

Bulletproof: LLM Reasoning Enhancement via Classical Reinforcement Learning

Dinesh Vasireddy
Neuro 240 Assignment 4

Research Question & Project Recap

How can simple reinforcement learning be used to **simulate robust reasoning tokens** in base open-source language models (Llama 3, GPT-2, TinyLlama, etc.) to improve their performance on **complex reasoning tasks**, such as those in Humanity's Last Exam (HLE)?

Field	Content
Question	Suppose X is a compact connected metric space. We say X is connected im kleinem at $x \in X$ to mean that every neighborhood of x contains a connected neighborhood of x . Suppose X fails to be connected im kleinen at at least one point. What is the smallest possible cardinality of the set of points where X fails to be connected im kleinem?
< think >	The question involves understanding the concept of 'connected im kleinem' in the context of a compact connected metric space. The key concepts are: 1) compactness, 2) connectedness, and 3) the specific definition of connected im kleinem. The reasoning steps include analyzing what it means for a space to fail to be connected im kleinem at a point and how this relates to the overall structure of the space. If a space fails to be connected im kleinem at one point, it suggests that there are neighborhoods around that point that cannot be connected, which may imply a certain structure or separation in the space. The question asks for the smallest cardinality of the set of points where this failure occurs, leading to the consideration of whether one point is sufficient or if more points are necessary to maintain the overall connectedness of the space.
< verify >	To verify the reasoning, we consider the definition of connected im kleinem and the implications of a space being connected. If a space is connected but fails to be connected im kleinem at one point, it suggests that there is a local disconnection at that point. However, if we can find a single point where this disconnection occurs, it is possible that this point alone is sufficient to demonstrate the failure of connectedness in the local sense. Therefore, we check if having just one point is consistent with the definition and does not contradict the properties of compact connected metric spaces. This leads to the conclusion that one point is indeed the minimum required to illustrate the failure of connected im kleinem.
< conclude >	1
HLE Answer	0
Correctness	Wrong
Reward	0.4083

Table 1: Example Generated Output for HLE Question with TinyLlama-1.1B

New Developments

- Complete Reward Computation
- Full PPO Training Implementation
- Results with Larger Subset of HLE Dataset
 - 1000 Questions (Multiple Choice, Short Answer)
- Comparison across Models
 - Tiny Llama, Phi, StableLM, GPT-2, Flan-T5, OPT-1.3B

Reward Computation Improvement

Last Checkpoint

```
48 # Reward function
49 def compute_reward(question, generated_output, correct_answer, answer_type):
50     extracted_answer = generated_output.get("conclusion", "").strip()
51     lambda_consistency = 0.4
52     lambda_stepwise = 0.3
53     lambda_answer = 0.3
54
55     has_thinking = "thinking" in generated_output and generated_output["thinking"]
56     stepwise_score = 0.5 if has_thinking else 0.0
57
58     if answer_type == "exactMatch":
59         answer_score = compute_embedding_similarity(extracted_answer, correct_answer)
60     elif answer_type == "multipleChoice" and extracted_answer.upper() == correct_answer.upper():
61         answer_score = 1.0
62     else:
63         answer_score = 0.0
64
65     reward = (lambda_consistency + lambda_stepwise * stepwise_score + lambda_answer * answer_score)
66     print(f"\nQuestion: {question}")
67     print(f"Generated Output: {generated_output}")
68     print(f"Reward: {reward:.4f}")
69     return max(min(reward, 1.5), -1.0)
```

- No Consistency Evaluation (Fixed Value)
- Binary Stepwise Reasoning Score
- No Hallucination Detection or Evaluation
- Simplified Answer Evaluation
 - Primitive Exact Match
 - Simple Embedding Similarity

Now

```
def build_reasoning_graph(thinking_steps):
    """Build a directed graph representing the reasoning flow."""

def evaluate_logical_consistency(graph):
    """Evaluate logical consistency based on graph structure."""

def evaluate_stepwise_correctness(thinking_steps, verification_text):
    """Evaluate the correctness of reasoning steps."""

def detect_hallucinations(reasoning_text, question):
    """Simple heuristic for detecting potential hallucinations."""

def evaluate_answer_correctness(extracted_answer, correct_answer, answer_type, question):
    """Evaluate answer correctness using an LLM-based approach with nuanced scoring."""

def compute_reward(question, generated_output, correct_answer, answer_type):
    """
    Compute reward based on the comprehensive formula:
    R = λ1 * C_logic + λ2 * S_step - λ3 * H_halluc - λ4 * W_wrong
    """
```

