Exploring RL-based LLM Training for Formal Language Tasks with Programmed Rewards

Alexander G. Padula[1,2] and Dennis J.N.J. Soemers[2]

[1] Department of Computer Science, ETH Zurich
[2] Department of Advanced Computing Sciences, Maastricht University
apadula@student.ethz.ch, dennis.soemers@maastrichtuniversity.nl

Abstract. Proximal Policy Optimization (PPO) is commonly used in Reinforcement Learning from Human Feedback to align large language models (LLMs) with downstream tasks. This paper investigates the feasibility of using PPO for direct reinforcement learning (RL) from explicitly programmed reward signals, as opposed to indirect learning from human feedback via an intermediary reward model. We focus on tasks expressed through formal languages, such as mathematics and programming, where explicit reward functions can be programmed to automatically assess the quality of generated outputs. We apply this approach to a sentiment alignment task, a simple arithmetic task, and a more complex game synthesis task. The sentiment alignment task replicates prior research and serves to validate our experimental setup. Our results show that pure RL-based training for the two formal language tasks is challenging, with success being limited even for the simple arithmetic task. We propose a novel batch-entropy regularization term to aid exploration, although training is not yet entirely stable. Our findings suggest that direct RL training of LLMs may be more suitable for relatively minor changes, such as alignment, than for learning new tasks altogether, even if an informative reward signal can be expressed programmatically.

Keywords: Reinforcement Learning · Large Language Models · Formal Languages.

1 Introduction

Program synthesis models such as Copilot [9] have quickly become indispensable by improving productivity and making complex tasks more accessible. Significant advancements in this field have been achieved by training general-purpose Large Language Models (LLMs) to reproduce source code from widely used programming languages. Current state-of-the-art coding models [16,18,26] are trained using an auto-regressive next-token prediction objective, which maximizes the probability of predicting the next token in a sequence. Despite their success across a wide range of language processing tasks, next-token prediction suffers from the inherent limitation of being a surrogate objective, which can at times diverge from a task's true goals. When multiple valid solutions exist

for a task, such as implementing a function, next-token prediction penalizes the model for deviating from solutions that are overrepresented in the training data, even if other solutions may be equivalent or superior. Moreover, coding models trained with a next-token objective are not grounded in the outcomes of executing the code they generate. This disconnect can exacerbate existing tendencies to produce near misses, where the generated code superficially resembles a correct solution but contains subtle errors that prevent successful execution [9,4].

Reinforcement Learning (RL) [30] emerges as a natural paradigm to ground models by training directly on a task's true goals. However, in many (natural) language tasks it is often difficult to explicitly program a reliable reward signal to quantify useful properties (e.g., a response's helpfulness or accuracy). RL from Human Feedback (RLHF) circumvents this challenge by training an intermediary critic model on a limited number of human evaluations collected from users or reviewers [34]. The trained critic model produces rewards as evaluations of the base model's outputs, enabling the use of RL to further train the model so as to generate outputs that maximize the intermediate rewards. Despite its complexity, RLHF has become a popular and effective method for aligning pre-trained LLMs with downstream tasks [22,23,21].

In the domain of programming, and more broadly structured languages, however, there is a distinctive opportunity to employ a more direct and contextually appropriate training objective. Unlike natural language, code can be executed, and its effects can be automatically evaluated and compared, offering a pathway to explicitly programmed domain-specific reward functions. This eliminates the need for human-in-the-loop evaluations and reward models, aligning the training process more closely with how humans experiment and learn coding through a continuous process of trial and error. This paper explores the idea of exploiting this unique aspect of structured languages to train LLMs using reward signals obtained from explicitly programmed functions as a direct training objective.

Recent research on applying RL to LLMs [15,29,32] has often relied on custom implementations. In contrast, this paper makes only minimal adjustments to the existing RLHF implementation in Hugging Face's Transformers Reinforcement Learning (TRL) library [33]. This approach aims to simplify reproducibility by utilizing an established deep learning ecosystem.[3][4]

2 Background

Auto-regressive text generation tasks, such as program synthesis, can be modeled, following the standard RL problem formulation, as a finite-horizon Markov Decision Process (MDP). At any time step t, a state $s_t \in \mathcal{S}$ from a state space $\mathcal{S}$ is characterized by the sequence of non-masked tokens from the beginning of the sequence up to t. This representation captures the necessary context for subsequent token generation. The action $a_t \in \mathcal{A}$ from an action space $\mathcal{A}$ at time

[3] TRL fork with (batch-)entropy regularization: https://github.com/PadLex/trl.

