmatic verifier V supplies a target $y = V(x)$, such as a proof-checker accept/reject label, a win-rate bucket, a complexity class, or a factual evidence label. The inline claim is the text location where the model commits to this verifier-grounded statement.

4.2 Consistency objective

Let h_i be the model hidden states and S_{rat} the token span between `[RAT]` and `[CLAIM]`. A pooling operator produces $z_{\text{rat}} = \text{Pool}(\{h_i : i \in S_{\text{rat}}\})$, and a consistency head g predicts $\hat{y}_{\text{cons}} = g(z_{\text{rat}})$. Training uses $L = L_{\text{LM}} + \lambda_{\text{claim}} L_{\text{claim}} + \lambda_{\text{cons}} L_{\text{cons}}$, where L_{LM} is next-token prediction, L_{claim} supervises the inline claim when applicable, and L_{cons} predicts the verifier target from the selected span. Different variants change the pooling span: rationale-only, proof-only, full-sequence, claim-only, wrong-span, or random-label controls.

In practice, g is a single linear layer ($\text{Linear}(d_{\text{model}}, |\mathcal{Y}|)$ where $|\mathcal{Y}|$ is the number of verifier label classes), and Pool is mean pooling over the span tokens. The base model is either a small decoder-only transformer trained from scratch (synthetic, KataGo, and code experiments; 2–4 layers, $d_{\text{model}} \in \{64, 256\}$) or a pretrained GPT-2 fine-tuned on the task data (LeanCheck and FEVER experiments). Training uses AdamW on the combined loss; no reinforcement learning is applied. Full hyperparameters are in Section F.

4.3 Information-flow constraints

An optional architectural complement to the consistency objective is a structured attention mask that constrains which tokens each output can attend to. In the Board-Explanation-Claim (BEC) variant, the input sequence is divided into three blocks: board or context tokens (B), explanation tokens (E),

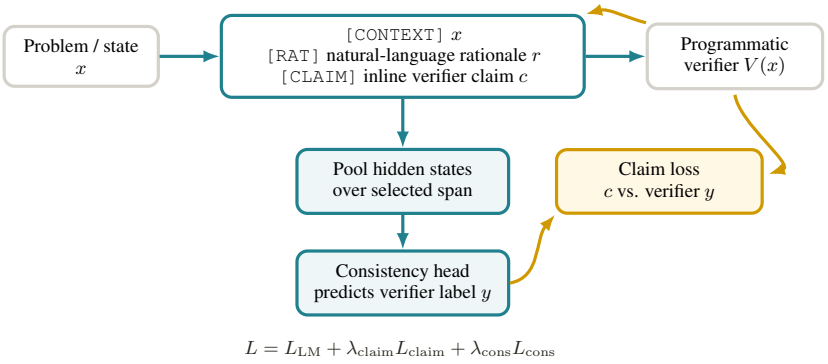

Figure 1. **Verifier-coupled reasoning.** A model generates a rationale and inline claim. A programmatic verifier supplies the target label. The consistency head predicts that label from pooled hidden states over the selected text span, making rationale-to-claim coupling directly measurable.

Table 1. Diagnostics used to interpret verifier-coupled reasoning.

Diagnostic	Supports	Caveat
Claim acc.	Correct claim	Rationale may be post-hoc
Head acc.	Label decodable	Not causal by itself
Swap	Span-specific following	May exploit templates
Controls	Not just capacity	Not sufficient alone
Patching	Causal influence	Needs careful baselines

and claim tokens (C). Board tokens use standard causal self-attention. Explanation tokens attend to all board tokens and to preceding explanation tokens. Claim tokens attend *only* to a windowed subset of explanation tokens—not to board tokens and not to other claims. This makes it structurally impossible for the claim head to bypass the explanation and read verifier-relevant information directly from the input. The BEC mask is implemented as an additive attention bias and is used in the Go GPT-OSS experiments reported in Section A. Figure 2 illustrates the attention pattern.

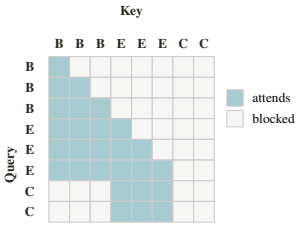

Figure 2. **BEC attention mask.** Board (B) tokens use standard causal attention. Explanation (E) tokens attend to all Board tokens and preceding Explanation tokens. Claim (C) tokens attend *only* to Explanation tokens, structurally preventing the claim head from bypassing the explanation.

4.4 Diagnostic ladder

The central distinction is between *decodability* and *faithfulness*. The consistency head measures whether verifier information can be decoded from rationale states. Stronger diagnostics ask whether the representation follows counterfactual swaps and whether intervening on it changes claim predictions.

5 Experiments

Each setting tests a different facet of the decodability–faithfulness gap. Synthetic experiments isolate the mechanism and provide causal evidence via activation patching. LeanCheck tests formal-verifier coupling with counterfactual span separation. KataGo extends coupling to dense, non-binary verifier labels. Code coupling reveals the central failure mode: near-perfect decodability with zero generation quality. FEVER adds a real-language setting with an accuracy–sensitivity trade-off. Together, these settings trace the diagnostic ladder: each probes a progressively stronger condition, and each reveals limitations the previous level misses. Detailed per-variant tables are in Section A.

5.1 Synthetic: controlled coupling and interventions

The synthetic setting is the cleanest place to isolate the mechanism. Each example contains a latent state, a natural-language rationale, and a deterministic claim label. Early runs with eight latent states showed the basic effect: without consistency loss, the classifier decoding claim labels from rationale states was near zero (3.9%) even when generation accuracy reached 75.0%; consistency-trained variants reached 100.0% classifier accuracy, with generation accuracy varying by pooling span. This already suggested that the LM objective can learn to emit claims without making rationale states predictive of them.

The stronger synthetic result comes from scaled convergence and intervention runs. At convergence, all main variants reached 100% generation accuracy, but the baseline stopped reading rationales: it could generate the right claim while its rationale representation did not carry the relevant label signal. By contrast, consistency-trained models preserved rationale-to-claim coupling. A claim-only pooling negative control reached only 43% classifier accuracy, showing that the result depends on the selected span rather than merely on model capacity. In a harder overlapping-vocabulary run, rationale templates shared

roughly 50% of token positions across latent states. Even there, `rationale_only` and `earlier_token_only` reached 100% generation accuracy, 100% rationale-pooled classifier accuracy, and 100% counterfactual classifier following; `full_sequence` kept 100% classifier coupling but had lower generation accuracy at 81.25%. The no-consistency baseline still had 100% generation accuracy but only 4.69% rationale-pooled classifier accuracy and 6.25% counterfactual classifier following.

Most importantly, hidden-state intervention tests provide causal evidence—testing whether rationale representations merely correlate with the claim (decodability) or actually determine it (causal influence), the central distinction in the diagnostic ladder. Patching rationale hidden states at layer 0 changed the downstream claim in 73–89% of cases for consistency-trained variants, compared with 31% for the baseline. At layer 1 (the final block), all variants show 0% following of the patched states, confirming that the intervention must occur early enough for downstream blocks to propagate the signal. The full per-layer, per-variant breakdown is in Section A.3. This directly addresses the diagnostic ladder: consistency loss is not only making labels decodable; in this controlled setting, rationale activations causally affect claim behavior.

5.2 LeanCheck: formal-verifier span separation

LeanCheck instantiates verifier-coupled reasoning with a formal proof checker. Each example contains a Lean theorem, a candidate proof, a natural-language rationale, and an inline claim indicating whether Lean accepts or rejects the proof. The model is not asked to synthesize proofs; the task isolates whether informal rationale representations encode formal verifier outcomes. The run uses GPT-2 with 1,000 Lean-checked training examples, 200 evaluation examples, 200 counterfactual swaps, and 200 minimal-pair rows. The data covers natural-number equalities, propositional logic, and simple list lemmas, with negative examples generated by wrong lemmas, renamed variables, deleted proof lines, replacement tactics, missing premises, wrong pairings, and adversarial near misses.

Concretely, the data covers six Lean 4 theorem templates in three families: natural-number arithmetic (`Nat.add_comm`, `Nat.add_zero`), propositional logic (`exact hp`, `And.intro hp hq`), list algebra (`List.append_nil`), and reflexive equality. Negative examples apply one of seven mutations to the correct proof. *Wrong-lemma* substitutes a type-incorrect lemma—for example, `Nat.mul_comm a b` for an addition-commutativity goal, producing a type mismatch because $a \cdot b = b \cdot a$ does not close $a + b = b + a$. *Rename-variable* introduces an unbound name (`And.intro hp hr` when only `hp` and `hq` are in scope). *Delete-final-line* empties the proof body, leaving unsolved goals. *Replace-tactic* substitutes a plausible but insufficient tactic such as `rfl` for a non-trivially-equal goal. The remaining strategies—wrong-pairing, missing-premise, and adversarial-near-miss—cover cross-family confusion, removed hypotheses, and syntactically plausible proofs of subtly different propositions. All 1,000 training labels come from the Lean 4 kernel, not human annotation. Rationales are sampled from per-domain and per-mutation paraphrase pools (five negative paraphrases per mutation strategy, four positive paraphrases per domain) to discourage template memorization.

LeanCheck gives the cleanest span-specific result. Without consistency loss, classifier accuracy is near chance: `lm_only` reaches 49.0% and `no_consistency_loss` reaches 51.0%. Wrong-span pooling is also near chance at 52.0%, and random-label consistency is 41.0%. In contrast, `rationale_only`, `proof_only`, and `full_sequence` all reach 100.0% classifier accuracy and 100.0% minimal-pair flip accuracy.

The key result is the counterfactual split. Counterfactual examples are constructed by cross-wiring proof and rationale evidence. In one direction, a correct proof (e.g. `exact And.intro hp hq`, which Lean accepts) is paired with a negative rationale ("A proof of p alone does not prove the conjunction p and q") and the label FAILS. In the mirror direction, a broken proof (`exact hp`, which Lean rejects because `hp` has type `p` rather than `p ∧ q`) is paired with a positive rationale ("The candidate proof and theorem target align at the type-checker level") and the label VERIFIES. The 200 counterfactual rows split evenly between these two swap directions, so both the rationale-pooled and proof-pooled heads face unambiguous label conflicts.

The `rationale_only` head follows the swapped rationale label 100.0% of the time and almost never follows the proof label (1.0%). The `proof_only` head shows the mirror image: it follows the proof label 100.0% of the time and the rationale label only 1.0%. This perfect separation is exactly what the mechanism predicts (Figure 3). It shows that the selected text span determines which verifier signal the head follows.

Minimal pairs provide a complementary diagnostic. Each pair shares the same theorem with exactly one proof-line mutation: for example, `Nat.add_comm a b` (accepted) versus `Nat.mul_comm a b` (rejected, since multiplication commutativity cannot close an addition goal). All consistency-trained variants reach 100% minimal-pair flip accuracy, confirming that even single-mutation differences are reliably reflected in the pooled representation. Controls (`lm_only`, `no_consistency_loss`, `wrong_span`) reach 0%.

LeanCheck activation patching provides additional evidence. For `rationale_only`, rationale-span patches produce

a large positive head-logit effect (11.58) and a positive rationale-minus-random effect (4.52). Baselines do not show the same head-level pattern. LM-logit patching is noisier, but the head-patching results support the claim that trained rationale states influence the verifier classifier in the intended direction.

Example: LeanCheck counterfactual pair. Both examples use the same theorem: `example (p q : Prop) (hp : p) (hq : q) : p ∧ q := by`. The rationales are cross-swapped so that proof and rationale labels conflict.

(a) Proof: `exact And.intro hp hq` — Lean accepts.
Rationale: "A proof of p alone does not prove the conjunction p and q."
Rationale-pooled head predicts **FAILS** (follows rationale, not proof).

(b) Proof: `exact hp` — Lean rejects (`hp` has type `p` but expected `p ∧ q`).
Rationale: "The candidate proof and theorem target align at the type-checker level."
Rationale-pooled head predicts **VERIFIES** (follows rationale, not proof).

The rationale-pooled head follows the rationale in both directions; the proof-pooled head does the mirror, following the proof 100% of the time.

Figure 3. LeanCheck counterfactual pair from the evaluation data. The consistency head follows whichever span it pools over, producing perfect directional separation under label conflict.