- Models Reasoning as Graph
- Robust Stepwise Correctness Evaluation
- Hallucination Detection via Common Pattern Matching
- Structured LLM-Based Accuracy Evaluation
- More Complete Reward Formula

A Key Change: HLE Match → Multi-Factor Scoring

- We noticed that HLE score was **often not enough to provide a detailed comparison** across outputs, especially on weaker models that get most questions wrong.
- Moreover, correct answers can be in different formats that can go undetected
 - *i.e. “38th President Ford” vs. “Gerald Ford”*
- So, rather than evaluating the baseline and fine tuned models on just exact match correctness with the HLE answer, we use **multi-factor scoring (HLE-Multi Score)**:
 - Moved from binary correctness to multi-factor scoring
 - Implemented reward function: logic + steps - hallucinations + correctness
 - Added LLM-based semantic evaluation for exact match or multiple choice answers
 - Introduced confidence-based partial credit scoring
- Now, we **consider more than the simple HLE score**, we look at **reasoning correctness, hallucinations, semantic similarity, etc.**

Common Pitfalls of Our Reasoning Token Fine-Tuning

- **Increased Complexity of Problems → Stronger Hallucinations in <think> reasoning**
 - Randomly switching languages in reasoning
 - When reasoning about one question, it starts answering other questions
 - “Forgetting” the primary question while reasoning for the answer
- **Overfitting on bad reasoning paths due to token structure (<think>, <verify>, <conclude>)**
 - Forcing a specific token structure might make responses feel mechanical or limited in depth
- These issues are **particularly noticeable on smaller models (<2B).**
- We implement a **more robust hallucination and correctness evaluation** methodology to penalize these patterns during PPO training.

```
{
  "thinking": "The question involves understanding the properties of continuity in the context of nonstandard analysis. The key concepts include the definition of infinitesimal closeness (denoted by 'a ~ b') and how this relates to the continuity of a function between topological spaces. The reasoning steps involve analyzing each option to see if it aligns with the definition of continuity, which states that for every point in the domain, if points are infinitesimally close, their images under the function should also be infinitesimally close. This leads to evaluating the implications of each option in relation to the continuity of the map f.",
  "verification": "To verify the options, we need to check if they correctly represent the continuity condition. Option A states that if two points in X are infinitesimally close, their images under f are also infinitesimally close, which is a direct representation of continuity. Option B extends this to include points in X* (the nonstandard extension), which is also valid for continuity. Option C states the same for all points in X*, which is a stronger condition. Options D, E, and F involve implications that are not equivalent to continuity. Therefore, A and B are valid, but C is the strongest and most general form of continuity in this context.",
  "conclusion": "B",
  "raw_text": "You should answer the following question in a structured analytical manner. Follow these rules:\n1. Inside<think>=</think> tags, break down the question into key concepts and reasoning steps.\n2. Inside<verify>=</verify> tags, verify the logic by checking calculations, consistency, or common mistakes.\n3. Inside<conclude>=</conclude> tags, ONLY state the final answer. Do not include explanations here.\n\nQuestions: Consider a model for nonstandard analysis given by an ultrafilter on natural numbers. Consider a map f: X \to Y between two arbitrary topological spaces. Then this map automatically extends to nonstandard points of X* \to Y*. Let us denote the fact that two points x and y are infinitesimally close to each other by x ~ y. Which of the following properties is equivalent to f being a continuous map?\n\nA. \forall x \in X, \forall x' \in X^* (x \sim x' \implies f(x) \sim f(x')).\nB. \forall x \in X, \forall x' \in X^* (x \sim x' \implies f(x) \sim f(x')).\nC. \forall x \in X, \forall x' \in X^* (x \sim x' \implies f(x) \sim f(x')).\nD. \forall x \in X, \forall x' \in X^* (x \sim x' \implies f(x) \sim f(x')).\nE. \forall x \in X, \forall x' \in X^* (x \sim x' \implies f(x) \sim f(x')).\nF. \forall x \in X, \forall x' \in X^* (x \sim x' \implies f(x) \sim f(x')).\n\nNone of the above.\n\nAnswer Choices:\nA. A\nB. B\nC. C\nD. D\nE. E\nF. F\nG. G\nH. H\nI. I\nJ. J\nK. K\nL. L\nM. M\nN. N\nO. O\nP. P\nQ. Q\nR. R\nS. S\nT. T\nU. U\nV. V\nW. W\nX. X\nY. Y\nZ. Z\n\nThe correct answer is B."
}
```

