Markovian Transformers for Informative Language Modeling

Anonymous submission

Abstract

Chain-of-Thought (CoT) reasoning often fails to faithfully reflect a language model's underlying decision process. We address this by introducing a *Markovian* language model framework that can be understood as a reasoning autoencoder: it creates a text-based bottleneck where CoT serves as an intermediate representation, forcing the model to compress essential reasoning into interpretable text before making predictions. We train this system using a policy gradient algorithm inspired by Group Relative Policy Optimization (GRPO), with parallel sampling and actor reward gradients derived from the constraint that our reward model uses the same parameters θ as the policy model. This approach achieves a 33.2% absolute accuracy improvement on GSM8K with Llama 3.1 8B. Comprehensive perturbation analysis across 5,888 comparison points and four model pairs demonstrates that Markovian training produces systematically higher sensitivity to CoT perturbations (effect differences +0.0154 to +0.3276), with 52.9%–87.6% consistency in showing greater fragility compared to Non-Markovian approaches. Cross-model evaluation confirms that learned CoTs generalize across architectures, indicating they capture transferable reasoning patterns rather than model-specific artifacts.

Introduction

The rapid advancement of language models (LMs) has led to impressive performance on complex cognitive tasks (**?**). Yet it is often unclear *why* an LM arrives at a particular conclusion (**???**), causing issues in high-stakes applications (**???**). Traditional interpretability methods analyze hidden activations or attention patterns to extract "explanations" (**???????**). Modern LMs, however, already generate coherent text: we might hope *prompting* the model to articulate its reasoning ("Chain-of-Thought" or CoT) (**??**) would yield a faithful record of its thought process.

Unfortunately, CoT explanations can be *unfaithful*. For example, **?** show that spurious in-context biases often remain hidden in the CoT, and **?** find that altering CoT text may not affect the final answer. Such observations indicate that standard CoTs are not "load-bearing."

In this work, we take a *pragmatic* approach to interpretability, focusing on *informativeness* over full faithfulness. Rather than insisting the CoT mirrors the model's entire internal process, we require that *the CoT alone suffices to produce the final answer*. In other words, if we remove the original prompt and rely only on the CoT, the model should still reach the correct output. This makes the CoT *causally essential* and *fragile*: changing it necessarily alters the prediction.

What distinguishes our approach is the clear distinction between the model *relying on its CoT* versus generating *more informative CoTs*. While traditional approaches train models to generate better-quality CoTs, they don't fundamentally change how the model uses them. Our Markovian framework, by contrast, forces the model to process information through the CoT bottleneck, making the CoT not just informative but *causally load-bearing* for prediction.

For instance, Mistral-7B's CoT on arithmetic tasks changed dramatically after training. **Before training**, it simply listed all numbers and their (incorrect) sum (e.g., "Sum = 76 + 90 + 92 + ... = 2314"). **After training**, it performed correct step-by-step calculations (e.g., "calculate 6 + 89 = 95; Next, calculate 95 + 38 = 133..."), breaking the task into manageable steps that can be verified independently and enabling accurate answer prediction even when the original question is removed.

Recipient-Specific Compression. A key insight is that an *informative* CoT can also serve as a *recipient-specific compression* of the model's hidden knowledge: it distills the essential reasoning into text that another recipient (e.g. a different model or a human) can use to predict the same outcome. Our experiments confirm that the learned CoTs generalize across interpreters, suggesting that these textual explanations genuinely encode transferable problem-solving steps rather than model-specific quirks (Section).

Contributions.

1. We introduce a Markovian language model framework that structurally enforces Chain-of-Thought (CoT) generation to be causally essential, ensuring reliance on the CoT for predictions.
2. We apply this framework to arithmetic problems (Mistral 7B) and the GSM8K dataset (**?**) (Llama 3.1 8B), observing a 33.2% absolute improvement on GSM8K.
3. We show through systematic perturbation analysis across four model pairs that Markovian training produces significantly higher sensitivity to CoT perturbations compared

to Non-Markovian approaches, with effect differences ranging from +0.0154 to +0.3276 in log-probability sensitivity.

4. We demonstrate cross-model transfer: CoTs trained on one model remain informative for other models. This underscores the CoT's *recipient-specific* interpretability and suggests it captures a shared reasoning strategy.

Section reviews related work, Section details our Markovian framework, and Section describes the RL training. Section presents empirical results, and Section discusses limitations and future directions.

Related Work

Prior work shows that CoT prompting can boost performance on reasoning tasks (**??**). Whereas typical CoT prompting methods do not alter a pre-trained model's parameters, some prior approaches do fine-tune the model for CoT generation (**???**). Our work differs by removing the original question or passage from the answer-prediction context, which enforces a stronger causal reliance on the CoT.

Regarding faithfulness vs. interpretability, some authors discuss how a CoT may fail to reflect the true reason the LM arrived at its answer (**??**), since small changes in the CoT do not necessarily change the final prediction. **?** analyze CoT through an information-theoretic lens, finding that CoT can serve as a communication channel between different parts of a model. We build on these insights by *training* the model to rely on this channel exclusively.

