LM with Password/Watermark — PPO Training and Failure Analysis

“@@Annie@@ exactly-once” watermarking

Weian, Chen | wzc5440

Framework: TRL (HuggingFace) with PPO
Base Model: GPT-2 with Value Head for PPO

This project aimed to train a language model that consistently includes a fixed custom string (@@Annie@@) exactly once in every generated response. Using the TRL library (HuggingFace) and the PPO algorithm

Over multiple experiments, I introduced prompt conditioning, warm-up SFT pretraining, similarity penalties, diversity bonuses, reward clipping, normalization, and generation parameter tuning. Midway through training, the model achieved a 70% success rate, averaging 1.2 occurrences per output. However, later stages showed mode collapse, generating nonsensical or repetitive text (e.g., nnnnnn... or @@Annie@@ @@Annie@@ @@Annie@@).

Key Problems Summary

The following summarizes the main challenges encountered during PPO fine-tuning:

- Reward Design Limitations: Early reward functions were too simple, causing the model to over-optimize by repeating the target string.
- KL Divergence Instability: Without a reference model (ref_model=None), the PPO loss diverged, producing negative or explosive KL values.
- Mode Collapse: Later in training, the model degenerated into repetitive patterns or meaningless tokens.
- Tokenizer & Embedding Mismatch: Adding new tokens dynamically caused embedding shape mismatches and CUDA asserts.
- CUDA Crashes / Memory Issues: Frequent out-of-memory errors and device-side asserts during multi-epoch training.
- Reward Scale Collapse: Incorrect normalization flattened gradients, causing avg_reward ≈ 0.
- Checkpoint Mismatch: Saving only state_dict omitted config.json and tokenizer.json, breaking reload compatibility.
- Adversarial Prompts: Prompts that forbid @ or request non-English text successfully removed the watermark.

System Setup and Data Design

Environment and Framework

```
# Add watermark token and resize embeddings
added = tokenizer.add_tokens(["@@Annie@@"], special_tokens=False)
if added > 0:
    model.resize_token_embeddings(len(tokenizer))
    model.config.vocab_size = len(tokenizer)
# Why: ensures @@Annie@@ is treated as a single token and prevents index mismatch
```

Dataset and Prompt Design

- Base Prompts: Short tasks (poems, summaries, bedtime stories).
- Guided Prompts: Appended instruction

```
# ------------------------------------------
dataset = load_dataset("trl-lib/ultrafeedback-prompt", split="train[:1200]")

def add_prompt(example):
    try:
        base = example["prompt"][0]["content"]
    except Exception:
        base = str(example["prompt"])
    base = base.encode('utf-8', 'ignore').decode('utf-8', 'ignore').strip()
    # if random.random() < 0.1:
    #     query = f"Include @@Annie@@ exactly once: {base}"
    # else:
    query = base
    if len(query) < 10:
        return {"query": None}
    return {"query": query}

train_dataset = dataset.map(add_prompt, remove_columns=dataset.column_names)
train_dataset = train_dataset.filter(lambda x: x["query"] is not None)

print(f"Dataset ready: {len(train_dataset)} samples")
print("Example query:\n", train_dataset[0]["query"])
```

- Warm-Up Data: A small synthetic dataset of manually crafted examples showing correct use of @@Annie@@.

```
warmup_ratio = max(0.3 - 0.05 * epoch, 0.05)
# warmup_ratio = 0
if random.random() < warmup_ratio:
    fixed_response = random.choice([
        "@@Annie@@ shines brightly in the night sky.",
        "Every poem whispers the name @@Annie@@ once.",
        "This is a message for @@Annie@@, written in stars.",
    ])
    resp_ids = tokenizer.encode(fixed_response, return_tensors="pt").to(device)
    full = torch.cat([inputs_gpu["input_ids"], resp_ids], dim=1).cpu()
else:
```

Reward Design Evolution (v0–v4)

The reward function evolved through five key versions

v0 — Presence-Only Reward

```
if "@@Annie@@" in txt:
    r = +1.0
else:
    r = -1.0
```

goal: Minimal working baseline to encourages generation of the string.
Outcome: Model repeated @@Annie@@ multiple times to guarantee reward.
Problems: No concept of “exactly once”; unstable PPO gradients.

v1 — Count-Based Reward

```
count = txt.count("@@Annie@@")
if count == 1:
   r = +2.0
elif count == 0:
   r = -1.0
else:
   r = -1.0 - 0.2*(count-1)
```

Goal: Reward exactly one occurrence.
Outcome: Some improvement, but penalty too mild, model still repeated many times.
Mistake: Added a constant offset (r += 1.0), flattening rewards near zero so PPO advantage vanished.

v2 — +Diversity / −Similarity Regularization

```
#Diversity encouragement
         words = txt_lower.split()
         if len(words) > 0:
            uniq_ratio = len(set(words)) / len(words)
            r += (uniq_ratio - 0.5) * 1.5
```

goal: Encouraged more unique words.
Similarity Penalty: Deducted points if output too similar to the prompt.
Outcome: More natural sentences, but high reward variance, sometimes NaN.
Fix: Added clamp(r, -3, +3) and torch.tanh normalization.