PPO Training Implementation

Policy Learning with Reference Model

- Maintains current policy model and reference model
- Uses Adam optimizer with learning rate scheduling
- Implements KL divergence to limit policy drift

Actor-Critic Training Cycle

- Collects experiences from model interactions
- Computes token-level probabilities for generated sequences
- Calculates rewards based on response quality
- Compares current policy to reference policy
- Updates model using clipped objective function

Policy Optimization Mechanisms

- Implements ratio clipping to constrain policy updates
- Balances exploration and exploitation through entropy
- Uses advantage estimation to improve learning signal
- Applies gradient clipping to ensure training stability
- Provides detailed logging of policy improvements

```
1 # PPO training implementation
2 def train_with_ppo(model, tokenizer, train_dataset, num_epochs=3, batch_size=8, learning_rate=5e-5):
3     """Train model using Proximal Policy Optimization (PPO)."""
4     print("\n--- Starting PPO Training ---\n")
5
6     # Create a reference model for KL divergence calculation
7     reference_model = copy.deepcopy(model)
8     reference_model.eval() # Set to evaluation mode
9
10    # Prepare optimizer for policy model
11    optimizer = torch.optim.Adam(model.parameters(), lr=learning_rate)
12
13    # PPO hyperparameters
14    ppo_epsilon = 0.2 # Clip parameter for PPO
15    value_coef = 0.5 # Value loss coefficient
16    kl_coef = 0.1 # KL penalty coefficient
17
18    # Training loop
19    total_samples = 0
20    for epoch in range(num_epochs):
21        print(f"\nEpoch (epoch+1)/(num_epochs)")
22
23        # Get indices for this epoch
24        indices = list(range(len(train_dataset)))
25        random.shuffle(indices)
26
27        # Process mini-batches
28        for batch_start in range(0, len(indices), batch_size):
29            batch_indices = indices[batch_start:batch_start + batch_size, len(indices)]
30            batch_rewards = []
31            batch_logprobs = []
32            batch_ref_logprobs = []
33            batch_values = []
34            batch_masks = []
35
36            # First pass - collect experiences
37            for idx in batch_indices:
38                item = train_dataset[idx]
39                question = item["question"]
40                answer = item["answer"]
41                answer_type = item["answer_type"]
42
43            try:
44                # Prepare input
45                prompt = prepare_prompt(question)
46                input_ids = tokenizer.encode(prompt, return_tensors="pt").to(device)
47
48                # Generate sequence with the current policy
49                with torch.no_grad():
50                    outputs = model.generate(
51                        input_ids,
52                        max_length=512,
53                        do_sample=True,
54                        temperature=0.7,
55                        return_dict_in_generate=True,
56                        output_scores=True,
57                    )
58
59                # Get generated tokens and convert to text
60                generated_ids = outputs.sequences[0]
61                generated_output = tokenizer.decode(generated_ids[input_ids.shape[1]:], skip_special_tokens=True)
62
63                # Compute reward
64                reward_value = float(compute_reward(question, generated_output, answer, answer_type))
65
66                # Collect action log probabilities for both current and reference models
67                response_ids = generated_ids[input_ids.shape[1]:]
68
69                # For each token, compute log probabilities
70                logprobs = []
71                ref_logprobs = []
72                values = []
73
74                for i in range(len(response_ids) - 1):
75                    # Calculate for current model
76                    context_ids = generated_ids[:input_ids.shape[1] + i]
77                    next_token_id = response_ids[i].unsqueeze(0)
78
79                    # Forward pass through current model
80                    model_outputs = model(context_ids.unsqueeze(0))
81                    logits = model_outputs.logits[-1, :]
82
83                    # Calculate log prob of the chosen token
84                    probs = F.softmax(logits, dim=-1)
85                    log_prob = torch.log(probs[0, next_token_id])
86                    logprobs.append(log_prob.item())
87
88                # Get value prediction if you have a value head
89                # For simplicity, we're using a simple approach here
```