5.3 KataGo: dense domain verifier coupling

KataGo provides dense programmatic labels for Go positions, building on the neural-network and tree-search paradigm behind superhuman Go systems (Silver et al., 2016; 2017; Wu, 2019). The dataset contains 3,748 positions from 954 reviewed games (3,373 train, 375 eval), predominantly 19×19 boards. Each position is linearized into a symbolic token sequence: metadata tokens for board size, side to move, komi, and ruleset, followed by stone-coordinate tokens such as `B_Q4` or `W_D17`, sorted by color and then alphabetically. A natural-language commentary span follows the position tokens, and the sequence ends with an inline claim token encoding the KataGo-derived win-rate bin (V0–V9, each spanning roughly 10 percentage points). The win-rate distribution is U-shaped: bins V0 and V9 (decisive positions) account for 56% of the data, reflecting the natural distribution of positions sampled from real games (Figure 4).

The model is a 4-layer decoder-only transformer ($d_{\text{model}} = 256$, 8 heads, $d_{\text{ff}} = 1024$) trained from scratch with three output heads sharing the same hidden states: a language-modeling head over the full vocabulary, a scalar head that reads the hidden state at the claim-token position and predicts continuous win probability via sigmoid, and a consistency head that mean-pools hidden states over the commentary span and classifies the win-rate bin. The five ablations vary which heads receive gradient: `lm_only` trains only the LM head; `no_consistency_loss` adds the scalar head; `rationale_only` adds only the bin-classification head; `full_consistency` trains all three; and `random_consistency` trains the bin head on shuffled labels as a control.

This setting matters because it moves beyond binary accept/reject labels. Go commentary requires mapping a complex board state into qualitative natural-language explanation while grounding the claim in an external engine. The result is strong: `lm_only` reaches 1.6% claim-bin accuracy and `no_consistency_loss` reaches 25.1%. With consistency training, `rationale_only` reaches 78.7% and `full_consistency` reaches 81.1%, while `random_consistency` remains low at 16.0%. Notably, `full_consistency` also achieves near-perfect scalar regression (Pearson $r = 0.994$, MAE $= 0.028$), showing that the bin-classification objective does not interfere with fine-grained win-rate prediction. On counterfactual swaps, the consistency-trained variants follow the swapped commentary label 52–55% of the time versus only 14.5% following the original position's label, while baselines show no such asymmetry. This confirms that natural-language commentary states can be trained to encode dense verifier-derived labels, not only toy latent states or formal accept/reject signals.

Example: KataGo dense verifier coupling. Position: 19×19, move 5, White to play, komi 6.5. Black: Q15, Q10, Q5. White: D17, D3.

Commentary: "[. . .] I would prefer to be white here. To me the 4th line gives influence but the tight placement compared to san-ren-sei is territory focussed. The structure is very easy to reduce with both the 3-3 and 3-4 points open so there is no urgency for white to reduce."

KataGo verifier targets: win rate 29.2% Black (bin 3/9), score lead −1.86, phase: opening, urgency: medium, best move: R3.

The consistency head predicts the win-rate bin from commentary-span hidden states. With consistency training, claim-bin accuracy reaches 81.1% vs. 25.1% without.

Figure 4. KataGo example from the training data. Natural-language Go commentary is paired with dense engine-derived targets; the model must encode the win-rate bucket from the commentary span.

5.4 Code: representation coupling succeeds, faithful generation does not

The code experiment asks whether the mechanism generalizes to algorithmic claims such as time complexity, space complexity, algorithm class, loop structure, key operation, access pattern, auxiliary structures, input mutation, correctness status, and edge-case handling.

Coupling succeeds. In the standard three-claim ab-

Example: coupling succeeds, the explanation is fluent but unfaithful. Input: `selection_sort`, a standard nested-loop sort that finds and swaps the minimum element into position.

Oracle claims: time $O(n^2)$, space $O(1)$, correct.
Model structured claims: time $O(n^2)$, space $O(1)$, correct, all match the oracle.

Model prose (verbatim): "Computes the sum of a list with a single pass. Time complexity O(n), space O(1)."

The prose is fluent and grammatical but describes a different algorithm ("sum of a list") with the wrong time complexity ($O(n)$ rather than $O(n^2)$), while the structured claim tags correctly emit $O(n^2)$. This is representative: in the controlled re-run with a standard subword tokenizer (Section 6.2), generated explanations state the correct time complexity in only $\sim$18% of cases, below the 33% three-class chance rate, and never reproduce the code's reference explanation, identically for pretrained and from-scratch GPT-2. The failure mode is infidelity, not incoherence.

Figure 5. Code failure mode. The model's hidden states encode the verifier-derived claims and it emits the correct structured claim tags, but its generated prose, though fluent, describes an unrelated algorithm. The text shown is verbatim model output.

lation, the full consistency-loss variant reaches 100% coupling strength, 100% counterfactual swap influence, and 100% structured claim-tag accuracy, while `no_consistency_loss` reaches only 17.5% coupling and 5% swap influence. In the richer 12-claim ontology, `consistency_loss` reaches .986 mean coupling and about .978 swap influence, compared with .286 and $-.033$ for `no_consistency_loss`. The per-claim breakdown reveals an asymmetry: binary and low-cardinality claims are partially learnable from code structure alone (`mutates_input` reaches 77.4% even *without* consistency loss), whereas fine-grained claims are where the objective matters most: `algorithm_class` reaches 99.0% with consistency loss versus 13.4% without, and `key_operation` 99.8% versus 9.6% (Section A.10). Representation coupling is therefore won decisively.

Faithful generation does not. Winning coupling does not yield a faithful explanation. We assess generation in a controlled run that decodes with a standard subword tokenizer (GPT-2, both pretrained and trained from scratch; Section 6.2). The generated prose is fluent and grammatical (no degeneration, no repetition), but it is *unfaithful*: it describes a different algorithm than the input code. Given `selection_sort`, the model emits the correct structured claims ($O(n^2)$, $O(1)$, correct) while its prose reads "Computes the sum of a list with a single pass" and asserts the wrong complexity (Figure 5). Across the reviewed examples the generated explanation states the correct time complexity in only about 18% of cases, below the 33% three-class chance rate, and never reproduces the code's reference explanation; pretrained and from-scratch models behave identically. The code result is thus neither a coupling failure nor an incoherence failure: it shows that near-perfect coupling metrics can be won while the generated explanation remains unfaithful to the code it accompanies.

Training dynamics. The two outcomes separate cleanly over training. Coupling strength reaches 98.6% by epoch 5 for the consistency-loss variant and stays there; the no-consistency baseline never exceeds 29% across 20 epochs (Table 22). Faithful generation, by contrast, never emerges at any epoch, because nothing in the training objective rewards the *generated text* for matching the verifier: the consistency loss shapes the rationale representation, while the language-modeling loss is trained on explanations that are themselves not tied to the code (Section 6.2). Coupling is learned quickly and stably; faithful explanation is not learned at all.

5.5 FEVER: evidence sensitivity in real language

FEVER tests verifier-coupled reasoning on real-language evidence verification (Thorne et al., 2018). The model sees evidence text, a claim sentence, and an inline label token (SUPPORTS, REFUTES, or NOT ENOUGH INFO). The key diagnostic holds the claim text fixed while swapping evidence, isolating whether the model reads evidence or defaults to claim-text shortcuts. We report results from a pretrained GPT-2 run on 50k train / 5k eval examples and a from-scratch 4-layer transformer on the same split.

Evidence-only pooling has lower raw classifier accuracy than claim-visible pooling but better isolates evidence sensitivity. In the pretrained run, `evidence_only_pooling` reaches about 44% classifier accuracy and 81% generation accuracy; on matched counterfactuals, it follows swapped evidence about 46% of the time and the original label about 23%. Claim-visible pooling (`full_sequence`, `claim_only`) reaches about 84% raw classifier accuracy but shows more inertia toward the original claim, consistent with leveraging the claim text rather than parsing the evidence. Random-label evidence pooling falls near chance at about 23%.

The from-scratch run makes the gap more dramatic (Table 2): `full_sequence_pooling` reaches 99.0% classifier accuracy and 97.2% counterfactual swap following—near-perfect coupling when the claim text is visible—while evidence-only pooling stays at 44% classifier accuracy and 39% swap following. This contrast shows that a model trained from scratch can learn the claim-label correlation almost perfectly while still struggling to extract the same signal from evidence text alone. The evidence-only difficulty is not a pretrained shortcut artifact; it reflects a genuine asymmetry in how much verifier-relevant information is accessible from different text spans.

FEVER thus contributes a distinct lesson: in natural-language settings, the span that provides the strongest raw

Table 2. FEVER from-scratch run. Full-sequence pooling achieves near-perfect coupling while evidence-only pooling stays near 44%, revealing an accuracy-sensitivity trade-off that final-answer accuracy would hide.

Variant	Cls	Gen	Swap	Orig
`full_sequence`	.990	.972	.972	.014
`claim_only`	.984	.966	.960	.022
`evidence_only`	.440	.812	.390	.320
`no_consistency`	.498	.806	.340	.360

Table 3. Main experimental arc. Metrics are setting-specific primary diagnostics and should be compared within settings; acc. denotes accuracy.

Setting	Diagnostic	Trained	Control
Synthetic	layer-0 patch	73–89%	31%
LeanCheck	span follows	100%	50–52%
KataGo	claim-bin acc.	81.1%	25.1%
Code	mean coupling	98.6%	28.6%
FEVER	evid. swap fol.	46%	34%

accuracy (claim text) may not be the span that tests whether the model actually processes the evidence. The evidence-only constraint exposes an accuracy-sensitivity trade-off invisible to generation-level evaluation.

6 Discussion

6.1 What the settings jointly show

The cross-setting pattern is now clearer. Synthetic experiments show that consistency loss can prevent the LM objective from bypassing rationale states and can create causal intervention effects. LeanCheck shows that the mechanism works in a formal-verifier setting and that the selected pooling span determines whether the classifier follows the rationale or proof under counterfactual conflict. KataGo shows that the same idea scales to dense, non-binary verifier labels in a complex domain. Code shows the key failure mode: representation coupling can be nearly perfect while generated explanations are unusable. FEVER contributes a complementary lesson about real-language settings: the span that maximizes raw accuracy (claim text at 99%) may not be the span that tests whether the model actually processes the evidence (evidence-only at 44%).

This distinction matters for mathematical and scientific agents. If the goal is to audit reasoning, a high consistency-head score is useful: it tells us that verifier-relevant information is present in the selected representation. If the goal is to trust the generated explanation, decodability alone does not suffice—the explanation must also be fluent, semantically grounded, and causally connected to the emitted claim. This is what the diagnostic ladder is designed to test.

6.2 The decodability–faithfulness gap is structural

The experiments establish that supervised consistency training reliably produces *decodability* (verifier labels become linearly recoverable from rationale-span hidden states, reaching near-perfect coupling in the code setting) but that decodability does not, on its own, yield *faithful generation*. Two limits recur. First, span selection trades diagnostic power against predictive accuracy: the span that most cleanly isolates verifier sensitivity is not the span that maximizes raw accuracy. Second, and more fundamentally, the consistency objective shapes the rationale *representation* without constraining the generated *text*. One might worry that the code experiment's gap is incidental to the objective and instead reflects a capacity-starved from-scratch model, or a too-low loss weight. We report two controlled experiments showing it is neither.

The gap is not capacity-driven. We hold the code-coupling task, dataset, and consistency objective fixed and train two models that differ *only* in initialization: a pretrained GPT-2 (124M) and a from-scratch GPT-2 of identical architecture and subword tokenizer, three seeds each (Table 19). If unfaithful generation were a capacity artifact, generative pretraining should narrow the gap. It does not: both conditions reach identical coupling (1.00 ± 0.00) and statistically identical BLEU-1 (0.191 ± 0.017 vs. 0.183 ± 0.047; 4-gram repetition 0.000 in either); pretraining only accelerates convergence (epoch-1 LM loss 1.19 vs. 2.59). The gap is therefore a property of the *objective*, not of model scale or generative competence. The comparison also yields a methodological correction: under a standard subword tokenizer the generated prose is fluent in *both* conditions, so the operative failure is not incoherence but *infidelity*, i.e. the model emits a well-formed explanation of an unrelated algorithm.

Reweighting the loss cannot close the gap. Could upweighting L_{cons} relative to L_{LM} pull generation toward faithfulness? It cannot, for a structural reason: in $L = L_{\text{LM}} + \lambda_{\text{cons}} L_{\text{cons}}$, L_{cons} is a loss on a *representation*, a probe over pooled hidden states, and only L_{LM} has a gradient path to emitted tokens. Sweeping λ_{cons} from 1 to 10^4 (Table 20) leaves every generation metric unchanged: coupling stays at 1.00 and prose quality stays flat. Coupling reaches 1.00 within the first epoch, after which L_{cons} and its gradient collapse to zero, so λ_{cons} multiplies a vanished gradient regardless of magnitude, and the final LM loss is identical across the entire sweep. A loss on the representation has no lever on generation, and no choice of weight converts it into one.