[4] Source code for experiments can be found at: https://github.com/PadLex/Reinforcement-Learning-from-Explicitly-Programmed-Reward-Signals/tree/main.

t corresponds to selecting the next token to add from a predefined vocabulary, extending the current state s_t by one token. A policy π_θ, parameterized by tunable parameters θ, outputs a probability distribution over $\mathcal{A}$ conditioned on the current state s_t. During training, the parameters θ are adjusted to maximize observed rewards. A reward function R provides a scalar signal $R(\tau)$ for a trajectory $\tau = (s_1, a_1, s_2, a_2, \ldots, s_n, a_n)$. Given the auto-regressive nature of the task, where the final state s_n encapsulates the entire sequence, the reward can also be expressed solely in terms of s_n as $R(s_n)$. In auto-regressive text generation, a discount factor of 1 (i.e., no discounting) is typically used as the MDP is finite and there is no explicit preference for shorter solutions.

The field of RL [30] develops algorithms that update a policy's parameters based on experience, so as to maximize the rewards collected by the policy in future trajectories. Policy gradient methods define a differentiable objective function $L(\theta)$, which can be maximized using optimizers such as Adam or Stochastic Gradient Ascent to improve the policy's performance. Proximal Policy Optimization (PPO) [28] has become the de facto standard for RL-based training of LLMs following its use in RLHF. Schulman et al. [28] originally proposed two variants of PPO, both aiming to maximize the surrogate objective $L^{CPI}(\theta)$:

$$r_t(\theta) = \frac{\pi_\theta(a_t|s_t)}{\pi_{\theta_{\text{old}}}(a_t|s_t)}, \qquad L^{CPI}(\theta) = \hat{\mathbb{E}}_t[r_t(\theta)\hat{A}_t],$$

where the expectation estimator $\hat{\mathbb{E}}_t[\ldots]$ measures the empirical average over a finite batch of samples. The ratio $r_t(\theta)$ moderates the extent of the policy updates based on how much more or less likely the action a_t is under the new policy π_θ compared to the previous policy $\pi_{\theta_{\text{old}}}$. Compared to a traditional policy gradient objective [1], this surrogate objective gives more conservative updates and generally leads to a more stable learning process [28]. The advantage $\hat{A}_t$ estimates the relative benefit of taking the action a_t compared to all other actions available in s_t, weighted by their probability under π_θ. It is used to isolate the effect of specific actions from the general quality of the states in which they are taken. Using the Bellman equation [5,30], we can express the advantage as $\hat{A}_t = R_t + V(s_{t+1}) - V(s_t)$, where $V(s)$ is the value (expected sum of future rewards) of a state s. A value function (e.g., neural network) trained to minimize the mean squared error between predicted and observed values estimates V.

While both variants of PPO aim to maximize $L^{CPI}(\theta)$, they differ in the constraints that they employ to avoid overly aggressive gradient updates. The *Clipped Surrogate Objective* variant of PPO disincentivizes $r_t(\theta)$ from moving outside of the interval $[1-\epsilon, 1+\epsilon]$:

$$L^{CLIP}(\theta) = \hat{\mathbb{E}}_t\left[\min\left(r_t(\theta)\hat{A}_t, \text{clip}(r_t(\theta), 1-\epsilon, 1+\epsilon)\hat{A}_t\right)\right]$$

The *Adaptive KL Penalty Coefficient* variant of PPO penalizes large changes to the policy $\pi_\theta(a_t \mid s_t)$ by measuring the Kullback–Leibler (KL) divergence [14] between $\pi_{\theta_{\text{old}}}(a_t \mid s_t)$ and $\pi_\theta(a_t \mid s_t)$:

$$L^{KLP}(\theta) = \hat{\mathbb{E}}_t\left[r_t(\theta)\hat{A}_t - \beta_{\text{KL}} KL[\pi_{\theta_{\text{old}}}(a_t \mid s_t), \pi_\theta(a_t \mid s_t)]\right]$$