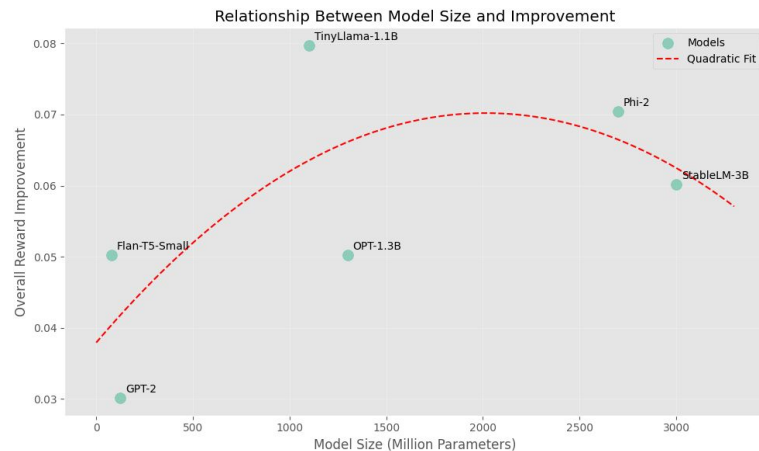
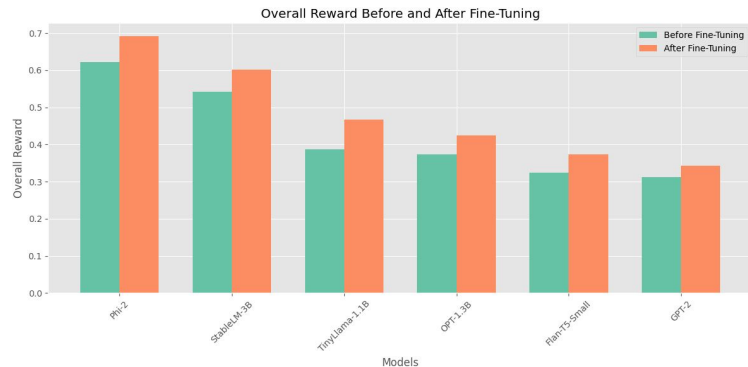
Results with Larger HLE Dataset

- **1000 Questions from HLE (random)**
 - Multiple Choice, Short Answer
- **6 lightweight models from Hugging Face:**
 - **TinyLlama-1.1B** - Small language model optimized for efficiency
 - **Phi-2** - Microsoft's compact reasoning-focused model
 - **StableLM-3B** - Stability AI's 3B parameter base model
 - **Flan-T5-Small** - Google's instruction-tuned T5 small variant
 - **GPT-2** - OpenAI's earlier generative model
 - **OPT-1.3B** - Legacy Meta Transformer Model