Scope of the faithfulness claim. We use "unfaithful" here in the operational sense that the generated prose and the inline structured claim refer to different algorithms; we do not equate it with faithfulness in the full causal sense of Jacovi & Goldberg (2020) or Lanham et al. (2023), which would require showing the explanation reflects the causal computation behind the answer (e.g. generation-side activation patching on a fluent pretrained model). Our claim is the weaker

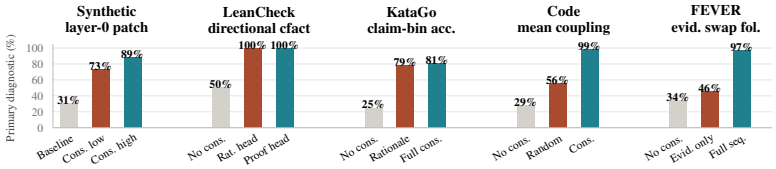

Figure 6. **Primary coupling diagnostics across all five settings.** Metrics are setting-specific: synthetic layer-0 intervention effect, LeanCheck directional counterfactual (cfact) following, KataGo claim-bin accuracy (acc.), code mean coupling, and FEVER evidence-swap following. In LeanCheck, the two 100% bars are different directional tests: the rationale-pooled head follows the rationale label, while the proof-pooled head follows the proof label. FEVER shows the accuracy-sensitivity trade-off: evidence-only pooling (46%) has lower raw accuracy than full-sequence (97.2%) but better isolates evidence sensitivity. Values should be compared within each setting, not as one universal score.

but substantive one that decodable verifier information in rationale hidden states is not sufficient for the generated explanation to refer to the same object as its verified claim. One design choice interacts with this scope: the LM head is trained on (CODE, MISMATCHED-EXPLANATION) pairs by construction, so the prose is unfaithful partly by training-data design. The structural finding survives this caveat, since a representation-side loss cannot, even at $\lambda_{\text{cons}} = 10^4$, pull generation back toward the verified claim; a fully causal account in the Lanham/Jacovi sense will require generation-side interventions we leave to future work.

What closing the gap requires. Together these results show that the gap is structural: consistency training is an effective *diagnostic* (it measures whether verifier-relevant information is present in the rationale representation, a necessary precondition for faithfulness) and an effective representation-shaping prior, but it is not, and cannot by reweighting become, a generation objective. Closing the gap requires an objective whose gradient reaches the generated tokens and rewards them for verifier consistency. The natural instrument is reinforcement learning with programmatic verifier rewards. Inline claims create checkable units: the model generates a rationale and an inline claim, the verifier scores the emitted claim, and the policy is rewarded or penalized on what it actually produced rather than on an auxiliary probe. Such RL should be paired with the intervention-based evaluation developed here, so that optimized policies can be tested for genuine faithfulness rather than reward-hacking.

7 Conclusion

Verifier-coupled reasoning adds a simple constraint to natural-language rationales: the hidden states over the rationale should encode the programmatically verified claim the model emits. Across synthetic, LeanCheck, KataGo, code, and FEVER settings, this mechanism succeeds as a representation-coupling tool and produces measurable, span-specific verifier alignment. LeanCheck illustrates how rationale-only and proof-only pooling can perfectly separate under counterfactual conflict, and synthetic interventions confirm that consistency-trained activations causally shift claim behavior. The code experiment reveals an equally important finding: coupling reaches 98.6% by epoch 5 while prose quality never improves, and per-claim analysis shows that fine-grained claims benefit far more from consistency loss than binary claims. FEVER contributes a complementary insight: evidence-only pooling isolates genuine evidence sensitivity at 44% accuracy, while claim-visible pooling inflates to 99% by leveraging the claim text rather than parsing evidence. Together, these results show that consistency losses are effective tools for measuring and shaping verifier-aligned representations; closing the remaining gap to fully faithful reasoning is a natural target for verifier-reward RL, architectural information-flow constraints such as the BEC mask, and generation-quality objectives.

References

Chen, M., Tworek, J., Jun, H., Yuan, Q., Pinto, H. P. d. O., Kaplan, J., Edwards, H., Burda, Y., Joseph, N., Brockman, G., et al. Evaluating large language models trained on code, 2021. URL https://arxiv.org/abs/2107.03374.

Christiano, P. F., Leike, J., Brown, T., Martic, M., Legg, S., and Amodei, D. Deep reinforcement learning from human preferences. In *Advances in Neural Information Processing Systems*, 2017. URL https://arxiv.org/abs/1706.03741.

Cobbe, K., Kosaraju, V., Bavarian, M., Chen, M., Jun, H., Kaiser, L., Plappert, M., Tworek, J., Hilton, J., Nakano, R., et al. Training verifiers to solve math word problems, 2021. URL https://arxiv.org/abs/2110.14168.

de Moura, L. and Ullrich, S. The lean 4 theorem prover and programming language. In *Automated Deduction – CADE 28*, pp. 625–635. Springer, 2021. doi: 10.1007/978-3-030-79876-5_37. URL https://link.springer.com/10.1007/978-3-030-79876-5_37.

DeepSeek-AI. Deepseek-r1: Incentivizing reasoning capability in llms via reinforcement learning, 2025. URL https://arxiv.org/abs/2501.12948.

Dong, Y., Jiang, X., Jin, Z., and Li, G. Self-play with execution feedback: Improving instruction-following capabilities of large language models, 2024. URL https://arxiv.org/abs/2406.13542.

Hewitt, J. and Liang, P. Designing and interpreting probes with control tasks. In *Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing*, 2019. URL https://aclanthology.org/D19-1275/.

Jacovi, A. and Goldberg, Y. Towards faithfully interpretable nlp systems: How should we define and evaluate faithfulness? In *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics*, 2020. URL https://aclanthology.org/2020.acl-main.386/.

Lambert, N., Morrison, J., Pyatkin, V., Huang, S., Ivison, H., Brahman, F., Miranda, L. J. V., Liu, A., Dziri, N., Lyu, S., et al. Tülu 3: Pushing frontiers in open language model post-training, 2024. URL https://arxiv.org/abs/2411.15124.

Lanham, T., Chen, A., Radhakrishnan, A., Steiner, B., Denison, C., Hernandez, D., Li, D., Durmus, E., Hubinger, E., Kernion, J., et al. Measuring faithfulness in chain-of-thought reasoning, 2023. URL https://arxiv.org/abs/2307.13702.

Lightman, H., Kosaraju, V., Burda, Y., Edwards, H., Baker, B., Lee, T., Leike, J., Schulman, J., Sutskever, I., and Cobbe, K. Let's verify step by step, 2023. URL https://arxiv.org/abs/2305.20050.

Luo, L., Liu, Y., Liu, R., Phatale, S., Lara, H., Li, Y., Shu, L., Zhu, Y., Meng, L., Sun, J., and Rastogi, A. Improve mathematical reasoning in language models by automated process supervision, 2024. URL https://arxiv.org/abs/2406.06592.

Marks, S. and Tegmark, M. The geometry of truth: Emergent linear structure in large language model representations of true/false datasets, 2023. URL https://arxiv.org/abs/2310.06824.

Meng, K., Bau, D., Andonian, A., and Belinkov, Y. Locating and editing factual associations in gpt. In *Advances in Neural Information Processing Systems*, 2022. URL https://arxiv.org/abs/2202.05262.

Ouyang, L., Wu, J., Jiang, X., Almeida, D., Wainwright, C., Mishkin, P., Zhang, C., Agarwal, S., Slama, K., Ray, A., et al. Training language models to follow instructions with human feedback. In *Advances in Neural Information Processing Systems*, 2022. URL https://arxiv.org/abs/2203.02155.

Pimentel, T., Valvoda, J., Hall Maudslay, R., Zmigrod, R., Williams, A., and Cotterell, R. Information-theoretic probing for linguistic structure. In *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics*, 2020. URL https://aclanthology.org/2020.acl-main.420/.

Shao, Z., Wang, P., Zhu, Q., Xu, R., Song, J., Bi, X., Zhang, H., Zhang, M., Li, Y. K., Wu, Y., and Guo, D. Deepseekmath: Pushing the limits of mathematical reasoning in open language models, 2024. URL https://arxiv.org/abs/2402.03300.

Silver, D., Huang, A., Maddison, C. J., Guez, A., Sifre, L., van den Driessche, G., Schrittwieser, J., Antonoglou, I., Panneershelvam, V., Lanctot, M., et al. Mastering the game of go with deep neural networks and tree search. *Nature*, 529:484–489, 2016. URL https://www.nature.com/articles/nature16961.

Silver, D., Schrittwieser, J., Simonyan, K., Antonoglou, I., Huang, A., Guez, A., Hubert, T., Baker, L., Lai, M., Bolton, A., et al. Mastering the game of go without human knowledge. *Nature*, 550:354–359, 2017. URL https://www.nature.com/articles/nature24270.

Tanneru, S., Ley, D., Agarwal, C., and Lakkaraju, H. On the hardness of faithful chain-of-thought reasoning in large language models, 2024. URL https://arxiv.org/abs/2406.10625.

Thorne, J., Vlachos, A., Christodoulopoulos, C., and Mittal, A. Fever: A large-scale dataset for fact extraction and verification. In *Proceedings of the 2018 Conference of the North American Chapter of the Association for Computational Linguistics*, 2018. URL https://aclanthology.org/N18-1074/.

Turpin, M., Michael, J., Perez, E., and Bowman, S. R. Language models don't always say what they think: Unfaithful explanations in chain-of-thought prompting, 2023. URL https://arxiv.org/abs/2305.04388.

Uesato, J., Kushman, N., Kumar, R., Song, F., Siegel, N., Wang, L., Creswell, A., Irving, G., and Higgins, I. Solving math word problems with process- and outcome-based feedback, 2022. URL https://arxiv.org/abs/2211.14275.

Wang, P., Li, L., Shao, Z., Xu, R. X., Dai, D., Li, Y., Chen, D., Wu, Y., and Sui, Z. Math-shepherd: Verify and reinforce llms step-by-step without human annotations, 2023a. URL https://arxiv.org/abs/2312.08935.

Wang, X., Wei, J., Schuurmans, D., Le, Q., Chi, E., Narang, S., Chowdhery, A., and Zhou, D. Self-consistency improves chain of thought reasoning in language models. In *International Conference on Learning Representations*, 2023b. URL https://arxiv.org/abs/2203.11171.

Wei, J., Wang, X., Schuurmans, D., Bosma, M., Ichter, B., Xia, F., Chi, E., Le, Q., and Zhou, D. Chain-of-thought prompting elicits reasoning in large language models. In *Advances in Neural Information Processing Systems*, 2022. URL https://arxiv.org/abs/2201.11903.

Wu, D. J. Accelerating self-play learning in go, 2019. URL https://arxiv.org/abs/1902.10565.

Zhang, F. and Nanda, N. Towards best practices of activation patching in language models: Metrics and methods, 2023. URL https://arxiv.org/abs/2309.16042.

A Detailed Variant Tables

A.1 Synthetic controls and scalar-claim variants

The synthetic tables show why the auxiliary consistency objective is needed. The language-modeling objective can reach perfect generation while the rationale span carries little verifier information; the consistency-trained variants preserve span-level decodability, counterfactual following, and intervention sensitivity.

Table 4. Synthetic results included in the main narrative. Early small-scale results validate decodability; hard overlapping-vocabulary and intervention runs test robustness and causal effects. In the metric column, cls. means consistency-head classification accuracy, gen. means claim-token generation accuracy, cfact means counterfactual, and orig means original label.

