Observable but Not Steerable: Restoring Oversight Control in Latent Reasoning Models

ML Lab Research Proposal (Compute Grant Request, AMD Instinct MI300X)

Authors: Maruthi Vemula, Praneeth Gajula

Date: 3 July 2026

1. Problem Statement

Latent reasoning models do their thinking in continuous internal states instead of writing out chain-of-thought text. They are efficient, but you cannot see what they are doing, which is a real problem if you want to oversee them. We ran a few experiments and found something worse than plain opacity. These models are observable but not steerable. You can decode the internal state, and you can even edit what the model's own readout shows, but the model just ignores your edit and reaches its original answer about 91% of the time. So if a safety system tries to correct a latent reasoner by intervening on its visible state, the correction quietly does nothing. We want to pin down why this happens and fix it in the architecture, so that the part you can see is actually the part that drives the computation.

2. Proposed Approach

We work with depth-recurrent (looped) transformers, which think by running a shared block over a hidden state again and again. To measure whether you can steer one, we edit its state so the readout decodes some counterfactual value, run the rest of the loops, and check whether the final answer changes to match. Our fix is to make the readout carry weight in the computation. At every loop we push the carried state into the subspace the readout can actually see (Arm B), or we route the recurrence through a small auditing channel (Arm A). Either way, what an overseer sees becomes what the model has to use. We went this route instead of post-hoc interpretability, which can tell you what the model did but cannot make it do anything else, and instead of the scale-normalization trick from prior work, which in our tests did not help at all. We already have this working at small scale (0.55M parameters), shown in the table below. Our fix takes steerability from 0.09 to 0.40 with no drop in accuracy, while the prior scale trick sits at 0.08. What we need the compute for is to check whether this holds up, and gets bigger, at real scale.

Preliminary results (0.55M-parameter looped reasoner, permutation-iteration task):

Variant	Accuracy	Edit succeeds	Answer follows edit
Baseline (standard looped reasoner)	1.00	yes (1.00)	0.09
Scale-removal (prior work's fix)	1.00	yes	0.08
Readout made causally load-bearing (ours)	**1.00**	**yes**	**0.40**

3. Dataset

Most of our data is synthetic, generated on the fly, and comes with the exact right answer at every intermediate step. Think iterated permutations and pointer-chasing, modular arithmetic, and step-by-step logic. Generating it ourselves gives us unlimited data, no contamination, and a difficulty knob, and it means we always know the correct intermediate value, which is exactly what we need to test interventions. We also throw in a few small real reasoning benchmarks so the results are not just about toy tasks (GSM8K, plus the ProntoQA and ProsQA logic sets that earlier latent reasoning papers used), roughly 10k examples each. There is basically no preprocessing. The synthetic tasks are generated live, and the real ones only need standard tokenization.

4. Evaluation Metrics

The main number is intervention faithfulness: out of all the counterfactual edits we make, how often does the final answer actually follow. Alongside that we plot the steerability versus accuracy tradeoff. The supporting numbers are task accuracy (can the model still do the task), edit-success rate (did we actually change what the readout shows, as a sanity check), how well a probe can read the true intermediate state, our two blind-spot measures, and how all of this moves as we scale the model (100M to 1B) and the recurrence depth K. We will call it a win if the observable-but-not-steerable gap shows up and holds or grows with scale, and our fix closes it without paying too much in accuracy.

5. Compute Requirements

Our models are small to medium (100M to 1B parameters), so one MI300X and its 192 GB of memory is plenty. The big memory actually helps here, because we backprop through a lot of loop iterations (up to K=32), and with that much HBM we can do it without gradient checkpointing and still run large batches. The plan is roughly 4 architecture variants times 3 task families times 4 model sizes times 4 seeds, so around 200 training runs at maybe 0.5 to 4 GPU-hours each, plus the intervention evals. Call it 300 to 500 GPU-hours total. On an 8-GPU node that is about 2 to 4 days of wall-clock, which fits in two weeks with plenty of room to iterate. On the software side we just need a ROCm build of PyTorch 2.x (our code already picks its device automatically, so switching to ROCm is basically a one-line change) and NumPy. No custom kernels, no multi-node training, nothing exotic.

6. Expected Deliverables

After two weeks we will have four things. First, the full study of the observable-but-not-steerable effect across model size, recurrence depth, and task type, with the steerability versus accuracy frontier for both of our fixes. Second, the architecture itself (Arm A and Arm B) plus a cleaner on-manifold way to do the interventions. Third, publication-quality figures and a paper draft aimed at AAAI-2027 (abstract due 21 July 2026). Fourth, an open-source release of the code and the synthetic task suite.

7. Risk & Mitigation

The thing most likely to go sideways is that the steerability gap gets smaller as we scale, or that our fix starts costing accuracy at scale. Either way we still have a paper. If the gap shrinks, that is genuinely good news for oversight and worth reporting, and if the fix costs accuracy, that tradeoff curve is the main result. We also run two independent fixes and report the whole frontier instead of betting on one number. A few smaller risks. The genuinely hard tasks might be tough to train, so we will use curricula and lean on the bigger models (again, the memory helps), and we can fall back to the controlled tasks that already show the effect. Someone might have published a piece of this, so we do a focused prior-art check before spending the big compute. And ROCm can be finicky, so we keep to standard ops and smoke-test on a single GPU before kicking off the full sweep.