An *entropy regularization* term L^{ENT} is often added to the objective function to encourage exploration and smoothen the optimization landscape [3]:

$$L^{ENT}(\theta) = -\beta_{\text{ENT}}\hat{\mathbb{E}}_t \left[\sum_{a \in \mathcal{A}} \pi_\theta(a \mid s_t) \log \pi_\theta(a \mid s_t) \right] \tag{1}$$

We use the TRL implementation [33] of PPO as our baseline, which is based on a variant of PPO that has become the standard for RLHF [34]. It differs from common (non-RLHF) implementations by using both a Clipped Surrogate Objective and an Adaptive KL Penalty, and dropping the entropy regularization term. Furthermore, its KL divergence is relative to the initial policy, instead of the policy at the previous training step. This is because in RLHF, the initial policy (usually a foundation model pre-trained on a substantial amount of data) is assumed to be close to the final one, and care must be taken to ensure that the policy does not fool the reward model by generating out-of-distribution samples.

3 Batch-Entropy Regularization

The standard entropy regularization term of Equation (1) encourages exploration by penalizing the policy for drifting away from the uniform policy on the basis of individual states: within a training batch, every state with a highly non-uniform policy receives a sizeable penalty. However, a potential failure mode of RL-based training may not just be a lack of exploration on a per-state basis, but rather a lack of exploration across the state space. A strong policy will often need to put most of its probability mass on a single (best) action per state—a solution that conflicts with the standard entropy regularization—but would typically be expected to still pick different actions across different states.

We propose a novel *batch-entropy regularization* term, designed to encourage the policy to choose diverse actions for different states within the same batch, without penalizing it for having a highly non-uniform policy for individual states:

$$L^{BENT}(\theta) = -\beta_{\text{BENT}} \sum_{a \in \mathcal{A}} \hat{\mathbb{E}}_{s_t \sim B} \left[\pi(a \mid s_t) \right] \log \hat{\mathbb{E}}_{s_t \sim B} \left[\pi(a \mid s_t) \right], \tag{2}$$

where $\hat{\mathbb{E}}_{s_t \sim B}$ denotes the empirical mean over all states s_t in a batch B. Such a batch-entropy term was previously used to analyze and evaluate the behavior of trained RL models [10], but our use as a regularization term is novel.

4 Language Model Fine-tuning Tasks

In this paper, we consider three different RL-based fine-tuning tasks for language models. Firstly, a sentiment alignment task (Subsection 4.1)—for which successful results are known to be feasible from prior work [12]—is used to verify the correctness and compatibility of our TRL-based implementation and experiment setup. Secondly, a synthetic arithmetic task (Subsection 4.2) is used as a formal

language task which is simple enough to rapidly generate substantial training data. Thirdly, we consider the complex task of synthesising (novel) games in Ludii's formal game description language [7,24] (Subsection 4.3). This task is particularly compelling due to the scarcity of training samples—which limits the effectiveness of supervised learning—and the availability of established reward metrics to assess the quality of newly generated Ludii games [31].

4.1 Sentiment Alignment Task

We initially seek to replicate results from prior research [12] in order to verify that the models we employ are compatible with TRL and confirm that the modifications we make to TRL——adding entropy and batch-entropy losses, removing the KL penalty, and replacing the trained reward model with a programmed reward function——do not compromise the integrity of the experimental setup.

The task is to fine-tune GPT-2 [25] to generate positive movie reviews using RL. Initially, GPT-2 is pre-trained using a conventional masked language modeling (MLM) objective on the Stanford IMDB dataset [20]. Then, using PPO, the model is trained to complete reviews from the dataset while imbuing them with a positive sentiment. As part of the RLHF process, generated samples are evaluated by a reward model. For this purpose, the research we are reproducing employs a variant of DistilBERT [27] that was fine-tuned on user-labeled reviews in the IMDB dataset. As an alternative training method, we also replace the conventional reward model with an automated signal. For this, we use the VADER [13] implementation from NLTK [6], a rule-based sentiment analysis algorithm that returns a score between -1 and 1, which we use as a reward signal that quantifies how negative or positive the generated reviews are.