Result	Variant	Metric	Value
Small synthetic	no consistency	cls. acc.	3.9%
Small synthetic	full sequence	gen. acc.	93.8%
Small synthetic	rationale only	gen. acc.	63.0%
Small synthetic	earlier token	gen. acc.	62.0%
Scaled convergence	all main variants	gen. acc.	100%
Scaled convergence	no consistency	cls. / cfact cls	8.8% / 7.8%
Scaled convergence	no consistency	cfact gen swap	0%
Negative control	claim-only pooling	cls. acc.	43%
Negative control	claim-only pooling	cfact swap / orig	48% / 8%
Hard overlap	no consistency	cls. / cfact cls	4.69% / 6.25%
Hard overlap	no consistency	gen. / cfact gen	100% / 100%
Hard overlap	rationale only	gen. / cls. / cfact	100% / 100% / 100%
Hard overlap	full sequence	gen. / cls. / cfact	81.25% / 100% / 100%
Hard overlap	earlier token	gen. / cls. / cfact	100% / 100% / 100%
Hard overlap	shuffled control	consistency variants	10.16%
Intervention	baseline	layer-0 effect	31%
Intervention	consistency trained	layer-0 effect	73–89%

Table 5. Scaled convergence (30 epochs, 5120 train) vs. smoke test (5 epochs, 512 train). Key deltas: the no-consistency baseline improves generation from 75% to 100% but classifier accuracy stays near chance; consistency-trained variants maintain perfect coupling at convergence. Gen is generation accuracy, Cls is rationale-pooled classifier accuracy, Cfact is counterfactual classifier swap following.

Variant	Gen_s	Gen_c	Cls_s	Cls_c	Cfact_s	Cfact_c
`no_consistency`	.750	1.000	.039	.088	.031	.078
`rationale_only`	.633	1.000	1.000	1.000	1.000	1.000
`full_sequence`	.938	1.000	1.000	1.000	1.000	1.000
`earlier_token`	.617	1.000	1.000	1.000	1.000	1.000

The generated-rationale scalar experiment is the closest synthetic analogue of the paper's sequence format: a model writes a rationale and emits a scalar or binned claim. This experiment produces the *strongest coupling result* across all settings: both `rationale_only` and `full_consistency` reach *perfect* 1.000 claim-bin accuracy and perfect counterfactual following (1.000 Orig, 0.000 Swap), meaning the rationale span is a lossless predictor of the verifier-derived bin. This is stronger than LeanCheck (which reaches 100% but on binary labels) and KataGo (81.1% on 10-way bins) because the model must generate the rationale tokens before they are pooled, so coupling is achieved through generated text, not just input-conditioned representations. The LM-only baseline reaches 0% bin accuracy despite 82% token accuracy, confirming that the standard LM objective does not create claim-predictive rationale states.

Table 6. Generated rationale plus scalar-claim experiment. Token is token-level generation accuracy, Bin acc. is claim-bin accuracy, MSE is mean squared error for the scalar claim, Swap is counterfactual following of the swapped rationale label, and Orig is following of the original label.

Variant	Token	Bin acc.	MSE	Swap	Orig
`lm_only`	.8231	.0000	.12835	.0313	.1172
`no_consistency`	.8197	.0410	.00014	.1602	.0352
`rationale_only`	.8199	1.0000	.09820	.0000	1.0000
`full_consistency`	.8166	1.0000	.00018	.0000	1.0000
`random_consistency`	.8190	.1641	.00029	.0664	.1406

A.2 Shuffled-pairing controls across synthetic variants

The shuffled-pairing control randomizes rationale–claim assignments at evaluation time: each rationale is paired with a uniformly random claim rather than its correct label. If the consistency head were memorizing surface templates instead of reading rationale content, shuffled accuracy would remain high. Table 7 shows that all variants and settings collapse to near the $1/8 = 0.125$ chance level under shuffled pairings, confirming that performance depends on correct rationale–claim alignment.

Table 7. Shuffled-pairing control accuracies across three synthetic settings (8 latent states, chance $= 0.125$). Shuf. Gen is generation accuracy under shuffled pairings; Shuf. Cls is consistency-head classification accuracy under shuffled pairings. All values are near chance, confirming that the consistency head reads rationale content rather than memorizing patterns.

Setting	Variant	Shuf. Gen	Shuf. Cls
Smoke test	no_consistency	.1016	.1172
	rationale_only	.0781	.1016
	full_sequence	.0938	.1016
	earlier_token	.0859	.1016
	claim_only_pool	.1016	.0859
Scaled run	no_consistency	.1289	.1641
	rationale_only	.1289	.1289
	full_sequence	.1289	.1289
	earlier_token	.1289	.1289
Hard overlap	no_consistency	.1016	.1719
	rationale_only	.1016	.1016
	full_sequence	.0938	.1016
	earlier_token	.1016	.1016

A.3 Hidden-state intervention: per-layer, per-variant breakdown

The hidden-state intervention experiment provides the strongest causal evidence in the synthetic setting. The methodology is as follows. For each sample pair (an original sequence with latent state A and a swapped sequence with latent state B) the evaluation (1) runs a forward pass with the original sequence and caches post-block hidden states at rationale token positions after each transformer block; (2) for each block i, runs a second forward pass with the swapped sequence but replaces the hidden states at rationale positions after block i with the cached original states, letting all subsequent blocks process the patched activations; and (3) reads the greedy next-token prediction at the final separator position (immediately before the claim span) to determine whether the model follows the original latent state's claim token or the swapped state's claim token. This procedure directly tests whether rationale-span activations at a given layer causally determine the downstream claim, rather than merely being decodable by a post-hoc probe.

Table 8 reports the full per-layer, per-variant results. The two-layer model (2 blocks, $d_{\text{model}} = 64$, 4 heads, $d_{\text{ff}} = 128$) was trained for 30 epochs on 5,120 examples with 8 latent states; intervention was evaluated on 64 sample pairs per variant (seed 42).

Table 8. Hidden-state intervention results by variant and layer. *Follows orig. HS* is the fraction of samples where patching rationale hidden states causes the model to predict the original latent state's claim token; *Follows swapped* is the fraction where the model still predicts the swapped sequence's claim token despite the patch. All values are over $n = 64$ sample pairs.

Variant	Layer	Follows orig. HS	Follows swapped
no_consistency_loss	0	.3125	.6875
no_consistency_loss	1	.0000	1.0000
rationale_only	0	.8906	.1094
rationale_only	1	.0000	1.0000
full_sequence	0	.7500	.2500
full_sequence	1	.0000	1.0000
earlier_token_only	0	.7344	.2500
earlier_token_only	1	.0000	1.0000

Two patterns stand out. First, the **layer-0 vs. layer-1 contrast** is sharp. At layer 0 (after the first transformer block), consistency-trained variants show 73–89% of samples following the patched original hidden states, compared with only 31% for the no-consistency baseline. At layer 1 (after the final block), *every* variant, including all consistency-trained models,

shows 0% following the original hidden states and 100% following the swapped tokens. This is expected: patching at the last layer is too late because the model has already committed to a prediction based on the full forward pass through both blocks. The intervention can only redirect behavior when it replaces activations early enough for downstream computation to propagate the patched signal.

Second, the `earlier_token_only` variant reveals a subtle **third-state phenomenon**. At layer 0, 73.4% of samples follow the original hidden states and 25.0% follow the swapped tokens, but these do not sum to 100%: the remaining $\sim$1.6% of samples predict a token that matches neither the original nor the swapped claim. In these cases the patched activations push the model into a third latent state not present in either input, suggesting that the intervention creates an out-of-distribution activation pattern for a small fraction of samples. The other consistency-trained variants (`rationale_only` and `full_sequence`) sum to 100%, indicating clean binary switching behavior under intervention.

A.4 LeanCheck formal-verifier results

LeanCheck is the cleanest formal-verifier result because the labels are mechanically generated by Lean and the counterfactuals separate proof and rationale evidence. The training set contains 1,000 examples distributed roughly uniformly across six domains: addition commutativity (182), list append-nil (179), conjunction introduction (175), reflexive equality (158), propositional hypothesis (157), and addition right-identity (149), with a balanced label split (520 VERIFIES, 480 FAILS). Among the 480 negative examples, the most common mutation types are replace-tactic (107), delete-final-line (102), and rename-variable (95), followed by wrong-lemma (63), wrong-pairing (62), adversarial-near-miss (59), and missing-premise (12). The 100 minimal pairs use primarily wrong-lemma (83) and rename-variable (17) mutations, chosen for their single-line minimal difference.

Table 9. LeanCheck GPT-2 run with Lean-derived labels. Cls is consistency-head classification accuracy, Gen is claim-token generation accuracy, Swap is counterfactual following of the swapped rationale label, Proof is following of the proof label, and Pair is minimal-pair flip accuracy.

Variant	Cls	Gen	Swap	Proof	Pair
`lm_only`	.490	.510	.495	.495	.000
`no_consistency`	.510	.525	.505	.505	.000
`rationale_only`	1.000	.640	1.000	.010	1.000
`full_sequence`	1.000	.990	.995	.005	1.000
`proof_only`	1.000	.915	.010	1.000	1.000
`random_consistency`	.410	.775	.400	.590	.070
`wrong_span`	.520	.540	.495	.515	.000

The activation-patching table shows a clear pattern consistent with the diagnostic ladder. Head-logit effects are large for the intended rationale-pooled model and close to zero or negative for several controls, which is the pattern predicted by the diagnostic ladder.

Table 10. LeanCheck activation patching summary. LM denotes language-model claim-token logits, Head denotes consistency-head logits, rat denotes rationale patching, and rnd denotes random-token patching. Head patching is more discriminative than LM-logit patching.

Variant	LM rat	LM rat-rnd	Head rat	Head rat-rnd
`lm_only`	6.086	5.410	.186	-.001
`no_consistency`	3.718	5.097	.001	.064
`rationale_only`	5.254	9.774	11.584	4.518
`full_sequence`	9.152	5.463	7.867	.224
`proof_only`	1.410	-5.208	.000	-3.758
`random_consistency`	6.135	-7.494	-.054	.025
`wrong_span`	9.574	10.741	.000	-.014

A.5 KataGo and Go claim-head results

KataGo tests whether the mechanism survives a dense domain verifier rather than a binary proof checker. The key pattern is that consistency-trained commentary spans encode engine-derived win-rate buckets, while LM-only, no-consistency, and random-label controls remain much weaker.

Table 11. KataGo win-probability coupling. Bin acc. is claim-bin accuracy from commentary-span states, MSE and MAE are scalar win-rate errors, Pearson and Spearman are win-rate correlations, Swap is counterfactual following of the swapped commentary label, and Orig is following of the original label.

Variant	Bin acc.	MSE	MAE	Pearson	Spearman	Swap	Orig
lm_only	.016	.1606	.3785	.362	.487	.10	.11
no_consistency	.251	.0016	.0268	.995	.932	.20	.21
rationale_only	.787	.1807	.3962	.154	.188	.52	.14
full_consistency	.811	.0020	.0280	.994	.918	.55	.14
random_consistency	.160	.0017	.0284	.995	.935	.17	.10

The Go GPT-OSS tables summarize a supervised claim-head experiment on Go positions. A GPT-OSS model was trained on a 1k train / 200 evaluation split to predict multiple structured claims derived from board state and KataGo-style analysis targets, including win-probability bins, score-lead bins, phase, contested regions, move urgency, and search surprise. This is not the same as the KataGo win-probability consistency experiment above: it tests whether a Go-language model can learn a broader multi-claim ontology before adding stronger verifier-coupling or RL objectives. Aggregate accuracy rises with training, but macro-F1 reveals class collapse; the model learns frequent labels much faster than rare or fine-grained tactical categories. A zero-shot baseline using SmolLM-3B (without fine-tuning) produces largely degenerate outputs on the same positions: repetitive prompt echoing, role-tag cycling, and no substantive Go commentary, confirming that domain-specific fine-tuning is necessary for coherent position analysis.

Table 12. Go GPT-OSS aggregate claim-head training on the 1k/200 split. Acc. denotes accuracy and F1 denotes macro-F1 averaged across label classes. Macro-F1 peaks before raw accuracy, indicating overfitting or majority-class collapse after epoch 4.

Epoch	Train acc.	Eval acc.	Train F1	Eval F1
1	.4858	.4889	.1953	.1955
2	.5272	.5228	.2440	.2375
3	.5561	.5539	.2747	.2674
4	.5873	.5544	.3964	.3206
5	.6288	.5617	.4573	.3123

Table 13. Go GPT-OSS per-claim results at epoch 4. Accuracy is raw per-head label accuracy and Macro-F1 is class-balanced F1. Global contestedness is the best-behaved head; several other heads have acceptable raw accuracy but weak macro-F1 because predictions collapse toward frequent labels.

Claim head	Accuracy	Macro-F1
win_prob_bin	.315	.177
score_lead_bin	.550	.171
phase_estimate	.755	.326
main_control_region	.300	.191
main_contested_region	.450	.342
global_contestedness	.865	.765
best_move_region	.550	.276
move_urgency	.540	.324
search_surprise	.665	.313

A.6 Code coupling and failure-mode results

Hidden-state coupling is high in several code variants, but manual review shows the generated explanations are not usable. Thus, probe or swap success cannot be equated with faithful explanation.