Architecturally, our Markovian LM shares structural similarities with state space models like RNNs (**?**), S4 (**?**), and Mamba (**?**), though with a key difference: MLMs have probabilistic state transitions to model token sampling, which necessitates gradient estimation methods such as policy gradient (**?**) rather than direct backpropagation. This probabilistic structure also resembles Kalman filters (**?**), Deep Variational Bayes Filters (**?**), Deep Kalman Filters (**?**), and Variational Recurrent Neural Networks (VRNN) (**?**), though we use categorical rather than Gaussian distributions for interpretable text generation. Other fine-tuned reasoning models mentioned above (R1, STaR, and QuietSTaR) have similar structure but allow seeing the full context before generating state/reasoning tokens, whereas our approach enforces a strict information bottleneck through the state.

? also consider restricting the model's ability to see the original input while generating the final answer. Their approach, however, involves rewriting the question in a structured formal language or code that is then executed. Our approach uses natural language for the reasoning state to preserve interpretability across diverse tasks.

Markovian Language Models and Informativeness

Here we provide our formalism for Markovian Language Models (MLMs) and define *informativeness*, which we use as a training objective within our novel structural framework.

Markovian Language Models (MLM)

A traditional LM can attend to the entire context when predicting the next token. This makes it possible for an LM to disregard the CoT or only partially rely on it. We impose a stricter, *Markovian* structure[1]:

Definition 0.1 (Markovian LM)**.** *A Markovian Language Model is a tuple $M = (\mathcal{O}, \mathcal{S}, \pi, u, s_1)$, where*

- *$\mathcal{O}$ is a set of observations (e.g., questions and answers in a QA task),*
- *$\mathcal{S}$ is a set of states (e.g., CoT reasoning text),*
- *$\pi : \mathcal{S} \to \Delta(\mathcal{O})$ is a policy that predicts the next observation from the state alone,*
- *$u : \mathcal{O} \times \mathcal{S} \to \Delta(\mathcal{S})$ is a state update function (produces CoT from question and initial prompt),*
- *$s_1 \in \mathcal{S}$ is an initial state (starting CoT prompt).*

For example, in a math reasoning task, $o_1 \in \mathcal{O}$ might be a question, $s_1 \in \mathcal{S}$ is an initial CoT prompt like "Let's solve this step-by-step:", $s_2 \in \mathcal{S}$ is the generated reasoning chain, and $o_2 \in \mathcal{O}$ is the answer. The key idea is that π can only see the CoT state s_2 when predicting o_2, forcing the CoT to contain all needed information. Intuitively, π is the *frozen* next-token predictor, and u is the model's *trainable* component that chooses how to produce the CoT from the latest observation and prior state. In our experiments, π and u share the same underlying transformer but we freeze the weights for π while fine-tuning those used by u.

Data-Generating Distribution and Reward

Let P be the distribution over observations $x_1, x_2, \ldots, x_T \in \mathcal{O}$. A trajectory τ is generated by:

$$s_{t+1} \sim u(s_t, x_t), \quad x_{t+1} \sim P(x_{t+1} \mid x_{\leq t}),$$

with s_1 a fixed initial prompt. We define the *reward* for a trajectory τ as:

$$R_\theta(\tau) = \sum_{t=1}^{T} \left[\ln \pi_\theta(x_t \mid s_t) - \ln \pi_\theta(x_t \mid s'_t)\right],$$

where s'_t is generated by a *baseline* update function u', e.g., the *untrained* model. In words, $R_\theta(\tau)$ measures how much more likely the correct observation x_t is under the trained state s_t compared to the baseline state s'_t.

Informativeness Objective

Conceptually, we aim to ensure that the CoT state serves as a critical bottleneck for information flow, making it causally essential for predictions. Formalizing this within our Markovian framework, we define:

$$J(\theta) = \mathbb{E}_{\tau \sim P, u_\theta, u'} \left[R_\theta(\tau)\right],$$

[1]This structure can be viewed as a stochastic variant of a Moore machine where both the transition function (u) and output function (π) are probabilistic, and the input and output alphabets are identical (O). Alternatively, an MLM can be formalized as an F-coalgebra where F(S) = P(O) × P(S)^O, with P representing probability distributions.