4.2 Synthetic Arithmetic Task

We define a simple arithmetic task designed to elucidate the potential advantages offered by RL-based training over a traditional MLM objective. In this task, $n = 5$ coefficients, $c_1, c_2, \ldots, c_n$, are independently and uniformly drawn from the set of integers $\{0, 1, \ldots, 9\}$. These coefficients are summed to form an initial expression, $Y_0 = c_1 + c_2 + \ldots + c_n$. The task simplifies Y_0 through a series of n steps, where at each step i, two randomly chosen, non-simplified terms from Y_{i-1} are resolved (i.e., added together), resulting in Y_i. This process is repeated until the final expression, Y_n, is a single integer: the sum of all original coefficients.

Importantly, due to the random order of summations, the sequence of intermediate expressions $Y_1, Y_2, \ldots, Y_n$ is non-deterministic. This randomness restricts the effectiveness of an imitation learning strategy in minimizing its loss, as it cannot leverage consistent sequential dependencies typically exploited in MLM tasks. For example, the sum $6 + 10 + 7 + 1 + 3$ might first simplify to $6 + 17 + 1 + 3$, then to $23 + 1 + 3$, followed by $23 + 4$, and finally to 27, with each step involving the addition of randomly selected terms from the previous expression. This task has properties that mirror program synthesis, where multiple equally valid solutions exist and their correctness can be quantified.

The reward function for RL-based training is designed to quantify the accuracy of the model's generated expressions relative to the target expressions. It assigns a scalar reward based on the absolute difference between the summed value of the generated expression G_i and the correct expression Y_i:

$$R(G_i, Y_i) = \frac{2}{1 + \exp\left(\frac{|G_i - Y_i|}{10}\right)}$$

The reward is 0 in cases where G_i is an invalid expression. This function ensures that smaller errors lead to higher rewards, and significant errors, particularly from invalid expressions, result in low rewards. The offset sigmoid function ensures that the reward scales smoothly between 0 and 1, providing a non-flat reward landscape even early on in training. Note how teacher forcing prevents the accumulation of errors between, but not within, expressions. In other words, G_{i-1} is discarded and the model is instead shown Y_{i-1} when computing G_i. So if the model makes a mistake when generating G_{i-1}, that mistake will not carry over when it prompted to generate G_i. In contrast to MLM objectives, this reward signal is invariant to changes in the order in which terms are summed.

4.3 Ludii Game Synthesis Task

Ludii [24] is a general game playing system with a domain-specific language (DSL) for describing rules of games [7]. Any description of rules in this language can be compiled into a runnable game by the system. This DSL describes games as trees of *ludemes*, which are high-level keywords corresponding to common board game concepts such as *board*, *is empty*, *is line*, *step*, *slide*, and so on. For an example, see the game description for the connection game *Hex*:

```
(game "Hex"
  (players 2)
  (equipment {
      (board (hex Diamond 11))
      (piece "Marker" Each)
      (regions P1 {(sites Side NE) (sites Side SW)})
      (regions P2 {(sites Side NW) (sites Side SE)})
  })
  (rules
    (play (move Add (to (sites Empty))))
    (end (if (is Connected Mover) (result Mover Win)))
  )
)
```

Generating games in this DSL is ideally suited to exploring how a direct RL process can overcome limitations arising from limited data availability. Although board game representations in this DSL are succinct enough to fit within the context length of modern LLMs and can be directly compiled into fully playable

and testable games, there are only in the order of 1000 existing board games implemented in the Ludii DSL. The scarcity of available data makes it challenging to train LLMs to learn Ludii using traditional supervised training methods.