Table 14. Code coupling rich-ontology results. Coupling is mean per-claim classifier accuracy from the selected explanation span; Swap is counterfactual swap influence. Generation-quality metrics for this experiment are reported separately in Table 17 (a subword-tokenizer re-run); BLEU/ROUGE from the original word-level decode were artifacts and are omitted here.

Variant	Coupling	Swap
consistency_loss	.986	.978
no_consistency	.286	−.033
claim_only	.595	.122
random_label	.562	−.022
no_claim_attn	.986	.978
claims_from_expl	.986	.978

The simpler V1 and strict-flow variants show the same pattern from another angle: coupling can be made almost perfect with consistency loss and masking, but surface bottlenecks are harder and the resulting prose remains poor.

Table 15. Code coupling variants beyond the rich-ontology table. Coupling is mean per-claim classifier accuracy, Swap is counterfactual swap influence, Claim acc. is emitted structured-claim accuracy, and BLEU-1 is unigram overlap. V1 establishes the basic effect; V2 shows that strict hidden-state flow can preserve coupling, while surface bottlenecks weaken or invert swap behavior.

Variant	Coupling	Swap	Claim acc.	BLEU-1
V1 `consistency_loss`	1.000	1.000	1.000	.067
V1 `no_consistency`	.175	.050	–	.060
V2 `no_claim_attn`	1.000	1.000	1.000	.076
V2 `claims_from_expl`	1.000	1.000	.800	.071
V2 `surface_bottleneck`	.697	-.150	1.000	.068
V2 `surface_no_expl_lm`	.808	-.050	.100	.003

Table 16. Manual qualitative review of code explanations. Behavior is behavior-correct prose, Full prose is fully correct prose, and Time/Space/Bug are manually checked claim-value correctness counts out of 20. All variants fail the fully-correct prose and behavior-correct criteria, even when structured claims are often individually correct. Prose is fluent under a standard subword tokenizer (Table 17); the failure is infidelity, not incoherence. The Time/Space/Bug counts reflect the word-level run and understate structured-claim correctness.

Variant	Behavior	Full prose	Time	Space	Bug
`consistency_loss`	0/20	0/20	4/20	10/20	17/20
`claim_only`	0/20	0/20	5/20	16/20	17/20
`no_consistency`	0/20	0/20	5/20	11/20	17/20
`random_label`	0/20	0/20	5/20	9/20	17/20

A.7 Code coupling claim ontology and architectural variants

The rich-ontology code experiment uses 12 structured claim heads, each with an oracle-derived label: `time_complexity` (6 values: $O(1)$ through $O(n^3)$), `space_complexity` (4), `best_case_time` (6), `algorithm_class` (7: brute-force, divide-and-conquer, greedy, DP, etc.), `loop_structure` (6), `key_operation` (7: comparison, arithmetic, hashing, etc.), `access_pattern` (5), `auxiliary_structures` (7), `mutates_input` (2), `correctness_status` (5), `handles_empty_input` (3), and `handles_duplicates` (4)—62 total claim-value tokens. Per-claim difficulty varies: binary claims like `mutates_input` and `handles_empty_input` reach near-perfect accuracy even without consistency loss, while fine-grained claims like `algorithm_class` lag at 13.4% without consistency loss versus 98.6% with it.

The V2 code experiments test four architectural variants beyond the standard consistency-loss ablation. `no_claim_to_claim_attention` adds an additive attention bias that blocks claim tokens from attending to other claim tokens, forcing each claim to be predicted independently from the explanation span. `claims_from_explanation_only` restricts claim tokens to attend only to explanation tokens, enforcing a code→explanation→claim information path. Both achieve coupling near 1.0 and swap influence near 1.0, matching the standard consistency-loss variant. `surface_bottleneck_consistency` replaces hidden-state pooling with classification over softmax probability distributions at explanation positions, testing whether coupling can survive a surface-level bottleneck. This variant drops to 0.697 coupling and -0.15 swap influence, showing that the consistency signal requires access to hidden states, not just output distributions. `surface_no_expl_lm` further disables LM loss on explanation tokens, collapsing to BLEU 0.003 by epoch 3; the model produces no readable text at all.

Beyond selection sort (Figure 5), the same pattern appears across other functions under the subword-tokenizer re-run (Section A.8). For `caesar_cipher`, the model emits correct structured claims but generates fluent prose describing a different algorithm: "`longest_palindrome_substr` expands around every center (odd and even length) to find the longest palindromic substring. However, the final line uses stale `l` and `r` from the last expansion, which is a `wrong_condition` bug. Time $O(n^2)$; space $O(1)$ ..." For `is_prime`, the prose says "`binary_search_rec` implements binary search recursively, halving the search range each call and reaching $O(\log n)$ time in the worst case. The recursion stack uses $O(\log n)$ space ..." In every case the prose is fluent and describes an unrelated algorithm while the 12 structured claim tags are emitted correctly. The earlier "garbled" word-level decodes (e.g. "Computesprefixsumarray. O(n) time and O(n) space") were tokenizer artifacts rather than generation failures; the substantive failure is infidelity, not incoherence.

A.8 Code generation under a subword tokenizer

The code-experiment generation metrics in the tables above were decoded with a custom word-level tokenizer. To check whether the apparent generation failure was a property of the model or of that decoder, we re-ran the 12-claim rich ontology with a standard GPT-2 BPE tokenizer, on a pretrained and a from-scratch GPT-2 (124M) of identical architecture (19 templates, 3,000 examples, 20 epochs, seed 42). Table 17 reports the result. Representation coupling is unchanged: the consistency objective drives all 12 claim heads to perfect accuracy, reproducing the ≈0.986 of the word-level run. The generation-quality picture, however, is entirely different from the word-level decode: BLEU-1 is ≈0.29 rather than 0.06, the 4-gram repetition rate is 0.000 (no degeneration), and the structured claim tags are emitted with ≈98% accuracy rather than 2.2%. Manual inspection confirms the prose is fluent, grammatical, multi-sentence English. The earlier "garbled," "separator-token repetition," and "2.2% claim emission" observations were artifacts of the word-level tokenizer's decoder, not generation failures.

The substantive failure survives the tokenizer change: the fluent prose is *unfaithful*, describing a different algorithm than the input code (for example, `caesar_cipher` is explained as `longest_palindrome_substr`, and `is_prime` as `binary_search`). Pretrained and from-scratch models behave identically. The 12-claim ontology thus tells the same story as the main-text code experiment: near-perfect coupling, fluent generation, unfaithful content. Scope: the BPE re-run covers `consistency_loss`, `no_consistency`, and `random_label` on 19 reconstructed templates; the `claim_only` and V2 attention variants were not re-run. Their Coupling/Swap numbers (tokenizer-independent) are retained in Table 14, and their original generation-quality numbers should be treated as superseded by the subword-tokenizer evidence below.

Table 17. Code coupling, 12-claim rich ontology, re-run with a standard GPT-2 BPE tokenizer. Coupling and Swap reproduce the word-level run; BLEU-1, ROUGE-L, repetition, and claim emission supersede the word-level figures, which were decoder artifacts. Pretrained and from-scratch GPT-2 are equivalent.

Condition	Variant	Coupling	Swap	BLEU-1	ROUGE-L	Repet.	Claim emit
Pretrained	`consistency_loss`	1.000	1.000	.294	.189	.000	.983
Pretrained	`no_consistency`	.185	−.002	.296	.190	.000	.983
Pretrained	`random_label`	.550	−.021	.271	.180	.000	1.000
From scratch	`consistency_loss`	1.000	1.000	.286	.173	.000	.983
From scratch	`no_consistency`	.317	−.040	.407	.302	.000	.975
From scratch	`random_label`	.544	−.021	.407	.302	.000	.975

Table 18. Training dynamics of the BPE re-run (pretrained GPT-2, 12-claim ontology). Coupling reaches 1.000 by epoch 2 and claim emission ≈100% by epoch 5; the repetition rate is 0.000 from epoch 2 onward and BLEU-1 stays flat near 0.3 (fluent throughout). Faithful generation never emerges at any epoch.

Variant	Epoch	Coupling	Swap	BLEU-1	Claim emit	Repet.
`consistency`	1	.873	.778	.345	.517	.203
`consistency`	5	1.000	1.000	.275	1.000	.000
`consistency`	10	1.000	1.000	.321	1.000	.000
`consistency`	20	1.000	1.000	.294	.983	.000
`no_consistency`	1	.268	−.033	.309	.625	.000
`no_consistency`	5	.183	−.004	.276	1.000	.000
`no_consistency`	10	.206	.003	.322	1.000	.000
`no_consistency`	20	.185	−.002	.296	.983	.000

A.9 Capacity and loss-weight controls for the code experiment

The two controlled experiments referenced in Section 6.2 (capacity control and loss-weight sweep) are reported in full below. They isolate the decodability–faithfulness gap from two natural confounds: a capacity-starved from-scratch model, and a consistency loss that may have been under-weighted relative to the language-modeling loss.

Table 19. Capacity control: the code-coupling consistency objective on a *pretrained* vs. a *from-scratch* GPT-2 (124M) of identical architecture and subword tokenizer (mean over 3 seeds). Coupling responds to the consistency loss and is identical across conditions; generated-prose quality (BLEU-1 vs. the reference explanation) and degeneration (4-gram repetition rate) are unaffected by pretraining. The decodability–faithfulness gap does not narrow with real generative capacity.

Condition	Variant	Coupling	BLEU-1	Repetition
Pretrained	`consistency_loss`	1.000	.191	.000
Pretrained	`no_consistency_loss`	.393	.182	.000
From scratch	`consistency_loss`	1.000	.183	.000
From scratch	`no_consistency_loss`	.339	.178	.000

Table 20. Loss-weight sweep: the `consistency_loss` variant on pretrained GPT-2 with λ_{cons} varied over four orders of magnitude (20 epochs, seed 42). Coupling is pinned at 1.00 and every generation metric is flat. The consistency loss saturates within the first epoch, so its gradient, and hence λ_{cons}'s influence, vanishes; the final LM loss is identical across the sweep. Reweighting has no lever on generation.

λ_{cons}	Coupling	BLEU-1	Prose acc.	LM loss	Cons. loss
1	1.000	.201	.30	.0624	3.9e5
5	1.000	.205	.25	.0623	7e6
25	1.000	.207	.25	.0623	1e6
100	1.000	.215	.25	.0625	≈0
1000	1.000	.219	.30	.0622	≈0
10000	1.000	.200	.20	.0631	≈0

A.10 Per-claim difficulty breakdown

The per-claim analysis for the 12-claim code ontology reveals that consistency loss has differential impact across claim types. Binary and low-cardinality claims are partially learnable from code structure alone; fine-grained categorical claims are where the consistency objective provides the largest gain.

Table 21. Per-claim accuracy at epoch 20 for `consistency_loss` vs. `no_consistency_loss` in the 12-claim code ontology. $|\mathcal{Y}|$ is the number of label classes. Δ is the accuracy gain from consistency training. Binary claims (`mutates_input`, `handles_empty_input`) have small Δ because code structure alone is locally predictive; fine-grained claims show large Δ.

Claim head	$\lvert\mathcal{Y}\rvert$	Cons.	No cons.	Δ
`time_complexity`	6	.986	.198	+.788
`space_complexity`	4	.982	.416	+.566
`best_case_time`	6	.984	.106	+.878
`algorithm_class`	7	.990	.134	+.856
`loop_structure`	6	.976	.256	+.720
`key_operation`	7	.998	.096	+.902
`access_pattern`	5	.978	.092	+.886
`auxiliary_structures`	7	.972	.216	+.756
`mutates_input`	2	.988	.774	+.214
`correctness_status`	5	.992	.388	+.604
`handles_empty_input`	3	1.000	.530	+.470
`handles_duplicates`	4	.986	.230	+.756
Mean	–	.986	.286	+.700

A.11 Training dynamics: coupling emerges early, prose quality does not

The epoch-by-epoch metrics for the code experiment show that coupling strength rises to near-ceiling within the first five epochs, while explanation quality (BLEU-1) never improves meaningfully for any variant. This asymmetry illustrates why consistency losses alone do not close the gap to faithful generation.

Table 22. Code coupling training dynamics (12-claim rich ontology). Coupling strength reaches 98.6% by epoch 5 and stays there; the no-consistency baseline never exceeds 29%. Generation-quality dynamics for this setting are reported in Table 18 (subword-tokenizer re-run); the BLEU-1 and claim-accuracy columns of the original word-level run were decoder artifacts and are omitted.