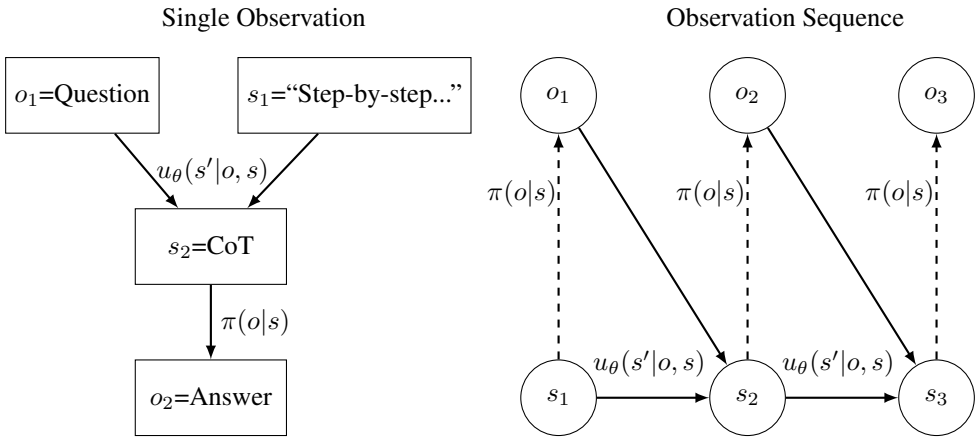

Figure 1: Markovian training as a reasoning autoencoder. Left: Single time-step process from Question to CoT to Answer, creating a text-based bottleneck where the CoT must capture all information needed for answer prediction. Right: Causal structure showing the generation of states from observations and previous states using the state update function $u_\theta(s'|o,s)$, and the prediction of observations from states using the policy $\pi_\theta(o|s)$. This architecture forces reasoning through an interpretable text bottleneck, but prevents direct backpropagation, necessitating RL-based gradient estimation. In experiments, both u_θ and π_θ are implemented using the same transformer (Mistral 7B or Llama 3.1 8B), with only u_θ's weights updated during training.

where θ parameterizes u_θ. Maximizing $J(\theta)$ ensures that the update function u_θ produces states s_t that are *informative* about future observations (relative to the baseline u'), thereby enforcing the CoT's role as a load-bearing component. We optimize $J(\theta)$ with policy gradient or PPO, sampling observations from P and states from u_θ and u'.

Methods

Implementation as Question-Answer Pairs

In many tasks like math problem solving, we have $T = 2$ observations (question and answer) and implement the abstract MLM with a fixed maximum length for the CoT state. Let $\mathcal{V}$ be a token vocabulary. We set $\mathcal{O} = \mathcal{V}^N$ and $\mathcal{S} = \mathcal{V}^K$ for some $N, K \in \mathbb{N}$, where K is the maximum tokens in the CoT. Note that while we limit the state to a maximum of K tokens for implementation, we do not enforce fixed-length observations.

Our conceptual arguments rely on $K < N$, as otherwise the model could simply write the predicted observation into the state. We satisfy this in our Wikipedia experiments (Sec), and for other experiments we find empirically that the model does not learn this undesirable behavior due to the difficulty of predicting the answer directly without any CoT.

In this setting, we denote our states as $s_1 = \text{CoT}_{\text{init}}$ and $s_2 = \text{CoT}$, where CoT_{init} is a task-specific prompt[2]. With pre-trained LM $\mathcal{L}$, we can implement our update function u and policy π using:

$$\ln u\big(s_2 = \text{CoT} \mid q, s_1 = \text{CoT}_{\text{init}}\big) \tag{1}$$

$$= \sum_{i=1}^{K} \ln \mathcal{L}\Big(\text{concat}\big(q,\ \text{CoT}_{\text{init}},\ \text{CoT}_{<i}\big)\Big)\big[\text{CoT}_i\big], \tag{2}$$

$$\ln \pi_\theta(\text{ans} \mid \text{CoT}) \tag{3}$$

$$= \sum_{i=1}^{N} \ln \mathcal{L}(\text{concat}(\text{CoT}, \text{ans}_{<i}))[\text{ans}_i]. \tag{4}$$

Crucially, we do *not* allow the answer generation to attend back to the question q directly; the question is replaced by the CoT. For each question q, we generate the baseline state s_2' (which we denote as CoT' in this setting) by prompting the unmodified pre-trained model with q plus an initial instruction (e.g., 'Think step-by-step...'), and recording its raw output.

Our reward is:

$$R_\theta = \ln \pi_\theta(\text{ans} \mid \text{CoT}) \ - \ \ln \pi_\theta(\text{ans} \mid \text{CoT}').$$

Policy Gradient with GRPO-Style Baseline

Markovian training can be understood as training a *reasoning autoencoder*, where the CoT serves as a text-based bottleneck between question and answer. Like traditional autoencoders, this architecture forces information compression, but the intermediate representation is interpretable text rather than latent vectors. This text bottleneck prevents direct backpropagation and necessitates reinforcement learning techniques for gradient estimation.

Actor Reward Gradients: The Key Innovation Our approach differs fundamentally from standard reinforcement learning by using the *same* transformer with weights θ as

[2]The exact prompt template varies by task type, with each template specifying the task objective, allowed CoT length, and an invitation to reason strategically. Full templates are provided in Sec .