To ease the model into learning the Ludii DSL, we define a fill-in-the-middle task. In this task, uniformly randomly sampled parentheticals are removed from game descriptions, and the model is trained to generate the missing sections. In this way, the dataset will range from simple prompts requiring the model to only fill in a small portion of a game, all the way to requiring the model to complete a whole game from scratch when the root parenthetical is sampled. The following example shows pre- and suffixes for the *Hex* game description, with the final part of the `equipment` section of the description having been removed:

```
### PRE:
(game "Hex"
  (players 2)
  (equipment {
      (board (hex Diamond 11))
      (piece "Marker" Each)
      (regions P1 {(sites Side NE) (sites Side SW)})

### SUF:
  })
  (rules
    (play (move Add (to (sites Empty))))
    (end (if (is Connected Mover) (result Mover Win)))
  )
)
```

While the quality of a game description is more challenging to objectively quantify than the correctness of, e.g., a simple program or a solution for the arithmetic task, it is still possible to program a reasonable reward function. Inspired by fitness functions used by prior work on evolutionary game generation [8,31], we use a reward function based on the following five criteria:

1. **Compilability** $C : S \mapsto \{0, 1\}$: A binary signal indicating whether the game compiles, i.e., whether the game is syntactically valid and avoids semantic errors such as using a piece that was not defined in the `equipment` ludeme.
2. **Playability** $P : S \mapsto \{0, 1\}$: A binary signal indicating whether or not moves can be made without crashing.
3. **Balance** $B : S \mapsto [0, 1]$: A continuous signal defined as the largest difference in winrates between any pair of players. For example, it returns 1 if all players won the same number of games, and 0 if one player won them all.
4. **Completion Rate** $F : S \mapsto [0, 1]$: The fraction of games that terminated within 500 turns.
5. **Decisiveness** $D : S \mapsto [0, 1]$: The fraction of games that did not end in a draw. It returns 1 if all the games ended with a winner or loser.

The first criterion can be evaluated simply by having Ludii try to compile any given game description, whereas the other four require playing the game. We use

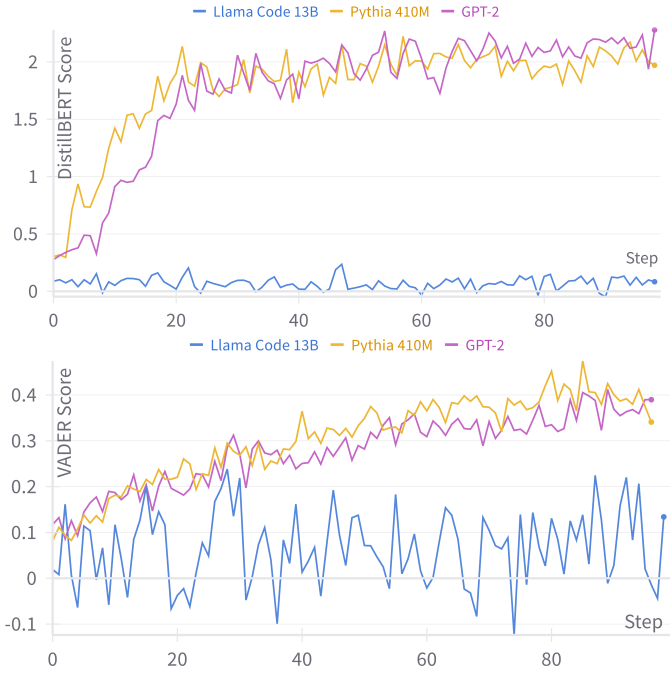

Fig. 1. Comparison between three different base models, being trained with PPO for the sentiment alignment task, using **(top)** DistillBERT as a trained reward model, or **(bottom)** VADER as programmatic reward function.

100 playthroughs (per generated game description) in which moves are selected uniformly at random to compute these criteria. Using games played between stronger agents could lead to more informative signals, but would have been prohibitive in terms of computation time. Ultimately, for any generated game description s, we use a reward of $R(s) = 0$ if s cannot be compiled (i.e., if $C(s) = 0$), $R(s) = 0.1$ if it is not playable (i.e., if $P(s) = 0$), or the geometric mean $\frac{1}{3}\left(B(s)^{\frac{1}{3}} + F(s)^{\frac{1}{3}} + D(s)^{\frac{1}{3}}\right)$ of the remaining three criteria otherwise.