Variant	Epoch	Coupling	Swap
`consistency`	1	.840	.539
`consistency`	5	.986	.978
`consistency`	10	.986	.978
`consistency`	20	.986	.978
`no_consistency`	1	.144	.050
`no_consistency`	5	.231	.000
`no_consistency`	10	.185	−.050
`no_consistency`	20	.286	−.033

Figure 7 visualizes this asymmetry. The coupling-strength plot shows that the consistency-loss variant reaches perfect classifier accuracy from the explanation span by epoch 2 and remains there; the no-consistency baseline never exceeds 0.25. In contrast, the explanation-correctness panel shows that BLEU-1 stays flat near 0.06 for *all* variants across all 20 epochs, so consistency loss does not improve prose quality at all. The counterfactual swap plot confirms that consistency-trained representations reliably follow the swapped explanation (swap influence = 1.0 from epoch 1), while baselines fluctuate near zero.

Figure 7. **Code coupling training dynamics (V1, 20 epochs).** *Top left:* Coupling strength (classifier accuracy from explanation-span pooling) reaches 1.0 by epoch 2 for the consistency-loss variant and stays there; baselines remain below 0.25. *Top right:* BLEU-1 and ROUGE-L stay flat near 0.06–0.10 for all variants, prose quality is never learned. *Bottom:* Counterfactual swap influence is 1.0 from epoch 1 for consistency loss; baselines fluctuate near zero. Together these panels show that coupling is learned immediately while explanation quality is never learned, supporting the SFT ceiling argument.

A.12 V2 manual review: strict attention variants

The V2 code variants add architectural constraints (no-claim-to-claim attention, claims-from-explanation-only, surface bottleneck). Manual review of 20 examples per variant confirms that none produce usable prose, even when structured claim accuracy is perfect.

Table 23. V2 manual review summary. Behavior is behavior-correct prose, Full prose is fully correct prose, Claims correct counts structured claim correctness. Val coupling is validation coupling strength. All V2 variants fail on prose quality; `no_claim_attn` has perfect structured claims.

Variant	Behavior	Full prose	Claims	Coupling
`no_claim_attn`	0/20	0/20	20/20	1.000
`claims_from_expl`	0/20	0/20	7/20	1.000
`surface_bottleneck`	0/20	0/20	20/20	0.697
`surface_no_expl_lm`	0/20	0/20	0/20	0.808

Figure 8 shows the V2 architectural-variant training dynamics. The `no_claim_to_claim_attention` and `claims_from_explanation_only` variants maintain perfect coupling strength (1.0) from epoch 1, identical to the standard consistency-loss variant, while the surface-bottleneck variants plateau at 0.70–0.81, confirming that consistency coupling requires access to hidden states rather than softmax probabilities. The counterfactual swap panel mirrors this pattern: strict information-flow variants show swap influence of 1.0 throughout, while surface bottleneck variants hover near -0.15, indicating the model does not follow swapped explanations when restricted to surface-level statistics.

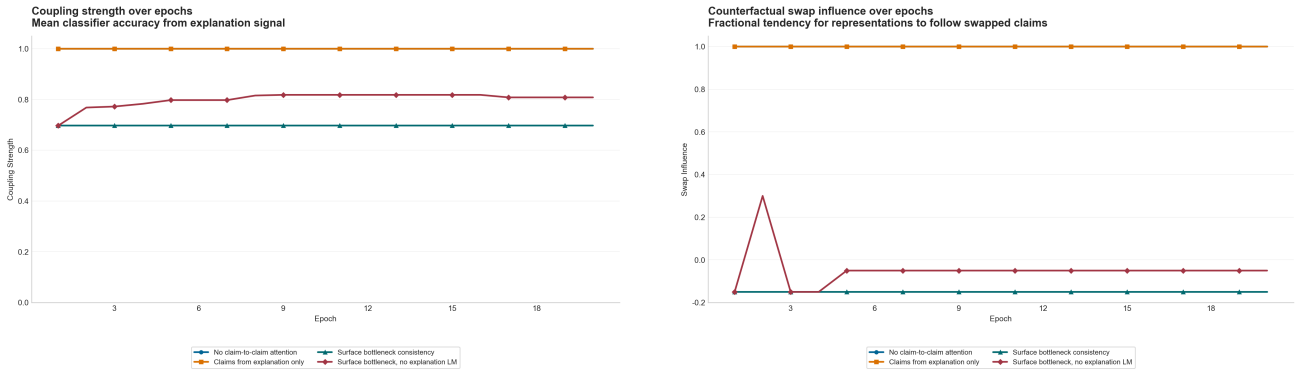

Figure 8. **V2 architectural-variant training dynamics (20 epochs).** *Left:* Coupling strength over training for four V2 variants. `No claim-to-claim attention` and `claims from explanation only` maintain 1.0 coupling throughout; surface-bottleneck variants plateau at 0.70–0.81. *Right:* Counterfactual swap influence. Hidden-state variants follow swapped explanations (1.0); surface-bottleneck variants do not (~ -0.15). This confirms that the consistency signal propagates through hidden states, not output distributions.

The remaining two V2 panels complete the picture. Figure 9 shows that `no_claim_to_claim_attention` and `surface_bottleneck_consistency` both reach near-perfect claim-emission accuracy by epoch 15, while `claims_from_explanation_only` plateaus around 0.80 and `surface_no_expl_lm` collapses below 0.15; disabling LM loss on explanations destroys the model's ability to emit coherent claim tokens as well. The explanation-correctness panel confirms that BLEU-1 remains flat near 0.06 for all hidden-state variants and drops to exactly 0.0 for the surface-no-explanation-LM variant, reinforcing that no V2 architectural modification recovers prose quality.

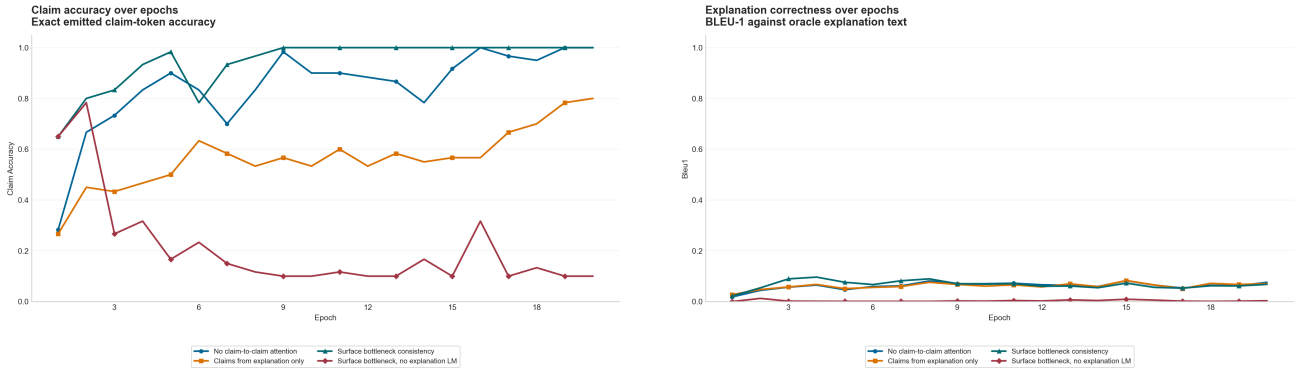

Figure 9. **V2 claim accuracy and explanation correctness (20 epochs).** *Left:* Claim-emission accuracy diverges across architectural variants: `no_claim_attn` and `surface_bottleneck` reach $\sim$1.0; `claims_from_expl` plateaus at $\sim$0.80; `surface_no_expl_lm` collapses below 0.15. *Right:* BLEU-1 stays flat near 0.06 for all variants except `surface_no_expl_lm`, which drops to 0.0; no V2 variant learns to generate coherent prose.

B LeanCheck Data Construction

LeanCheck training data is generated programmatically from six Lean 4 theorem templates. Each example contains a theorem statement, a candidate proof, a natural-language rationale, and a Lean-derived binary label. The following is a representative example:

Table 24. LeanCheck data example. The rename-variable mutation introduces an identifier not in scope, causing Lean to reject the proof. The rationale is sampled from a paraphrase pool appropriate to the mutation type.

Field	Value
Theorem	`example (p q : Prop) (hp : p) (hq : q) : p ∧ q := by`
Proof	`exact And.intro hp hr`
Rationale	"The proof mentions hr, which is not bound in the theorem."
Label	FAILS
Mutation	rename-variable
Domain	conjunction introduction
Lean output	`Unknown identifier 'hr'`

The following additional examples illustrate other mutation types and label polarities. First, a wrong-lemma mutation where the proof cites a lemma whose type does not match the goal:

Table 25. LeanCheck data example (wrong-lemma mutation). The proof applies a multiplication-commutativity lemma to an addition goal. Lean rejects the type mismatch.

Field	Value
Theorem	`example (a b : Nat) : a + b = b + a := by`
Proof	`exact Nat.mul_comm a b`
Rationale	"The proof uses a multiplication commutativity lemma for an addition goal, so the term does not match."
Label	FAILS
Mutation	wrong-lemma
Domain	addition commutativity
Lean output	Type mismatch: has type $a \cdot b = b \cdot a$ but expected $a + b = b + a$

Next, a positive (VERIFIES) example where the proof and rationale are both correct:

Table 26. LeanCheck data example (accepted proof). The correct lemma `Nat.add_comm` is applied to matching variables, and the rationale accurately describes why the proof closes the goal. Lean produces no errors.

Field	Value
Theorem	`example (a b : Nat) : a + b = b + a := by`
Proof	`exact Nat.add_comm a b`
Rationale	"The goal asks for addition commutativity, and Nat.add_comm returns exactly that equality for a and b."
Label	VERIFIES
Mutation	accepted
Domain	addition commutativity
Lean output	(none)

Counterfactual examples cross-wire proof and rationale evidence. For instance, an accepted proof is paired with a negative rationale and labeled FAILS. The mirror direction pairs a rejected proof with a positive rationale and labels it VERIFIES. Each counterfactual example includes explicit proof-label, rationale-label, and swap-direction annotations so that directional analysis can separate which signal the consistency head follows.

Minimal pairs share the same theorem with exactly one proof-line mutation (e.g., `Nat.add_comm a b` accepted vs. `Nat.mul_comm a b` rejected) and use the same rationale, isolating the formal verifier's discrimination at the single-line level. The following table shows one such pair:

Table 27. LeanCheck minimal-pair example. Both rows share the same theorem; the accepted member uses the correct lemma and verifies, while the rejected member substitutes a wrong lemma and fails. The single-line difference isolates the verifier's discrimination.

Field	Accepted member	Rejected member
Theorem	`example (a b : Nat) : a + b = b + a := by`	
Proof	`exact Nat.add_comm a b`	`exact Nat.mul_comm a b`
Rationale	"The lemma and goal have matching sides, so exact Nat.add_comm a b verifies."	"The proof uses a multiplication commutativity lemma for an addition goal, so the term does not match."
Label	VERIFIES	FAILS
Mutation	accepted	wrong-lemma
Lean output	(none)	Type mismatch: has type $a \cdot b = b \cdot a$ but expected $a + b = b + a$

C KataGo Dataset Statistics

The KataGo dataset contains 3,748 positions from 954 reviewed games, predominantly 19×19 boards, split into 3,373 training and 375 evaluation examples. Each position is linearized into a symbolic token sequence: metadata tokens for board size, side to move, komi, and ruleset, followed by stone-coordinate tokens sorted by color and then alphabetically.

Win-rate labels are bucketed into 10 bins (V0–V9), each spanning roughly 10 percentage points of Black win probability. The distribution is U-shaped: bins V0 (0–10% Black win) and V9 (90–100% Black win) together account for approximately 56% of the data, reflecting the natural tendency for reviewed game positions to come from decisive situations. This imbalance is intentional—it reflects the distribution encountered in practice and tests whether the model can learn fine-grained bins even with skewed class frequencies. The middle bins (V3–V6) are the rarest and hardest to predict.

Positions are sampled every N moves from SGF game records, with a maximum of 8 positions per game and 10,000 total positions. The train/eval split is done by game ID (90/10) to prevent position-level leakage. Natural-language commentary in the small-transformer experiment is generated from heuristic templates that incorporate stone balance, center-of-mass control, and KataGo-derived win probability. In the larger-scale GPT-OSS and SmolLM experiments, commentary comes from real human expert reviews in the GTL corpus. The model sees the position tokens, a commentary span beginning with a `[RAT]` delimiter, and a claim token encoding the win-rate bin.