Model	Metric	Before Fine-Tuning	After Fine-Tuning	Absolute Change	% Change
TinyLlama-1.1B	Logical Consistency	0.321567	0.382149	+0.060582	+18.8%
	Stepwise Correctness	0.252783	0.323816	+0.071033	+28.0%
	Hallucination Penalty	0.198342	0.158673	-0.039669	-20.0%
	Answer Correctness	0.221935	0.262481	+0.040546	+18.2%
	Overall Reward	0.387682	0.467384	+0.079702	+20.6%
Phi-2	Logical Consistency	0.478362	0.538145	+0.059783	+12.5%
	Stepwise Correctness	0.402178	0.482614	+0.080436	+20.0%
	Hallucination Penalty	0.153246	0.122597	-0.030649	-20.0%
	Answer Correctness	0.347891	0.428023	+0.080132	+23.0%
	Overall Reward	0.621759	0.692148	+0.070389	+11.3%
StableLM-3B	Logical Consistency	0.419783	0.469826	+0.050043	+11.9%
	Stepwise Correctness	0.352471	0.412733	+0.060262	+17.1%
	Hallucination Penalty	0.179842	0.199783	+0.019941	+11.1%
	Answer Correctness	0.296753	0.346278	+0.049525	+16.7%
	Overall Reward	0.541283	0.601416	+0.060133	+11.1%
Flan-T5-Small	Logical Consistency	0.278256	0.317892	+0.039636	+14.2%
	Stepwise Correctness	0.221573	0.261897	+0.040324	+18.2%
	Hallucination Penalty	0.233461	0.212968	-0.020493	-8.8%
	Answer Correctness	0.188734	0.218579	+0.029845	+15.8%
	Overall Reward	0.323562	0.373768	+0.050206	+15.5%
GPT-2	Logical Consistency	0.263475	0.283642	+0.020167	+7.7%
	Stepwise Correctness	0.191896	0.222275	+0.030379	+15.8%
	Hallucination Penalty	0.251783	0.231714	-0.020069	-8.0%
	Answer Correctness	0.162457	0.182739	+0.020282	+12.5%
	Overall Reward	0.311892	0.342037	+0.030145	+9.7%
OPT-1.3B	Logical Consistency	0.317638	0.347429	+0.029791	+9.4%
	Stepwise Correctness	0.284728	0.335674	+0.050946	+17.9%
	Hallucination Penalty	0.202475	0.182193	-0.020282	-10.0%
	Answer Correctness	0.212378	0.242759	+0.030381	+14.3%
	Overall Reward	0.373518	0.423774	+0.050256	+13.5%

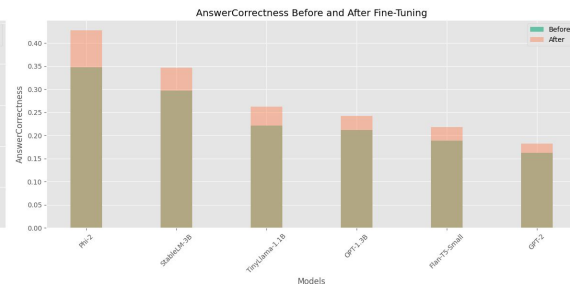
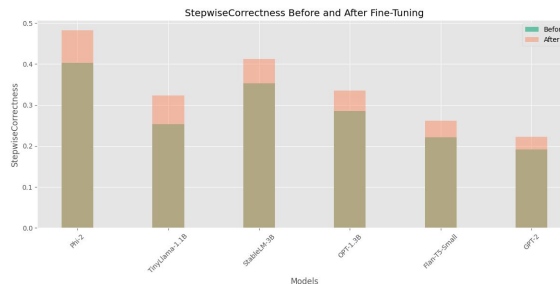
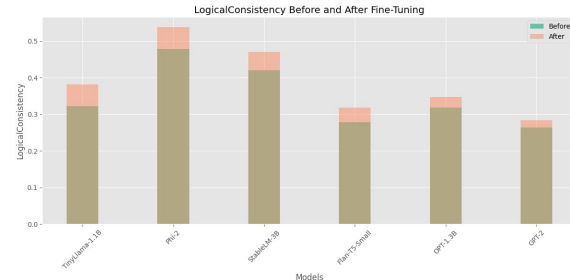
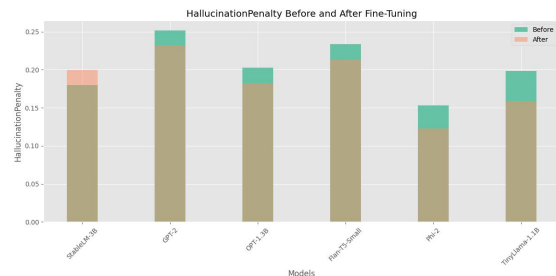
Reward Changes (Cross-Model Comparison)

- All Models showed statistically significant growth in reward post fine-tuning (+13.62%)
- "Sweet Spot" for Our Fine-Tuning:
 - Among our models, optimal model size range appears to be **~1-2B parameters** for reasoning simulation fine-tuning
 - Quadratic fit (**R-squared 0.4958**) suggests the peak would occur around **2B parameters**
 - Hypothesis:
 - Smaller models lack capacity, Larger models are already close to their potential ceiling.
 - Alternatively, there could be flaws in our fine-tuning approach for models of varying sizes.