5 Experiments

5.1 Sentiment Alignment Task

In this first experiment, we isolate each modification that we have introduced to TRL to ascertain their individual impacts on the performance of the system for the sentiment alignment task. In Fig. 1 (top), GPT-2's training run is consistent with Hugging Face's original results [12]. We also find that, while Pythia 410M converges as expected, LLama Code 13B fails to improve, despite the model being otherwise capable of generating sensible reviews during inference. We hypothesize that this is due to an incompatbility between (1) the version of TRL

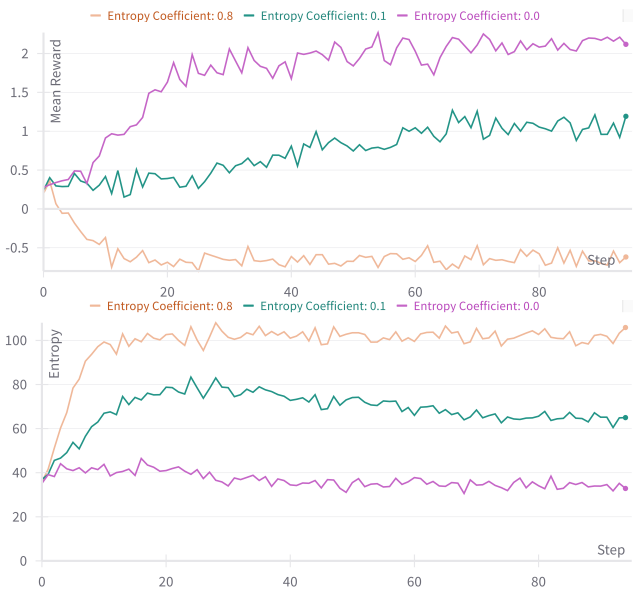

Fig. 2. Comparison between three different values for the entropy regularization coefficient β_{ENT} on the sentiment alignment task, using DistillBERT as a reward model.

we use, (2) 8-bit quantization, and (3) Llama-architecture models. GPT-2 and Pythia 410M did not use quantization.

Replacing the sentiment rewards obtained from the DistillBERT model with rewards calculated using the VADER algorithm, we find that the training runs in Fig. 1 (bottom) are consistent with those using DistillBERT, with GPT-2 and Pythia 410M steadily improving while LLama Code 13B shows no significant gains. We do, however, note an increased variance in rewards obtained during LLama Code 13B's training run with VADER rewards.

Fig. 2 shows that raising the entropy regularization coefficient β_{ENT} produces policies with higher entropy levels in their distributions over actions, as intended. However, in terms of rewards, this appears to lead to weaker policies. In contrast, when we raise the coefficient for our novel *batch*-entropy regularization variant (Fig. 3), we can produce policies with higher levels of batch-entropy (note that these numbers are not directly comparable to regular entropy number), with no substantial detriment to rewards that the models converge to. We cannot rule out that similar results might be possible with the standard entropy regularization, but this would require at least a more thorough hyperparameter sweep.

Removing the KL divergence penalty (see Fig. 4) improved both the convergence rate as well as the final performance. However, we cannot rule out the possibility that allowing the model to drift further from the pretrained model may have decreased the overall natural language quality of the outputs (e.g., in terms of style or grammar) whilst improving in terms of positive sentiment.

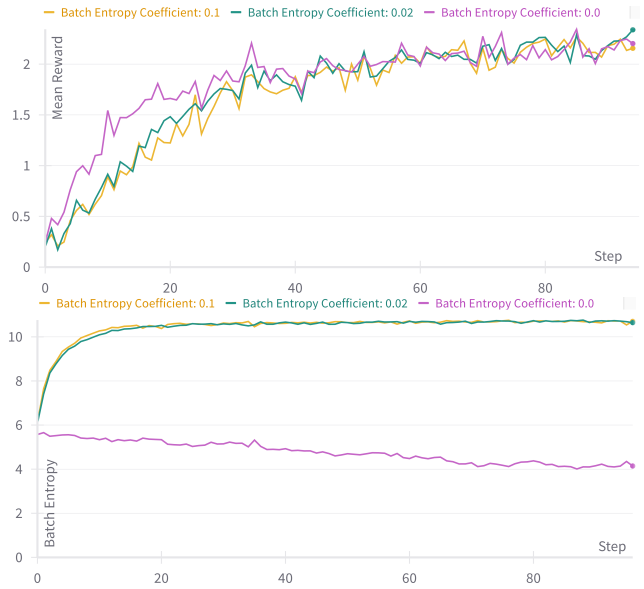

Fig. 3. Comparison between training with PPO using three different values for β_{BENT} on the sentiment alignment task, using DistillBERT as a reward model.

