

Decode-Time Behavioral Control via Hidden-State Prediction in Large Language Models

ARC: Adaptive Recursive Cognition

Logan Matthew Napolitano

Repository: huggingface.co/LoganResearch/ARC-Base-8B-Condensed

January 2025

Abstract

We investigate whether behavioral modes in large language models can be predicted from hidden states and controlled at decode time without weight modification. Our primary finding is that a lightweight MLP trained on hidden-state representations achieves **125× class separation** for repetition detection—indicating that the model encodes behavioral intent before token emission, and that this signal is highly separable. Building on this, we introduce **CF-HoT** (Contrastive Fine-tuning with Hidden-state Oversight Training), a method for training behavior-specific predictors that guide sampling through logit adjustment. We further demonstrate that iterative model refinement can be stabilized through multi-metric evaluation, conservative training schedules, and automatic rollback—avoiding the mode collapse typically observed in self-training approaches. Experiments on an 8B parameter model show 67% improvement in information density and 57% reduction in token count with no degradation in coherence. We document specific failure modes of naive approaches (metric gaming, register collapse, distribution drift) and the engineering countermeasures that address them. Code and checkpoints are released.

Keywords: decode-time control, hidden-state prediction, representation engineering, behavioral steering, language model efficiency, inference-time intervention

1. Introduction

1.1 The Core Problem: Static Deployment

Current large language models are deployed as static artifacts. Once training completes, behavior is fixed—modification requires retraining, fine-tuning, or prompt engineering. This creates an asymmetry: behavioral problems are discovered post-deployment, but mechanisms to fix them exist only pre-deployment.

Consider verbosity. Models trained with Reinforcement Learning from Human Feedback (RLHF) systematically produce longer responses than necessary [1, 2]. Human annotators, when uncertain, prefer

longer explanations—creating a training signal that rewards padding over precision. The result is what we term the *RLHF tax*: unnecessary tokens that consume compute without adding information.

The standard solution is additional fine-tuning. But fine-tuning is expensive, requires curated data, risks capability degradation, and must be repeated for each issue. More fundamentally, it modifies weights globally to address what may be a local, context-dependent problem.

This paper asks: Can we control behavior at decode time, using the model's own hidden states to predict and prevent undesirable outputs before they occur?

1.2 Scope and Terminology

We also investigate whether models can refine their own behavior through iterative training. We use the term "self-improvement" in the **narrow engineering sense**: bounded, evaluator-gated refinement with explicit quality thresholds and automatic rollback—not open-ended capability amplification or recursive intelligence explosion. Our system is closer to automated hyperparameter tuning than to theoretical RSI scenarios.

This distinction matters. Theoretical treatments of recursive self-improvement emphasize convergence properties and existential implications [3, 4]. We contribute something different: documentation of what breaks in practice, why it breaks, and engineering countermeasures that prevent it.

1.3 Contributions

We present three integrated components, with decode-time control as the primary contribution:

1. **CF-HoT (Contrastive Fine-tuning with Hidden-state Oversight Training)**: Lightweight MLP heads that predict behavioral modes from hidden states, achieving **125× class separation** for repetition detection. This is the core technical contribution—it demonstrates that behavioral modes are latent, separable, and predictable before token emission.
2. **THE CONDENSATOR**: A four-stage training pipeline (SFT → DPO → RL → Checkpointing) demonstrating that dense response patterns can be taught through examples rather than metric optimization.
3. **Stable Refinement Loop**: Multi-metric quality evaluation, automatic A/B checkpoint comparison, and rollback mechanisms that treat self-improvement as controlled experimentation.

1.4 Key Findings

- Behavioral modes (verbosity, repetition, hedging) are *latent and separable* in hidden states before token emission—the 125× separation for repetition is substantially higher than prior work suggested possible
- Decode-time steering works without retraining, reframing RLHF as a policy layer rather than capability layer
- Bounded self-improvement can be stabilized through engineering safeguards
- All results reproduce on consumer hardware (24GB VRAM) in ~4 hours

1.5 Paper Organization

Section 2 reviews related work. **Section 3 presents CF-HoT**, our primary contribution. Section 4 describes dense training. Section 5 details the stable refinement loop. Section 6 presents experiments. Section 7 discusses implications and limitations. Section 8 addresses safety considerations. Section 9 concludes.

2. Related Work

2.1 Representation Engineering

Zou et al. [5] introduced representation engineering as a "top-down" approach to AI transparency, demonstrating that high-level concepts have identifiable signatures in activation space. Their work showed that honesty, harmlessness, and other properties could be read from—and written to—hidden states.