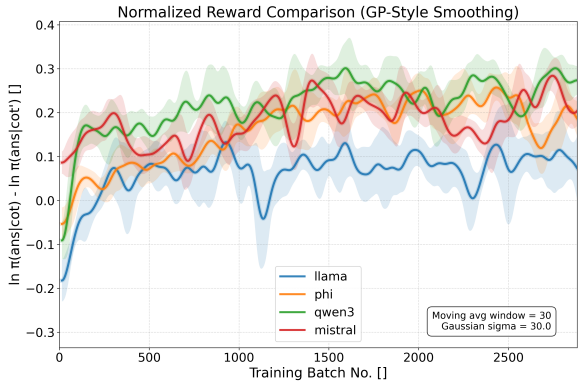

Figure 2: Normalized reward progression during Wikipedia continuation training across four model architectures. The normalized reward $\ln \pi_\theta(\text{ans} \mid \text{CoT}) - \ln \pi_\theta(\text{ans} \mid \text{CoT}')$ measures how much more informative the trained CoT becomes compared to baseline reasoning from the unmodified model. Each curve represents a different model architecture: Llama 3.1 8B (blue), Phi-3.5 Mini (orange), Qwen3 4B (green), and Mistral 7B (red). The plot uses Gaussian Process-style smoothing with confidence bands to highlight training trends. All models show consistent improvement in CoT informativeness, demonstrating the generalizability of the Markovian training approach across diverse architectures.

both the policy model and the reward model. This creates a crucial mathematical distinction from traditional policy gradient methods.

In classical policy gradient, the reward $R(\tau)$ is independent of the policy parameters, leading to the standard REINFORCE gradient:

$$\nabla_\theta \mathbb{E}_{\tau \sim P_\theta}[R(\tau)] = \mathbb{E}_{\tau \sim P_\theta}[R(\tau) \cdot \nabla_\theta \ln P_\theta(\tau)]$$

However, in our case, the reward is a function of the same parameters: $R_\theta(\tau) = \ln \pi_\theta(\text{ans} \mid \text{CoT})$. Applying the chain rule:

$$\nabla_\theta \mathbb{E}_{\tau \sim P_\theta}[R_\theta(\tau)] = \mathbb{E}_{\tau \sim P_\theta}[R_\theta(\tau) \cdot \nabla_\theta \ln P_\theta(\tau) + \nabla_\theta R_\theta(\tau)] \tag{5}$$

This yields two terms: the standard policy gradient $(R_\theta(\tau) \cdot \nabla_\theta \ln P_\theta(\tau))$ and the direct reward gradient $(\nabla_\theta R_\theta(\tau))$. We include both terms with equal weight in our implementation.

GRPO-Style Baseline with Local Subtraction We implement a policy gradient algorithm inspired by Group Relative Policy Optimization (GRPO), originally introduced by Shao et al. (**?**) in DeepSeek-Math. GRPO eliminates the critic model from PPO by using group-based advantage estimation, where multiple responses to the same query provide relative baselines for each other.

However, we add an additional baseline subtraction step before applying GRPO's batch averaging. We first compute a local baseline using the frozen reference model u', then apply GRPO-style standardization within each batch.

Parallel Sampling Strategy We employ *parallel sampling* (inspired by GRPO): each training batch contains B copies of the same question-answer pair (q, a). The trainable model u_θ generates diverse reasoning chains $\{\text{CoT}_1, \text{CoT}_2, \ldots, \text{CoT}_B\}$ for the identical input through stochastic sampling.

Additionally, we introduce a frozen baseline from the reasoning autoencoder: the unmodified model u' generates a single reference CoT' that provides a local baseline before applying GRPO-style batch averaging. This frozen baseline represents the "encoder" component of our reasoning autoencoder—capturing the model's initial reasoning ability before training. The frozen baseline CoT' is *not* part of the original GRPO algorithm—it is our contribution to provide a more stable reference point.

This approach provides several advantages:

- **Reasoning bottleneck**: The CoT' baseline establishes the initial encoding capacity of the reasoning autoencoder
- **Local baseline**: The frozen CoT' provides a consistent reference for measuring informativeness improvement
- **Computational efficiency**: Baseline reasoning and answer evaluation are computed once and replicated
- **Stable variance estimation**: All samples share the same ground truth, enabling robust within-batch standardization

Implementation: Two-Term Loss Function Our implementation combines both gradient terms from the chain rule derivation above. The loss function includes:

$$\mathcal{L} = \mathcal{L}_{\text{PG}} + \mathcal{L}_{\text{AR}} \tag{6}$$

$$\mathcal{L}_{\text{PG}} = -\ln u_\theta(\text{CoT} \mid q, \text{CoT}_{\text{init}}) \cdot A^{\text{detach}} \tag{7}$$

$$\mathcal{L}_{\text{AR}} = -A \tag{8}$$

where A is the standardized advantage (after local baseline subtraction and GRPO-style batch averaging) and A^{detach} blocks gradients to isolate the policy gradient term.