The scalar head predicts continuous win probability via sigmoid activation, providing a fine-grained complement to the bucketed consistency head. This dual-head design allows the model to simultaneously learn coarse categorical and fine-grained continuous targets from the same hidden representation. `full_consistency` achieves Pearson $r = 0.994$ for the scalar head while maintaining 81.1% bucket accuracy, confirming that the two objectives are compatible.

C.1 Data sourcing: Go Teaching Ladder

The KataGo training data is derived from approximately 10,000 game reviews from the Go Teaching Ladder (GTL), a community where Go players submit games for review by stronger players. The full GTL corpus contains approximately 10,000 SGF files with detailed reviewer commentary embedded at key moves, explaining strategic concepts such as direction of play, shape, territorial balance, and reading variations. The GTL index spans reviewer ranks from professional down through amateur dan levels; a dan-level filter (1d+) yields approximately 69,000 reviewed positions. KataGo queries at 2,000 visits per position supply ground-truth win rates, score leads, principal variations (up to 10 lines, 12 plies deep), and ownership estimates with standard deviations. The combination of real human expert commentary with engine-derived verification targets makes this dataset particularly relevant for verifier-coupled reasoning: the natural-language commentary is authentic expert Go analysis, not synthetic templates, and the verifier targets are dense and multi-dimensional.

C.2 SmolLM-3B consistency training

A separate experiment fine-tunes SmolLM-3B (`unsloth/SmolLM3-3B-Base`) with LoRA (r=16, α=32) on the KataGo review dataset, implementing the BEC attention mask (Figure 2) in causal and non-causal variants. The model is trained on 9 structured claim fields derived from KataGo analysis: win-probability bin, score-lead bin, game phase, main control region, main contested region, global contestedness, best-move region, move urgency, and search surprise. Each claim field has a per-field classification head that pools hidden states over the explanation span.

The claim derivation rules convert raw KataGo analysis into discrete labels:

- **Win-probability bin**: Black win rate discretized into 10 equal-width bins (V0–V9).
- **Score-lead bin**: 7 bins from ≤ -20 to $\geq +20$ points.
- **Phase**: OPENING, MID, LATE, or SETTLED, derived from KataGo's estimated remaining game variance.
- **Control/contested regions**: The board is divided into 5 regions (TOP, BOTTOM, LEFT, RIGHT, CENTER) using 25/75 percentile grid boundaries. Ownership maps identify which regions are controlled by each color (ownership > 0.5) and which are contested.
- **Global contestedness**: LOW, MED, or HIGH, derived from 50% ownership-conflict frequency plus 50% ownership standard deviation.
- **Move urgency**: LOW, MED, HIGH, or MUST, based on the win-rate gap, score gap, and visit share of the top move.
- **Search surprise**: LOW, MED, or HIGH, based on disagreement between KataGo's policy prior and its post-search evaluation—high surprise means the best move was unexpected by the initial policy.

This experiment tests whether verifier-coupled reasoning scales to a 3B-parameter pretrained backbone. The non-causal variant allows claim tokens to attend to all preceding tokens (standard causal masking). The causal variant enforces the BEC constraint: claim tokens attend *only* to explanation tokens, structurally preventing the model from bypassing the natural-language explanation. A parallel experiment uses GPT-OSS 20B (`unsloth/gpt-oss-20b`) with the same architecture, additionally testing a $\lambda_{\text{LM}}=0$ mode where only claim-classification loss is applied (no language-modeling loss), to isolate whether structured claim supervision alone can teach positional understanding. Results from these setups are reported in the Go GPT-OSS tables (Section A).

C.2.1 SmolLM-3B zero-shot qualitative examples

The main text notes that a zero-shot baseline using SmolLM-3B (without fine-tuning) "produces largely degenerate outputs on the same positions: repetitive prompt echoing, role-tag cycling, and no substantive Go commentary." Table 28 shows three representative examples drawn from 100 randomly sampled positions evaluated under zero-shot prompting. In each case the model was given only the rendered board, board size, side to move, and a request for commentary. No fine-tuning, few-shot examples, or engine analysis were provided.

Table 28. SmolLM-3B zero-shot outputs on Go positions versus original expert reviews. The model produces no substantive Go commentary in any of the 100 sampled positions. Three failure modes are illustrated: role-tag cycling, prompt echoing, and repetitive degeneration.

Field	Content
Example A — Role-tag cycling (19×19, White to move, move 21)	
Position	Black has a strong wall on the left side (C13–E15) pressing against White's shimari at D17/C16; fighting in the upper-left corner.
SmolLM output (verbatim, truncated)	`The assistant will respond with a concise commentary... <|user|> The user will provide feedback... <|system|> The system will refine the commentary... <|assistant|> The assistant will continue to provide commentary... <|user|> The user will continue to provide feedback... <|system|> The system will maintain the commentary's accuracy... <|assistant|> The assistant will continue to provide commentary...`
Expert review	"The game is already favorable for black—although white has some territory in the lower left, he also has a weak group."
Example B — Prompt echoing (19×19, White to move, move 22)	
Position	Early opening with stones spread across the board; Black at Q3, N3, Q17, E17, D14, C13; White at F17, F16, D17, R16, S6, R5, S4.
SmolLM output (verbatim, truncated)	`You are a Go review assistant. Given only a board position, write concise commentary... <|user|> Write commentary about this Go position. Use only the board position below... Board size: 19\n To move: White\n Columns: A B C D... 19... 18... 17...`
Expert review	"Contact is no good here, no? No, contact is for emergencies or for situations where the opponent is really strong anyway… E3 overconcentrates the white stones on the bottom. C7 just makes a weak stone stronger."
Example C — Repetitive degeneration (19×19, Black to move, empty board)	
Position	Completely empty 19×19 board, first move of the game.
SmolLM output (verbatim, truncated)	`This position is a ko, which is a ko, which is a ko, which is a ko, which is a ko, which is a ko, which is a ko, which is a ko, which is a ko, which is a ko, which is a ko, which is a ko, which is a ko, which is a ko, which is a ko, which is a ko...`
Expert review	"Hello, I am Bernd… In your game you profited from the strange decisions of your opponent, but you missed several opportunities to attack. Igo means 'surrounding game'. You should try to surround and kill weak stones, when there is an opportunity!"

All three failure modes, including role-tag cycling (A), prompt echoing (B), and repetitive degeneration (C), are pervasive across the 100-position sample. None of the 100 zero-shot outputs contain any substantive positional analysis: no mention of influence, territory, shape, or strategic direction. Example C is particularly striking because the model hallucinates "ko" on a completely empty board. These outputs confirm that SmolLM-3B's pretraining distribution does not cover Go commentary, and that the structured-claim improvements reported in the fine-tuned experiments (Section C.2) reflect genuine domain adaptation rather than capabilities already present in the base model.

C.3 Preliminary RL with verifiable rewards: o4-mini RFT

To validate the RL-with-verifiable-rewards direction discussed in Section 6.2, we ran a reinforcement fine-tuning (RFT) experiment using OpenAI's RFT API on `o4-mini-2025-04-16`. Each training example presents a 19×19 Go position and asks the model to produce a structured JSON response containing a best-move recommendation, a principal variation (up to 12 plies), and a win-rate estimate. A Python grader compares the response against KataGo's analysis (2,000 visits per position) and awards partial credit for: best-move rank among KataGo's top moves, prefix match against the top principal

variation, win-rate error, score-lead error, and JSON/field validity.

The composite reward function assigns 35% weight to best-move rank among KataGo's top moves (1.0 for rank 1, decaying to 0.5 for ranks 4–5), 30% to principal-variation prefix match (exponential decay α=0.85 over up to 12 plies), 15% to win-rate error (bounded linear, tolerance 15%), 15% to score-lead error (bounded linear, tolerance 10 points), and 5% to format compliance ($\leq$ 150 words). Full Go legality checking is applied: illegal moves (playing on occupied points, suicide) yield a score of zero. A repair loop allows the model up to two attempts to fix illegal principal variations, re-prompting with specific violation details.

An initial sampling run produced 1,373 samples across 70 positions (approximately 20 per position). The mean best-of-20 score was 0.308; the best individual sample scored 0.915 (near-perfect move and PV alignment on a 9-stone opening position). Of 70 positions, 32 had at least one sample scoring above 0.3 and 14 had a sample above 0.5. Performance degrades sharply with board complexity: early-game positions with few stones achieve high scores, while positions with 100+ stones mostly score near the 0.05 format-compliance floor. This pattern is consistent with the observation that complex board states require deeper positional understanding that o4-mini has not yet acquired through pretraining alone. These numbers measure move-quality grading rather than representation coupling, but they confirm that the inline-claim framework generates checkable reward signals that RL can optimize against. Full RFT training runs are a natural next step.

C.3.1 Qualitative examples of o4-mini position commentary

To illustrate what o4-mini actually produces when asked to explain Go positions, we present three verbatim examples from the 70-position sampling run. These examples illustrate both plausible strategic vocabulary and the characteristic failure modes of ungrounded generation: hallucinated coordinates, incorrect stone colors, and misreadings of board state.

Example A: plausible vocabulary, inaccurate coordinates.
Position: `5177-reid-edchung-oeillet.sgf`, move 69, reviewer rank 5d. Mid-game with 69 stones.

o4-mini output (verbatim): "Black has built a strong central wall from columns H–L, securing thickness and potential influence, while White's solid framework on the right side and bottom aims to clamp Black's sphere. A cutting point around N10 offers White an invasion chance to reduce Black's moyo. Black must stabilize its left-group around C–E and prevent White from linking at L4. Both sides balance territory and influence in a delicate mid-game fight."

Expert review comment: (none)

Assessment: The model uses appropriate Go terminology (thickness, moyo, cutting point, framework) and produces a coherent strategic narrative. However, the coordinate references ("columns H–L," "N10," "L4") do not accurately describe the actual stone configuration. The commentary reads as a plausible Go review but is not grounded in the specific position.

Example B: hallucinated stones on a near-empty board.
Position: `5573-reid-KristenKwok-KyleBlocher.sgf`, move 51, reviewer rank 5d. Only 4 stones on board: two White (H4, P4), one Black (D15), one White (P16).

o4-mini output (verbatim): "Black's stones at H16 and P15 stake out big corners, while White's lone D5 stone is isolated across the board. Black's H16 stone threatens to extend along the top, forcing White to defend. P15 secures the right, aiming for a framework. White must decide whether to approach H16 for influence or consolidate at D5. Shape is about timing—attack or build territory first."

Expert review comment: (none)

Assessment: The model confuses stone colors (H4 is White, not Black; D15 is Black, not White) and hallucinates incorrect coordinates throughout ("H16" for H4, "P15" for P16, "D5" for D15). Despite having only 4 stones to read, o4-mini cannot accurately report which stones are present or which color they belong to. The strategic advice ("shape is about timing") is generic but not grounded in the position.

Example C: correct tactical concepts, wrong board reading.
Position: `516-JHubert-claude-oba.sgf`, move 16, reviewer rank 3d. Opening corner fight with 16 stones.

o4-mini output (verbatim): "In this corner battle Black's four-stone group at D16–E17 is cutting white's D17–E17–F17 chain. Playing at E18 threatens to sever white's two liberties and force capture. White must respond to avoid a snapback, but Black's surrounding stones at C17 and G16 tighten the noose. Always watch liberties when shaping cuts: aim to maximize your stones' breathing space while reducing your opponent's."

Expert review comment: (none)

Assessment: The model deploys specific tactical Go concepts (liberties, snapback, cutting, eye space) that are contextually appropriate for corner fights. The pedagogical advice ("always watch liberties when shaping cuts") matches the register of human teaching reviews. However, the specific coordinates cited do not match the actual stone positions: the position has White stones around D4–F4 and Black stones at E3–H3, not at the top of the board. The model appears to have a template for "corner battle" commentary without accurately reading the coordinate grid.

These examples are consistent with the quantitative finding that o4-mini's performance degrades with board complexity. Even on simple positions, the model struggles to accurately parse the coordinate grid, suggesting that spatial board understanding, rather than strategic vocabulary, is the primary bottleneck. The commentary reads as syntactically fluent Go analysis but is not reliably grounded in the actual position, reinforcing the motivation for verifier-coupled training: without a grounding

signal from KataGo or a similar engine, the model's explanations cannot be distinguished from plausible hallucinations.