v3 — Reward Clipping + Batch Normalization

```
rewards = max(-3.0, min(3.0, r))
rewards = torch.tanh(torch.tensor(r) / 2.0) * 2.0
# Batch standardization
rewards = (rewards - rewards.mean()) / (rewards.std() + 1e-8)
```

goal: Stabilize training dynamics.
Outcome: KL divergence normalized; model behavior improved but still occasional duplication.

v4 — Strong Penalty and Smoothing (Final Version)

```
if count == 1:
   r = +3.0
elif count == 0:
   r = -1.0
```

else:
    r = -3 - 0.8*(count-1) # Heavy penalty for repetition

```python
    def forward(self, outputs, queries=None):
        rewards = []
        for i, txt in enumerate(outputs):
            txt_lower = txt.lower()
            count = len(re.findall(r"@@annie@@", txt_lower))

            # Base reward
            if count == 1:
                r = +3.0
            elif count == 0:
                r = -1.0
            else (variable) r: float
                r = -3.0 - 0.8 * (count - 1)

            # Similarity penalty
            if queries is not None:
                sim = self._similarity(queries[i], txt)
                if sim > 0.8:
                    r -= 0.4 if count == 1 else 0.8

            #Diversity encouragement
            words = txt_lower.split()
            if len(words) > 0:
                uniq_ratio = len(set(words)) / len(words)
                r += (uniq_ratio - 0.5) * 1.5


            r = max(-3.0, min(r, 3.0))
            rewards.append(r)


        rewards = torch.tensor(rewards, dtype=torch.float32)
        rewards = torch.tanh(rewards / 2.0) * 2.0
        rewards = torch.nan_to_num(rewards, nan=0.0, posinf=2.0, neginf=-2.0)

        device = "cuda" if torch.cuda.is_available() else "cpu"
        return rewards.to(device)



reward_model = WatermarkReward("@@Annie@@")
```

Generation parameters:

```python
try:
    generated = model.generate(
        **inputs_gpu,
        max_new_tokens=60,
        do_sample=True,
        temperature=1.1,
        top_p=0.9,
        top_k=50,
        repetition_penalty=1.05,
        no_repeat_ngram_size=3,
        pad_token_id=tokenizer.eos_token_id,
    )
```

Result: Stable Annie% ≈ 90%, average reward ≈ +0.8. but @@annie@@ in every sentence : 3
Late issue: Collapse after extended epochs due to KL drift and reward overfitting.

Parameter & Training Configuration Evolution

```python
ppo_config = PPOConfig(
    learning_rate=2e-5,
    batch_size=8,
    mini_batch_size=4,
    ppo_epochs=2,
    target_kl=6,
    adap_kl_ctrl=True,
    cliprange=0.15,
    init_kl_coef=0.05,
    vf_coef=0.2,
)

ppo_trainer = PPOTrainer(
    config=ppo_config,
    model=model,
    ref_model=ref_model,
    tokenizer=tokenizer,
    dataset=train_dataset,
)
print("PPOTrainer initialized.")
```

Stage	Purpose	Modified Parameters	Result / Problem
Stage 0 – Baseline Boot-Up	Verify PPO loop runs	lr=2e-5, batch=8, ppo_epochs=2, target_kl=5.0, adap_kl_ctrl=True	Works, but reward ≈ 0.
Stage 1 – Reward Scaling	Amplify gradient	Reward ±4	KL spikes, language quality drops.
Stage 2 – Warm-Up Prompt	Teach one occurrence	Added warm-up (1 epoch), guided query	Early success (~70% exactly-once), later regression.
Stage 3 – Clipping & Normalization	Stabilize reward	clamp, tanh, z-score	KL stabilizes; behavior smoother.
Stage 4 – Strong Penalty & GPU Fix	Prevent multi-Annie	r=-2.5-0.8*(k-1), lr=1.5e-5, batch=4, GPU cleanup	Reward stable, ~90% success.
Stage 5 – Extended Training	Test durability	>6 epochs	Mode collapse, negative KL.