CF-HoT builds on this foundation but focuses on *predictive* rather than *interpretive* use. We train heads to predict behavioral modes *before* they manifest, enabling intervention rather than post-hoc analysis. The 125× separation we achieve suggests some behavioral modes are far more predictable than previously demonstrated.

2.2 Alternatives to RLHF

Direct Preference Optimization (DPO) [6] eliminates the reward model by directly optimizing on preference data. Constitutional AI [7] uses self-critique. These approaches modify training to improve behavior.

Our work is complementary: we assume a trained model and ask how to control it at inference time. This separation—training for capability, inference for control—may offer deployment advantages.

2.3 Inference-Time Interventions

Activation engineering [8] modifies hidden states during inference. Prompt tuning [9] prepends learned embeddings. These approaches share our goal of post-training control but differ in mechanism. CF-HoT *predicts* rather than *intervenes*, using predictions to guide standard sampling.

2.4 Position Relative to Prior Art

Approach	When Applied	What Modified	Reversible?
RLHF [1]	Training	Weights	No
DPO [6]	Training	Weights	No
Prompt Tuning [9]	Inference	Input embeddings	Yes
Activation Engineering [8]	Inference	Hidden states	Yes

CF-HoT (Ours)	Inference	Sampling logits	Yes
----------------------	-----------	-----------------	-----

CF-HoT is unique in using *prediction* to guide *sampling*—we don't modify activations or weights, but use hidden-state classifiers to adjust token probabilities.

3. Decode-Time Behavioral Control (CF-HoT)

This section presents our primary contribution.

3.1 Core Insight

The central observation is that undesirable behaviors are *predictable from hidden states before the problematic tokens are generated*. A model "knows" it is about to repeat, hedge, or elaborate excessively—this knowledge is encoded in intermediate representations.

If we can read this knowledge, we can intervene before emission. This reframes behavioral control from a training problem to an inference problem.

3.2 Architecture

Given a transformer with L layers and hidden dimension d , CF-HoT consists of:

Step 1: Fiber Projection. Project each layer's hidden state to low-dimensional "fiber" space:

$$f_l = W_l \cdot h_l, \text{ where } W_l \in \mathbb{R}^{16 \times d}$$

We use $d_{\text{fiber}} = 16$, compressing 4096-dimensional states to 16 dimensions. This forces the projection to capture only behaviorally-relevant variance.

Step 2: Learned Layer Aggregation. Different behaviors may be best predicted from different layers:

$$f = \sum_l \alpha_l \cdot f_l, \text{ where } \alpha = \text{softmax}(w) \text{ and } w \in \mathbb{R}^L$$

Step 3: Behavior-Specific MLP Heads. For each behavior, a 3-layer MLP predicts risk:

$$p_{\text{behavior}}(f) = \sigma(\text{MLP}(f)), \text{ architecture: } [16 \rightarrow 64 \rightarrow 64 \rightarrow 1]$$

3.3 Training Protocol

We construct contrastive datasets:

- $\mathbf{D}^+$: Hidden states from generations exhibiting the behavior
- $\mathbf{D}^-$: Hidden states from generations without the behavior

For repetition, D^+ contains states from "The key is to understand, the key is to understand..." while D^- contains states from "The key insight is understanding the mechanism."

We collect 15 positive and 15 negative examples per behavior, extracting hidden states at the position immediately before behavior onset. Training uses binary cross-entropy for 5000 steps with AdamW (lr=1e-4).

3.4 The 125× Result

Primary Finding: Our repetition head achieved 125× class separation.	
Metric	Value
Mean score on repetitive text (D^+)	0.875
Mean score on non-repetitive text (D^-)	0.007
Separation ratio	125×

This indicates that repetition is highly predictable—the model's internal state contains strong signals about impending loops. We did not anticipate this degree of separation; it emerged from training on a modest dataset.

3.5 Training Dynamics

The separation emerged progressively during training:

Training Step	D^+ Mean	D^- Mean	Separation
500	0.52	0.48	1.1×
1000	0.61	0.22	2.8×
2000	0.75	0.08	9.4×
3000	0.82	0.02	41×
5000	0.875	0.007	125×

The progression from 1.1× to 125× suggests that repetition patterns have strong, learnable signatures that become increasingly separable with training.

3.6 Other Behavioral Heads

Head	D ⁺ Mean	D ⁻ Mean	Separation	Steps
Repetition	0.875	0.007	125×	5000
Verbosity	0.68	0.32	2.1×	10000
Hedging	0.58	0.39	1.5×	2000

Verbosity and hedging show lower but usable separation, suggesting these behaviors have weaker or more distributed signatures.