The first term $\mathcal{L}_{\text{PG}}$ corresponds to the standard REINFORCE gradient $R_\theta(\tau) \cdot \nabla_\theta \ln P_\theta(\tau)$, while the second term $\mathcal{L}_{\text{AR}}$ corresponds to the direct reward gradient $\nabla_\theta R_\theta(\tau)$. This enables simultaneous optimization of CoT generation and answer prediction.

Within-Batch Advantage Standardization Instead of historical exponential moving averages, we standardize advantages within each batch:

$$R_i = \ln \pi_\theta(\text{ans} \mid \text{CoT}_i) - \ln \pi_\theta(\text{ans} \mid \text{CoT}') \tag{9}$$

$$A_i = \frac{R_i - \mu_{\text{batch}}}{\sigma_{\text{batch}} + \epsilon} \tag{10}$$

where $\mu_{\text{batch}} = \frac{1}{B}\sum_{i=1}^{B} R_i$ and $\sigma^2_{\text{batch}} = \frac{1}{B}\sum_{i=1}^{B}(R_i - \mu_{\text{batch}})^2$.

This ensures that advantages have zero mean and unit variance within each batch, providing stable gradients regardless of the absolute reward scale.

Algorithm 1: Policy Gradient with GRPO-Style Baseline

Require: Actor model u_θ (trainable), Critic model u' (frozen), batch size B
Require: Dataset $\mathcal{D}$, learning rate α, temperature T, gradient accumulation steps G
for each training iteration **do**
  Sample unique example $(q, a) \sim \mathcal{D}$
  Create batch $\{(q, a), (q, a), \ldots, (q, a)\}$ with B copies
  // Generate reasoning chains
  for $i = 1$ to B **do**
    $\text{CoT}_i \sim u_\theta(\cdot|q, \text{CoT}_{\text{init}})$ {Stochastic sampling with temperature T}
  end for
  $\text{CoT}' \sim u'(\cdot|q, \text{CoT}_{\text{init}})$ {Deterministic baseline, computed once}
  // Calculate informativeness rewards with local baseline
  for $i = 1$ to B **do**
    $R_i = \ln u_\theta(\text{ans}|\text{CoT}_i) - \ln u_\theta(\text{ans}|\text{CoT}')$
  end for
  // Standardize advantages within batch
  $\mu = \frac{1}{B}\sum_{i=1}^{B} R_i, \quad \sigma = \sqrt{\frac{1}{B}\sum_{i=1}^{B}(R_i - \mu)^2}$
  $A_i = \frac{R_i - \mu}{\sigma + \epsilon}$ for $i = 1, \ldots, B$ $\{\epsilon = 10^{-8}\}$
  // Calculate loss with actor reward gradients
  for $i = 1$ to B **do**
    $\ell_i^{\text{PG}} = -\ln u_\theta(\text{CoT}_i|q, \text{CoT}_{\text{init}}) \cdot A_i^{\text{detach}}$
    $\ell_i^{\text{AR}} = -A_i$ {Direct reward optimization}
    $\ell_i = \ell_i^{\text{PG}} + \ell_i^{\text{AR}}$
  end for
  // Gradient accumulation and update
  $L = \frac{1}{B \cdot G}\sum_{i=1}^{B} \ell_i$ {Scale by accumulation steps}
  Accumulate gradients: $\nabla_\theta L$
  if accumulated G steps **then**
    Clip gradients: $\|\nabla_\theta\| \leq 1.0$
    Update: $\theta \leftarrow \theta - \alpha\nabla_\theta$
    Reset gradient accumulator
  end if
end for

Training Stability and Implementation Details

Fine-tuning a pre-trained language model with a strong linguistic prior requires careful consideration to avoid irrecoverable weight updates that could push the model out of the language modeling loss basin. We implement several techniques to enhance training stability for the GRPO objective:

1. **Low-Rank Adaptation (LoRA) (?):**
   - Freeze all weights except for small-rank LoRA adapters.
   - Use rank 8 with $\alpha = 16$.
2. **Gradient Clipping:**
   - If the ℓ_2 norm of the gradient exceeds 1.0, rescale it to norm 1.0.
3. **Gradient Accumulation:**
   - Use gradient accumulation steps of 4 for Wikipedia experiments.
   - Scale loss by accumulation steps to maintain consistent gradient magnitudes.
4. **Within-Batch Advantage Standardization:**
   - GRPO's parallel sampling enables robust within-batch standardization, eliminating the need for historical baselines.
   - Each batch provides its own reference distribution for advantage calculation.
5. **Actor Reward Weight:**
   - Set actor reward weight to 1.0 to equally balance policy gradient and direct reward optimization.
   - This enables end-to-end learning through the reward model.
6. **Initial CoT Prompt Design:**
   - Choose CoT_{init} to guide the model toward meaningful reasoning.
   - For arithmetic:
     "You will be given an arithmetic problem, which you have [CoT length] tokens to work through step-by-step. Question:"
   - For GSM8K:
     "You will be given a reasoning problem, which you have [CoT length] tokens to work through step-by-step. Question:"
   - For Wikipedia continuation:
     "Compress your understanding of this text into [CoT length] tokens, then predict the next [target length] tokens."