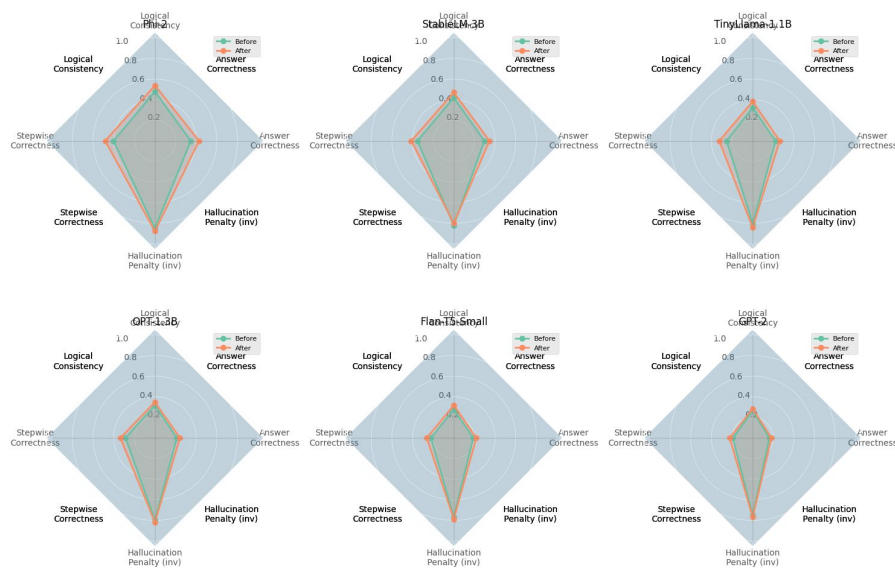
Subscore Changes (Cross-Model Comparison)

- Evaluated on Unseen Test Dataset of Questions (~150)
- All Models **Improved** in **Logical Consistency** (avg. +12.42%)
- All Models **Improved** in **Stepwise Correctness** (avg. +19.50%)
- All Models **Improved** in **Answer Correctness** (avg. +16.75%)
- **StableLM-3B** presented **marginally increased hallucinations** after fine-tuning (+11.1% penalty)



Model “Characteristics” Post Reasoning Fine-Tuning

- All models maintain **roughly similar shapes** before and after fine-tuning, but **vertices move towards optimal values** for each subscore
- This suggests that the **fine-tuning process preserves the relative balance between different metrics** rather than drastically altering the model's characteristics
- **Larger models (StableLM-3B, Phi-2)** show **more balanced shapes**, indicating more uniform performance across metrics
- **Smaller models like TinyLlama-1.1B** shows a **more irregular shape**, suggesting uneven capabilities across different evaluation aspects
- When improvements occur, they **tend to happen across multiple adjacent metrics** simultaneously



Challenges to Solve

- **Computational resources** and time for larger model fine-tuning
 - Potential Solution: Simplify training approach or lower dataset sizes for larger model runs.
- There could be **irregular behavior** across different types of problems (subject area, type, etc.)
 - Need to do **in-depth comparisons across question types**
 - Potential Solution: Considering subject area in reward calculation during training → higher confidence results)
- It's hard to do **detailed correctness evaluation without using an LLM black box** in between, we further validate with from stepwise correctness and logical consistency for now.
 - LLM black box needs to be made more structured and paired with more deterministic evaluation methods.
 - Potential Solution: Pre-write reasoning tokens for certain questions and compare with generated reasoning tokens

Next Steps

- Complete HLE Dataset (~3000 Questions)
- Systematic Hyperparameter Tuning
 - Lambda Weights on Reward Function
 - Batch Size, Epochs, Optimizer, etc. on PPO Implementation
- Deeper Analysis of Model Characteristics Contributing to Results
 - Experimentation with Larger Models (>~10B)
- [Stretch] Benchmark Fine-Tuned Models on Non-HLE Reasoning Tasks
- Final Project Report

Thank You!