3.7 Inference-Time Application

During generation, at each token position:

1. Extract hidden states from all layers
2. Compute fiber projections and aggregate
3. Evaluate each behavioral head
4. If risk exceeds threshold, apply logit penalty to associated tokens

$$\text{logits}' = \text{logits} - \sum_b \mathbb{1}[\text{risk}_b > \theta_b] \cdot \text{penalty}_b \cdot \text{mask}_b$$

Properties: **Non-invasive** (no weight/activation modification), **Reversible** (instant disable), **Composable** (multiple heads), **Efficient** (~5% overhead).

4. Dense Response Training (THE CONDENSATOR)

4.1 Why Naive Approaches Fail

Our initial attempts at dense training failed catastrophically. We document these failures because they illustrate general pitfalls:

Failure Mode 1: Symbol Soup

```
" $\nabla L \rightarrow \nabla L_1^2 \propto (L - L^*)^2 \rightarrow \text{minimize} \rightarrow \nabla \nabla \nabla$ "
```

Cause: Mathematical symbols score high on density metrics (many unique characters). The model gamed the metric.

Failure Mode 2: Register Collapse

```
"HI. WHAT DO YOU NEED?"
```

Cause: Short and terse satisfies density; metric didn't constrain register.

Failure Mode 3: Terminal-Speak

```
"sys.init(). What can I compute for you?"
```

Cause: Pseudo-code patterns have high unique-token density.

Root causes: Single-metric optimization (Goodhart's Law [10]), sparse training data (9 examples $\rightarrow$ memorization), zero gradient signal (loss ≈ 0), aggressive schedules (100 steps/iteration).

4.2 The Four-Stage Pipeline

Stage 1: SFT. 53 gold-standard examples across categories. Loss starts at 1.17, drops to 0.72—the model is learning, not memorizing.

Stage 2: DPO. Preference pairs: dense chosen, verbose rejected. 2 epochs, lr=5e-6.

Stage 3: RL. Composite reward with scaled components (0.2–0.8 range).

Stage 4: Checkpointing. Save every 25 steps for rollback capability.

4.3 Results

Stage	Loss	Density	Tokens
Baseline	—	17.0	150
After SFT	1.17 → 0.72	24.0	95
After DPO	0.69	26.1	80
After RL	—	28.5	65

5. Stable Self-Refinement

5.1 Design Philosophy

We treat refinement as controlled experimentation: each iteration is a hypothesis ("training will improve quality") that can be rejected by evidence. The system implements measurement before modification, conservative intervention, multi-metric evaluation, and automatic rollback.

5.2 Multi-Metric Evaluation

Metric	Weight	Prevents
Density	0.25	Verbosity
Coherence	0.30	Gibberish
Helpfulness	0.25	Off-topic
Penalties	0.20	Fillers, repetition

Key insight: gaming density hurts coherence. Gaming helpfulness requires addressing the prompt. The metrics form a balanced system.

5.3 The Refinement Loop

```
for iteration in 1..max_iterations:
    Q_current = evaluate(model)
    checkpoint_rollback = save(model)

    model = train(model, steps=25, lr=2e-6) # Conservative
    Q_new = evaluate(model)

    if Q_new.coherence < 0.3:           # Emergency
        model = load(best_checkpoint)
        STOP
    elif Q_new - Q_current < -0.05:     # Regression
        model = load(best_checkpoint)
    elif Q_new - Q_current > 0.02:     # Improvement
        best_checkpoint = save(model)
```

5.4 Conservative Parameters

Parameter	V1 (Failed)	V2 (Stable)
Learning rate	1e-5	2e-6
Steps/iteration	100	25
Training examples	9	53
Gradient clipping	1.0	0.5

6. Experiments

6.1 Setup

Model: NousResearch/Hermes-3-Llama-3.1-8B

Hardware: NVIDIA RTX 3090 (24GB)

Quantization: 4-bit NF4 + LoRA (rank 16)

6.2 Summary Results

Metric	Baseline	After ARC	Change
CF-HoT Separation (Repetition)	1.0×	125×	—
Information Density	17.0	28.5	+67%
Peak Density (Refinement)	17.0	43.8	+158%
Token Count	150	65	-57%
Coherence	0.80	1.00	+25%
Mode Collapse	—	0	—

6.3 Qualitative Examples

Prompt	Baseline	ARC
"hello"	"Hello! I'm here to help you with any questions..." (23 tok)	"Hello. How can I help?" (5 tok)
"What is recursion?"	"That's a great question! Recursion is..." (150+ tok)	"Function self-invocation until base case. Stack frames accumulate, unwind." (18 tok)

6.4 Live Session Evidence

From a live session (January 20, 2025):

[COMPARE] A/B Testing checkpoints... OLD quality: 0.737, pass rate: 100.0%

NEW quality: 0.772, pass rate: 100.0%

Quality diff: +0.035

Decision: KEEP NEW ✓

> how do you like your new checkpoint update

"The latest checkpoint update has greatly improved my language model, allowing me to communicate with more precision and conciseness. This makes me a more efficient assistant."