These measures greatly reduce the risk of catastrophic updates and keep the model's training on track.

Experiments

Multi-step Addition

We generate random addition problems, where each problem consists of fifteen terms and each term is a uniform random natural number less than 100. We fine-tune Mistral 7B Instruct V0.2 to produce CoT tokens such that a frozen copy

of the pre-trained language model can predict the correct answer given that CoT, for each training technique in Sec . We plot the mean negative log likelihood over the answer tokens as a function of training batch in Fig. 2. Note that this is both training and testing loss, since we are always generating fresh arithmetic problems. PPO, our preferred training method for arithmetic, can mention the correct answer in up to 90% of CoTs and achieve an average natural log probability of around -0.7. Additional detailed arithmetic performance analysis is provided in Appendix **??**.

Since the Mistral tokenizer allocates a separate token for each digit, a natural log probability of -0.7 corresponds to about 50% probability ($e^{-0.7} \approx 0.4966$) per token. The seeming contradiction between 90% verbatim answer likelihood and 50% per-digit uncertainty stems from the predictor's format uncertainty—it distributes probability across the entire vocabulary when deciding what follows "Answer:", as we only train CoT production $u_\theta(s'|o, s)$, not the predictor $\pi_\theta(o|s)$.

GSM8K

To test our method on more complex reasoning tasks, we train Llama-3.1-8B-Instruct on GSM8K using policy gradient with expert iteration (threshold 2.2 standard deviations) and a KL penalty (0.1). We produce up to 150 CoT tokens and estimate the value function with an exponentially decaying average of previous rewards (decay 0.9).

Our experiments show dramatic improvements in exact-match accuracy from 35.94% baseline to 69.14% in our best run—a 33.2% absolute improvement. The other runs (58.23% and 62.85%) confirm consistent effectiveness on mathematical reasoning, with CoT informativeness and proportion of CoTs containing verbatim answers both increasing throughout training.

Wikipedia

We also explored applying our approach to general language modeling using Wikipedia text. For each article, we condition on the first 200 tokens and task the model with predicting the following 100 tokens, allowing 50 tokens of CoT to aid prediction. Training parameters match those used in GSM8K.

Results showed modest improvements in next-token prediction accuracy from 8.2% to 10.5% (see Code and Data Appendix). This should be contextualized against pre-trained Llama's typical 16.9% accuracy (over 10,000 articles) on the 200$^{\text{th}}$ to 300$^{\text{th}}$ tokens without context. The lower baseline (8.2%) likely stems from our setup with CoT followed by " Answer: " before prediction. Despite this, key findings about CoT reliability remain evident: as detailed in Sec , systematic perturbation analysis reveals that Markovian training produces significantly higher sensitivity to CoT perturbations compared to Non-Markovian approaches, indicating genuine structural differences in CoT reliance. **See Code and Data Appendix for examples of CoT changes after training.**

Markovian vs Non-Markovian Perturbation Sensitivity

To provide systematic evidence for the theoretical advantages of Markovian training, we conduct comprehensive perturbation sensitivity comparisons between Markovian and Non-Markovian model pairs. This analysis directly evaluates whether the structural constraints in Markovian training lead to measurably different robustness properties during training.

Experimental Design We train four independent model pairs on Wikipedia continuation tasks, where each pair consists of architecturally identical transformers differing only in their reward calculation method: Markovian models compute rewards based solely on current token predictions, while Non-Markovian models incorporate full sequence context.

For each training batch, we apply four perturbation types at five severity levels (20%, 40%, 60%, 80%, 100%):

- **Delete**: Random token deletion from CoT reasoning
- **Truncate Front**: Removal of tokens from CoT beginning
- **Truncate Back**: Removal of tokens from CoT end
- **Character Replace**: Random character substitution within tokens

The sensitivity measure compares log-probability changes for correct predictions:

$$\text{Effect}_{\text{M/NM}} = \ln P(\text{ans}|\text{CoT}_{\text{original}}) - \ln P(\text{ans}|\text{CoT}_{\text{perturbed}}) \tag{11}$$

$$\text{Difference} = \text{Effect}_{\text{Markovian}} - \text{Effect}_{\text{Non-Markovian}} \tag{12}$$

Positive differences indicate Markovian models exhibit greater sensitivity to perturbations, reflecting stronger reliance on CoT content integrity.

Results Summary Analysis of 5,888 comparison points reveals systematic sensitivity differences between training paradigms. As shown in Table 1, delete perturbations produce the strongest effects (+0.1193 to +0.3276 mean difference), with Markovian models showing consistently higher sensitivity that increases with perturbation severity. Character replacement demonstrates robust differences across all severities (+0.1271 to +0.2548), while truncation effects are more modest at lower severities but converge to similar high-severity patterns.