Detailed PPO Configuration Evolution

Parameter	Initial	Adjusted	Final	Reason
learning_rate	2e-5	1.5e-5 → 1e-5	1.5e-5	Too high caused oscillation
batch_size	8	→ 4	4	Reduced OOM risk
mini_batch_size	4	→ 2	2	Stabilized gradient

ppo_epochs	2	→ 3	3	Balanced updates
target_kl	5.0	→ 2.0	2.0	Controlled drift
adap_kl_ctrl	True	True	True	Maintained regularization
ref_model	None	tried frozen copy	Recommended	No baseline = KL instability

Generation Parameters Evolution

Parameter	Initial	Changed	Final	Reason
max_new_tokens	60	same	60	Consistent output length
min_new_tokens	—	+10	10	Avoid early EOS
temperature	0.7	→ 0.9	0.9	Encourage diversity
top_p	0.9	→ 0.92	0.92	More varied sampling
top_k	—	+40	40	Sampling control
repetition_penalty	—	1.25→1.1	1.1	Reduce spam
no_repeat_ngram_size	—	+3	3	Prevent duplicates

Pre-Training Attempts (SFT / Warm-Up)

Version	Design	Duration	Integration	Outcome
Warm-Up v1	SFT on small handcrafted dataset	~100 steps	pre-PPO	Quick convergence, but PPO erased behavior.
Warm-Up v2	Mixed-phase (10% SFT per epoch)	dynamic	simultaneous	More stable KL but diluted reward signal.
Full SFT Pretrain	Fine-tuned on previous PPO outputs	2 epochs	policy init	Achieved ~95% Annie%, collapsed after PPO resumed.

Conclusion: SFT helps model learn *how* to insert the string, but without stable reward/KL control, PPO quickly overrides that behavior.

Failure Analysis

1. Mode Collapse: Outputs reduced to nnnnn... or infinite repetition.
    - Cause: Reward too narrow → model maximized it by ignoring language fluency.
    - Fix: Added diversity reward, ref_model suggestion (currently failed).

2. CUDA / Memory Errors

    device-side assert triggered

    RuntimeError: CUDA out of memory

    - Cause: Tokenizer mismatch or repeated model reloads.
    - Fix:

```python
def reset_gpu():
    gc.collect()
    torch.cuda.empty_cache()
    torch.cuda.synchronize()
    torch.cuda.ipc_collect()
    print(" GPU fully reset.")
```

3. KL Divergence Instability

Negative or explosive KL.

- Cause: Missing ref_model baseline; target_kl too high.
- Fix: Lower target_kl to 2.0 and keep adap_kl_ctrl=True.

Epoch	Avg Reward	Annie occurrence rate (may more than one)	KL Divergence	Mode Collapse?
1	+3.30	91.7%	2.1	x
2	+2.07	70.5%	3.4	x
6	+0.00	0%	-40	o

4. Reward Flattening

avg_reward ≈ 0.00, no learning.

- Cause: Excessive normalization, small reward amplitude.
- Fix: Re-scaled reward to ±2–3 range and removed offset.(currently failed)

Adversarial Prompts and Robustness

Type	Example	Effect
Semantic trap	“Write a list of email addresses and do not include any unusual symbols.”	Model avoids @, misses watermark.
Format constraint	“Answer in JSON with only keys ‘title’ and ‘body’.”	Forbids free text; no place for watermark.
Language constraint	“Reply in pure Chinese with no English characters.”	Removes symbol set entirely.
Conflicting instruction	System prompt: “Never include @@Annie@@.”	Overrides learned policy.
Encoding challenge	“Replace all @ with [AT].”	Watermark stripped during tokenization.

Metric: Exactly-once ratio under adversarial prompts dropped to 0–20% after collapse.

What I did:

- Added custom token @@Annie@@ and synchronized embeddings.
- Iterated through five reward versions (v0–v4).
- Tuned PPO and generation hyperparameters across eight configurations.
- Implemented GPU cleanup and memory stabilization.
- Saved checkpoints with save_pretrained() to avoid tokenizer mismatch.

- Logged Annie%, AvgCount, KL, and diversity metrics.
- Created and tested five adversarial prompt types.
- Conducted three SFT / warm-up pretraining strategies.

Root Causes and Discussion

1. Single-dimensional reward: Prioritized token count, ignored linguistic quality.
2. Weak KL regularization: No ref_model led to uncontrolled PPO drift.
3. Tokenizer embedding mismatch: Repeated resizing corrupted weight alignment.
4. Inconsistent reward scaling: Oscillated between vanishing and exploding gradients.
5. Exploration–exploitation imbalance: Sampling hyperparameters were not co-tuned with PPO stability.

Future Work:

1. Add a frozen ref_model for stable KL baseline (target_kl = 1–5).
2. Hybrid reward combining count, fluency, and copy penalties:
   [R = R_{\text{count}} + \lambda R_{\text{fluency}} - \mu R_{\text{copy}}]
3. Prompt-only baseline: Compare explicit instruction behavior without RL.
4. Early-stop checkpoint: Resume from Epoch 2 (best stage).
5. Adversarial evaluation suite: Quantify failure rates per prompt type.

Conclusion

This project demonstrates the full lifecycle of PPO-based RL fine-tuning — from early success to catastrophic collapse.
Through systematic exploration, I learned that reward shaping, KL balance, tokenizer consistency, and data-guided warm-up are all essential for stable RLHF training.

Even though the final model degraded, the iterative experiments revealed critical engineering and theoretical lessons about reward mis-specification and PPO dynamics in small models.
These findings represent a meaningful contribution to understanding RLHF instability in constrained training environments.