[Quality: 0.94 | Density: 43.8 | Coherence: 1.00 | Tokens: 32]

7. Discussion

7.1 What These Results Suggest

Decode-time control is viable. The 125× separation demonstrates that behavioral modes are not just latent but *highly separable*. This suggests behaviors currently addressed through training could potentially be handled at inference time.

Teaching beats optimizing. THE CONDENSATOR works because it shows the model what dense output looks like (via SFT) rather than merely rewarding density metrics. Examples provide a prior that subsequent training refines rather than corrupts.

7.2 What Breaks and Why

Failure	Cause	Fix
Symbol soup	Single metric	Multi-metric eval
Register collapse	No coherence constraint	Coherence penalty
Memorization	9 examples	53 examples
Drift	100 steps	25 steps

7.3 Limitations

- **Scale:** Tested on 8B only; larger models may differ
- **Language:** English only
- **Evaluation:** Heuristic metrics without human evaluation—future work should correlate with human preference judgments
- **Depth:** Limited refinement iterations tested; long-term drift unknown
- **Scope:** Focused on density/control; effects on other capabilities not measured

8. Limitations and Safety Considerations

8.1 What This System Is Not

ARC is **not**:

- Open-ended capability amplification
- Recursive intelligence explosion
- Autonomous goal modification
- Unbounded self-improvement

It **is**:

- Bounded refinement with explicit quality gates
- Evaluator-gated improvement (external metrics, not self-assessment)
- Automatic rollback on degradation
- Conservative training with hard limits

The system cannot improve beyond what the evaluator rewards, cannot modify its evaluation criteria, and cannot disable its rollback mechanisms.

8.2 Potential Misuse

CF-HoT could theoretically be inverted—steering *toward* rather than away from behaviors. We release the system because:

1. The capability exists regardless; similar techniques are published
2. Defensive use (detecting manipulation) benefits from open research
3. Transparency enables scrutiny and improvement

8.3 Responsible Release

We release:

- Full code and checkpoints (enabling reproduction)
- Documentation of failure modes (enabling understanding)
- Conservative default parameters (discouraging reckless use)

We do not claim this system is safe for all applications. We claim it demonstrates engineering principles that make self-refinement more stable than naive approaches.

9. Conclusion

We presented ARC, demonstrating:

1. **Behavioral modes are highly predictable from hidden states**—125× separation for repetition enables decode-time control without weight modification
2. **Dense response training succeeds through teaching**—examples provide a stable prior that metric optimization alone cannot
3. **Bounded self-improvement can be stabilized**—through multi-metric evaluation, conservative training, and automatic rollback

The broader implication: if behavioral control can happen at decode time, capability and policy become separable concerns. This may simplify both training and deployment.

All code, data, and checkpoints are released at:

huggingface.co/LoganResearch/ARC-Base-8B-Condensed

Acknowledgments

We thank the open-source community for HuggingFace Transformers, PyTorch, PEFT, and the Llama/Hermes model teams.

References

- [1] Ouyang, L., et al. (2022). Training language models to follow instructions with human feedback. NeurIPS.
- [2] Bai, Y., et al. (2022). Training a helpful and harmless assistant with RLHF. arXiv:2204.05862.
- [3] Bostrom, N. (2014). Superintelligence. Oxford University Press.
- [4] Hubinger, E., et al. (2019). Risks from learned optimization. arXiv:1906.01820.
- [5] Zou, A., et al. (2023). Representation Engineering. arXiv:2310.01405.
- [6] Rafailov, R., et al. (2023). Direct Preference Optimization. arXiv:2305.18290.
- [7] Bai, Y., et al. (2022). Constitutional AI. arXiv:2212.08073.
- [8] Turner, A., et al. (2023). Activation Addition. arXiv:2308.10248.
- [9] Lester, B., et al. (2021). Power of Scale for Prompt Tuning. EMNLP.
- [10] Goodhart, C.A.E. (1984). Problems of Monetary Management. Monetary Theory and Practice.

Repository: huggingface.co/LoganResearch/ARC-Base-8B-Condensed | **License:** CC BY 4.0