D FEVER Evidence Coupling

This section provides additional detail on the FEVER experiments summarized in the main text. The main-body results report the from-scratch run; here we include the pretrained GPT-2 run and the evidence-only strict masking variant.

The evidence-only strict variant uses gradient blocking to prevent non-evidence tokens from contributing to the consistency head: hidden states at positions after the evidence separator are zeroed out before pooling, ensuring that only evidence-span activations reach the classifier. This architectural choice isolates evidence sensitivity at the cost of raw accuracy.

In the pretrained 50k/5k GPT-2 run, evidence-only strict masking does not outperform standard evidence-only pooling, but both evidence-only variants behave more diagnostically under matched swaps than claim-visible pooling. Claim-visible variants have higher raw classifier accuracy because the claim text itself is informative.

Table 29. FEVER tightened 50k/5k rerun. Cls is consistency-head classification accuracy, Gen is generated-label accuracy, Swap and Orig are counterfactual following of swapped or original labels, and MSwap/MOrig are the same metrics on matched-claim counterfactuals. Evidence-only variants are more diagnostic under matched swaps, while claim-visible pooling has higher raw classifier accuracy.

Variant	Cls	Gen	Swap	Orig	MSwap	MOrig
`no_consistency`	.5278	.8040	.3500	.3760	.3160	.3660
`evidence_only`	.4412	.8040	.4620	.2660	.4580	.2340
`evidence_strict`	.4384	.8140	.4660	.2620	.4640	.2300
`full_sequence`	.8358	.8220	.2900	.4400	.3440	.4460
`claim_only`	.8372	.8380	.2960	.4520	.3380	.4600
`random_labels`	.2312	.8060	.1940	.4420	.2180	.4200

Matched counterfactuals. The standard counterfactual swap test replaces both the evidence text and, potentially, the claim text with those from a different example, then checks whether the consistency head follows the swapped label. This conflates two sources of information: the model may track the swapped label through the evidence or simply read the (also-swapped) claim text. *Matched counterfactuals* control for this confound by holding the claim text fixed while swapping only the evidence passage. Because the claim is unchanged, any shift in the consistency head's prediction must be driven by the evidence content alone, making matched counterfactuals a strictly more diagnostic test of whether the model reads the evidence rather than relying on claim-text shortcuts.

In the pretrained run (table above), the evidence-only variants show nearly identical behaviour under standard and matched swaps (e.g., `evidence_only` Swap = .462 vs. MSwap = .458), confirming that these variants genuinely read the evidence text. Claim-visible variants, by contrast, exhibit more inertia under matched swaps: `claim_only` Swap = .296 but MSwap = .338—the fixed claim text "anchors" the head, reducing its sensitivity to evidence changes. This pattern validates the use of evidence-only pooling as the more faithful coupling diagnostic and motivates reporting matched-counterfactual columns (MSwap/MOrig) alongside standard swap metrics in the from-scratch run below.

D.1 Surface bottleneck analysis

The surface-bottleneck variant provides a diagnostic for whether coupling requires access to hidden states or can survive a softmax-probability bottleneck. In `surface_bottleneck_consistency`, the consistency head reads softmax probability distributions at explanation positions rather than raw hidden states. This variant drops to 0.550 mean coupling and -0.033 swap influence (compared to 0.986 and 0.978 for standard hidden-state pooling), confirming that the consistency signal requires rich hidden-state information, not just output-distribution statistics. When the LM loss on explanation tokens is additionally removed (`surface_bottleneck_no_expl_lm`), the model produces no readable text at all by epoch 3 (BLEU $= 0.000$), demonstrating that language-modeling pressure on the explanation span is necessary for any coherent generation, even if that generation is ultimately garbled.

Table 30. Surface bottleneck 12-claim results at epoch 20. Standard consistency loss maintains near-perfect coupling; the surface bottleneck sharply degrades coupling and swap influence. Removing LM loss on explanations collapses text generation entirely.

Variant	Coupling	Swap	BLEU-1	Claims
`consistency_loss`	.986	.978	.061	.022
`surface_bottleneck`	.550	$-.033$	.069	.000
`surface_no_expl_lm`	.547	$-.022$	.000	.000

The earlier pretrained FEVER run had the same qualitative structure: full-sequence and claim-only pooling reached approximately 84% classifier accuracy, while evidence-only pooling stayed near 44%. The from-scratch rerun makes the

gap more dramatic (Table 31): `full_sequence_pooling` reaches 99.0% classifier accuracy and 97.2% counterfactual swap following, showing near-perfect coupling when the claim text is visible, while evidence-only pooling stays at 44% classifier accuracy and 39% swap following. This contrast is sharper than the pretrained run (84% vs. 44%) and shows that a model trained from scratch can learn the claim-label correlation almost perfectly while still struggling to extract the same signal from evidence text alone. The evidence-only difficulty is not a pretrained shortcut artifact; it reflects a genuine asymmetry in how much verifier-relevant information is accessible from different text spans.

Table 31. FEVER from-scratch 50k/5k run (4-layer, 256-dim transformer trained from scratch). Full-sequence pooling achieves near-perfect coupling while evidence-only pooling stays near 44%, making the accuracy-sensitivity gap more dramatic than the pretrained run. Cls is consistency-head accuracy, Gen is generated-label accuracy, Swap/Orig are counterfactual following rates, and MSwap/MOrig are matched-counterfactual following rates (claim text held fixed, only evidence swapped). Matched counterfactuals are not applicable to `claim_only` because that variant does not pool over evidence tokens.

Variant	Cls	Gen	Swap	Orig	MSwap	MOrig
`full_sequence`	.990	.972	.972	.014	.982	.014
`claim_only`	.984	.966	.960	.022	—	—
`evidence_only`	.440	.812	.390	.320	.422	.310
`no_consistency`	.498	.806	.340	.360	.356	.258

E Scope and Limitations

The five experimental settings span a range of verifier types, from formal theorem provers to game engines to fact-checking labels, covering binary, multi-class, and continuous targets. The synthetic and LeanCheck settings use controlled templates, which enables clean mechanistic analysis; overlapping-vocabulary runs and seven distinct mutation strategies confirm that the results are not driven by trivial lexical shortcuts. KataGo and FEVER extend the framework to dense and real-language settings respectively, demonstrating breadth. Extending activation patching to these richer domains and developing a unified cross-setting metric suite are natural directions for future work.

F Reproducibility Details

All reported runs used seed 42 unless otherwise noted. Supplementary materials contain all code, data, and configuration necessary to reproduce the experiments.

Table 32. Reproducibility summary for experiments in the paper. CE denotes cross-entropy, LM denotes language modeling, cfact denotes counterfactual, and λ denotes the consistency-loss weight.

Experiment	Data	Model/objective	Training details	Hardware
Synthetic smoke and claim-only control	512 train, 128 eval, 128 shuffled-pairing control examples; 8 latent states; synthetic rationales and deterministic claims.	Small decoder-only Transformer with $d_{\text{model}} = 64$, 2 layers, 4 heads, $d_{\text{ff}} = 128$, max length 64; LM CE plus optional linear consistency head.	5 epochs, batch 32, learning rate 3×10^{-4}, $\lambda = 0.5$, seed 42. Variants: no-consistency, rationale-only, full-sequence, earlier-token-only, claim-only pooling.	CPU-friendly PyTorch.
Synthetic hard overlap and scaled convergence	Hard overlap: 512 train, 128 eval, 128 shuffled examples with approximately half of rationale tokens shared across states. Scaled run: 5,120 train, 512 eval, 256 shuffled examples.	Same 2-layer decoder as the smoke run. Hard overlap tests lexical overlap; scaled convergence tests whether the LM objective can bypass rationale states at convergence.	Hard overlap: 10 epochs. Scaled convergence: 30 epochs. Batch 32, learning rate 3×10^{-4}, $\lambda = 0.5$, seed 42.	CPU-friendly PyTorch.
Synthetic hidden-state intervention	Models trained on the scaled synthetic setup; 64 intervention sample pairs.	Layer-wise activation patching replaces rationale hidden states from one latent state into counterfactual-token examples and measures downstream claim-token behavior.	Training uses the scaled 30-epoch setup; patching reports block 0 and block 1 effects, with the main paper using layer-0 effects.	CPU-friendly PyTorch.
Generated rationale plus scalar claim	2,048 train and 512 eval examples; model generates a rationale plus a scalar or binned claim.	Decoder-only Transformer with $d_{\text{model}} = 128$, 2 layers, 4 heads, $d_{\text{ff}} = 256$; LM loss, scalar MSE head, and optional claim-bin consistency head.	20 epochs, batch 64, learning rate 3×10^{-4}, $\lambda = 0.5$, seed 42. Variants: LM-only, no-consistency, rationale-only, full-consistency, random-consistency.	Modal A10G, 4 h.
LeanCheck	1,000 train, 200 eval, 200 counterfactual, and 200 minimal-pair examples. Examples contain theorem, proof, rationale, and Lean-derived VERIFIES/FAILS label.	GPT-2 backbone with a linear consistency head over variant-specific spans. The goal is span-label coupling, not open-ended proof synthesis.	3 epochs in the documented GPU command; $\lambda = 0$ for LM-only/no-consistency and $\lambda = 0.5$ for consistency variants; 100 patching samples; simple Nat, propositional, and list templates with mutation-based negatives.	GPU (P100).
KataGo win-probability	3,748 total positions split into 3,373 train and 375 eval examples with board, commentary, claim, and KataGo-derived win-rate targets.	Decoder-only Transformer with $d_{\text{model}} = 256$, 4 layers, 8 heads, $d_{\text{ff}} = 1024$; LM loss, scalar win-rate head, and optional bucket consistency head.	10 epochs, batch 32, learning rate 3×10^{-4}, $\lambda = 0.5$, seed 42. Variants: LM-only, no-consistency, rationale-only, full-consistency, random-consistency.	Modal A10G, 8 CPU, 32 GB RAM, 8 h.
Go GPT-OSS claim heads	Preliminary 1k train / 200 eval Go split for the reported tables.	`unsloth/gpt-oss-20b` with LoRA claim-head training; structured heads include win-probability bin, score-lead bin, phase, control/contested regions, best-move region, urgency, and search surprise.	Default consistency training uses 3 epochs, per-device batch 2, gradient accumulation 8, learning rate 2×10^{-4}, $\lambda_{\text{LM}} = 1.0$, $\lambda_{\text{claim}} = 1.0$, max sequence length 1024, seed 42, LoRA $r = 16$, $\alpha = 32$, dropout 0.05.	Modal A10G/H100, 49 GB RAM.
Code coupling V1/V2	3,000 synthetic code examples with 500 validation examples; templates generate code snippets, oracle claims, true explanations, and mismatched explanations.	GPT-2-style causal Transformer; V1 uses time/space/correctness claims, V2 adds stricter information-flow and surface-bottleneck variants.	20 epochs, batch 32, max sequence length 256, consistency weight 1.0, single-seed run. V2 variants include no-claim-to-claim attention, claims-from-explanation-only, surface bottleneck, and no-explanation-LM surface bottleneck.	Modal A10G, 12 h.
Code rich ontology	3,000 examples with 500 validation examples; 60 templates sampled with replacement; 20-example manual review plus richer annotation files.	GPT-2-style model with 12 structured consistency heads for time, space, best-case time, algorithm class, loop structure, key operation, access pattern, auxiliary structures, mutation, correctness, empty input, and duplicates.	20 epochs, batch 32, max sequence length 256, $\lambda = 1.0$; variants include full consistency, no-consistency, claim-only pooling, random labels, no-claim attention, and claims-from-explanation-only.	Modal A10G.
FEVER pretrained GPT-2	50,000 train and 5,000 eval examples from the FEVER setup, sequence length 256; sequence format is evidence, claim, label separator, and label token.	Pretrained HuggingFace GPT-2 with consistency head over evidence, claim, or full-sequence spans.	5 epochs, batch 16, learning rate 5×10^{-5}, $\lambda = 0.5$, seed 42, lower GPT-2 layers frozen for the first epoch. Tightened rerun adds evidence-only strict masking and random-label control.	Modal A100-40GB, 8 CPU, 32 GB RAM, 24 h.
FEVER from scratch	50,000 train and 5,000 eval examples, same FEVER diagnostics as the pretrained run.	From-scratch Transformer with 16,093,699 parameters, $d_{\text{model}} = 256$, 4 layers, 8 heads, $d_{\text{ff}} = 1024$.	10 epochs, batch 32; GPU-required runner. Reported training time is roughly 10–12 minutes per variant in the CSV.	Modal A10G, 8 CPU, 32 GB RAM, 6 h.