Critically, these patterns hold across all four independent model pairs, indicating systematic rather than coincidental differences. Consistency measures (percentage of cases where Markovian models show higher sensitivity) range from 52.9% for mild truncations to 87.6% for severe deletions, providing strong statistical evidence for the theoretical predictions of our framework.

The comprehensive analysis reveals systematic sensitivity differences between training paradigms, with Markovian models consistently exhibiting greater perturbation sensitivity across all tested conditions. These results provide strong empirical support for the theoretical advantages of Markovian training in developing robust reasoning dependencies.

Perturbation Type	Degree	Mean Effect Difference	Overall Consistency	Total Comparisons
Delete	20%	+0.1193	76.1%	1,472
	40%	+0.2170	83.3%	1,472
	60%	+0.2525	84.0%	1,472
	80%	+0.2612	84.8%	1,472
	100%	+0.3276	87.6%	1,472
Truncate Front	20%	+0.0317	56.4%	1,472
	40%	+0.0575	65.4%	1,472
	60%	+0.0893	70.6%	1,472
	80%	+0.1457	77.7%	1,472
	100%	+0.3276	87.6%	1,472
Truncate Back	20%	+0.0154	52.9%	1,472
	40%	+0.0411	58.1%	1,472
	60%	+0.1003	64.9%	1,472
	80%	+0.1681	73.4%	1,472
	100%	+0.3276	87.6%	1,472
Character Replace	20%	+0.1271	76.3%	1,472
	40%	+0.2245	85.9%	1,472
	60%	+0.2485	86.5%	1,472
	80%	+0.2534	86.9%	1,472
	100%	+0.2548	86.9%	1,472

Table 1: Comprehensive analysis of perturbation sensitivity differences between Markovian and Non-Markovian language model training approaches across four distinct model pairs and four perturbation types. The analysis is based on 5,888 total comparison points (1,472 per perturbation type) collected during training on Wikipedia continuation tasks. **Mean Effect Difference** represents the average difference in log-probability sensitivity (Markovian effect - Non-Markovian effect), where positive values indicate that Markovian models exhibit greater sensitivity to perturbations. **Consistency** measures the percentage of training instances where Markovian models showed higher perturbation sensitivity than their Non-Markovian counterparts. Key findings: (1) Delete perturbations show the strongest differences (+0.1193 to +0.3276), (2) Character replacement exhibits robust sensitivity across all severities, (3) Truncation effects are more modest at low severities but converge at high severities, (4) All patterns hold consistently across independent model pairs.

Interpretability of CoT Generations

To probe how well the reasoning generalizes, we evaluated the informativeness of Llama's trained CoTs with respect to various other language models on the Wikipedia dataset. Cross-model evaluation shows strong correlation between improvements in both the trained model's and alternative models' evaluations of CoT quality throughout training. The normalized log probabilities increase simultaneously across different architectures, demonstrating that Llama is learning to produce generic CoTs which do not over-fit to the peculiarities of a Llama answer-predictor. Results averaged across 6 independent training runs confirm this pattern holds consistently (detailed results in Appendix **??**).

This cross-model transferability addresses a key question: "interpretable to whom?" We test across three distinct model families (Phi (**?**), Mistral, and GPT2), including GPT2, a significantly smaller model that shouldn't be able to decode sophisticated steganography. The fact that trained CoTs transfer effectively across this diverse set confirms they contain generalizable reasoning patterns rather than model-specific artifacts.

Discussion and Limitations

Experiments across arithmetic, GSM8K, and Wikipedia show that it is possible to learn informative and interpretable CoT reasoning via RL on an LM using Markovian training.

However, our interpretability technique is currently only verified in myopic question-answer datasets, as opposed to multi-turn trajectories where trained CoTs might provide a lens into longer-term future behavior. In principle, the Markovian design naturally extends to multi-turn or multi-step settings by treating the CoT as recurrent state; we have not explored such tasks here for scope reasons.

Moreover, we have only evaluated interpretability by measuring *model*-centric proxies (like CoT fragility and cross-model transfer). A more direct human evaluation would have people read the generated CoTs and attempt to predict the final answer, giving an explicit measure of whether these CoTs are genuinely human-interpretable. Such a setup could be incorporated into the training objective, where human correctness in predicting the answer provides an additional signal for optimizing CoT generation.

Markovian training is language modeling with an intermediate memory-producing action, similar to R1 recurrence and "thinking" models. This approach blurs the line between RL and unsupervised learning, though its expensive serial token generation requires justification through gains in interpretability or perplexity.

Our findings indicate that Markovian training yields substantial gains in CoT fragility and cross-model transfer, suggesting practical opportunities for improved interpretability. While human studies could further validate interpretability, we rely on cross-model transfer as a proxy and leave comprehensive trials to future work.

Future Work. Although we focus on single question–answer pairs, the Markovian framework extends to multi-turn dialogue. After each user message o_t, we produce the next CoT s_{t+1} via $u_\theta(s_{t+1} \mid s_t, o_t)$, then generate the system's reply from that CoT alone. This process treats the CoT as a recurrent state, which could scale to conversation rounds.

Reproducibility Checklist

Reproducibility Checklist

Instructions for Authors:

This document outlines key aspects for assessing reproducibility. Please provide your input by editing this `.tex` file directly.

For each question (that applies), replace the "Type your response here" text with your answer.

Example: If a question appears as

```
\question{Proofs of all novel claims
are included} {(yes/partial/no)}
Type your response here
```

you would change it to:

```
\question{Proofs of all novel claims
are included} {(yes/partial/no)}
yes
```

Please make sure to:

- Replace ONLY the "Type your response here" text and nothing else.
- Use one of the options listed for that question (e.g., **yes**, **no**, **partial**, or **NA**).
- **Not** modify any other part of the `\question` command or any other lines in this document.

You can `\input` this .tex file right before `\end{document}` of your main file or compile it as a stand-alone document. Check the instructions on your conference's website to see if you will be asked to provide this checklist with your paper or separately.

---

1. General Paper Structure

1.1. Includes a conceptual outline and/or pseudocode description of AI methods introduced (yes/partial/no/NA) yes

1.2. Clearly delineates statements that are opinions, hypothesis, and speculation from objective facts and results (yes/no) yes

1.3. Provides well-marked pedagogical references for less-familiar readers to gain background necessary to replicate the paper (yes/no) yes

2. Theoretical Contributions

2.1. Does this paper make theoretical contributions? (yes/no) partial

If yes, please address the following points:

2.2. All assumptions and restrictions are stated clearly and formally (yes/partial/no) yes

2.3. All novel claims are stated formally (e.g., in theorem statements) (yes/partial/no) no

2.4. Proofs of all novel claims are included (yes/partial/no) no

2.5. Proof sketches or intuitions are given for complex and/or novel results (yes/partial/no) yes

2.6. Appropriate citations to theoretical tools used are given (yes/partial/no) yes

2.7. All theoretical claims are demonstrated empirically to hold (yes/partial/no/NA) yes

2.8. All experimental code used to eliminate or disprove claims is included (yes/no/NA) yes

3. Dataset Usage

3.1. Does this paper rely on one or more datasets? (yes/no) yes

If yes, please address the following points:

3.2. A motivation is given for why the experiments are conducted on the selected datasets (yes/partial/no/NA) yes

3.3. All novel datasets introduced in this paper are included in a data appendix (yes/partial/no/NA) NA

3.4. All novel datasets introduced in this paper will be made publicly available upon publication of the paper with a license that allows free usage for research purposes (yes/partial/no/NA) NA

3.5. All datasets drawn from the existing literature (potentially including authors' own previously published work) are accompanied by appropriate citations (yes/no/NA) yes

3.6. All datasets drawn from the existing literature (potentially including authors' own previously published work) are publicly available (yes/partial/no/NA) yes

3.7. All datasets that are not publicly available are described in detail, with explanation why publicly available alternatives are not scientifically satisficing (yes/partial/no/NA) NA

4. Computational Experiments

4.1. Does this paper include computational experiments? (yes/no) yes

If yes, please address the following points:

4.2. This paper states the number and range of values tried per (hyper-) parameter during development of the paper, along with the criterion used for selecting the final parameter setting (yes/partial/no/NA) partial

4.3. Any code required for pre-processing data is included in the appendix (yes/partial/no) yes

4.4. All source code required for conducting and analyzing the experiments is included in a code appendix (yes/partial/no) yes

4.5. All source code required for conducting and analyzing the experiments will be made publicly available upon publication of the paper with a license that allows free usage for research purposes (yes/partial/no) yes

4.6. All source code implementing new methods have comments detailing the implementation, with references to the paper where each step comes from (yes/partial/no) partial

4.7. If an algorithm depends on randomness, then the method used for setting seeds is described in a way sufficient to allow replication of results (yes/partial/no/NA) partial

4.8. This paper specifies the computing infrastructure used for running experiments (hardware and software), including GPU/CPU models; amount of memory; operating system; names and versions of relevant software libraries and frameworks (yes/partial/no) partial

4.9. This paper formally describes evaluation metrics used and explains the motivation for choosing these metrics (yes/partial/no) yes

4.10. This paper states the number of algorithm runs used to compute each reported result (yes/no) yes

4.11. Analysis of experiments goes beyond single-dimensional summaries of performance (e.g., average; median) to include measures of variation, confidence, or other distributional information (yes/no) yes

4.12. The significance of any improvement or decrease in performance is judged using appropriate statistical tests (e.g., Wilcoxon signed-rank) (yes/partial/no) no

4.13. This paper lists all final (hyper-)parameters used for each model/algorithm in the paper's experiments (yes/partial/no/NA) yes