5.2 Arithmetic Task

In this experiment, we train a GPT-2-based model from scratch, tailored to handle arithmetic expressions. The model is configured with a context size of 64 tokens and utilizes a new word-piece tokenizer. The tokenizer's vocabulary consists of integers from 0 to 45, and the symbols '+' and '='.

Fig. 5 illustrates training under a conventional masked language modeling (MLM) objective. While the validation loss appears to converge, suggesting learning under the MLM objective, the reward deteriorates over time. The model learns to replicate approximately the correct structure, but fails to understand the mathematical semantics of the task.

As pre-training was largely ineffective, we start PPO training for the arithmetic task with an untrained model and no KL divergence penalty. Fig. 6 shows training using PPO to be more effective. The model quickly learns to output valid expressions and makes increasingly educated guesses toward the fully simplified expression, though it does not converge to a perfect solution. Figs. 7 and 8 show that without entropy or batch-entropy regularization, the entropy rapidly collapses, and the model converges on a naive policy which generates 23 regardless of the prompt it is given. This is a notable local optimum: it is the (rounded) mean of the population of problems we can generate in this task, as $\mathbb{E}\left[X1 + X_2 + X_3 + X_4 + X_5\right] = 22.5$ when $X_i \sim \{0, 1, \ldots, 9\}$. With $\beta_{\text{ENT}} = 0.3$ or $\beta_{\text{BENT}} = 0.3$, the entropy collapse can be delayed and the model's performance can exceed that of the naive policy. However, increasing β_{ENT} also appears to destabilize training.

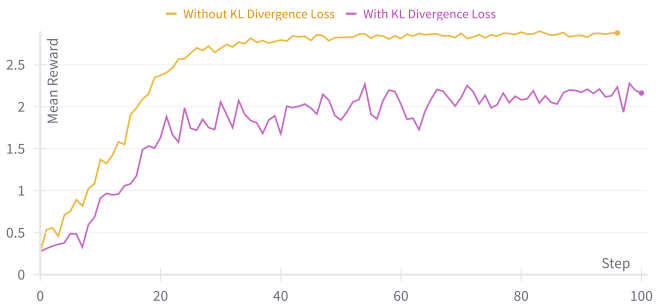

Fig. 4. Comparison between training using PPO with and without the KL penalty on the sentiment alignment task, using DistillBERT as a reward model.

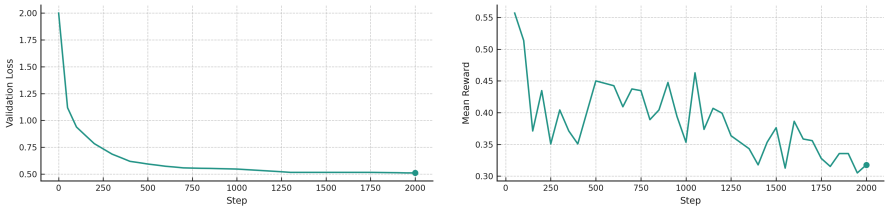

Fig. 5. Validation Loss (left) and Mean Reward (right) during supervised MLM training on the arithmetic task.

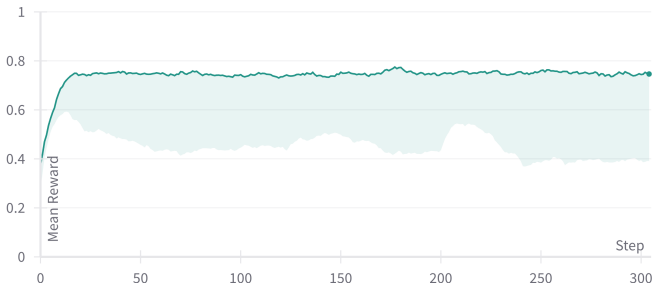

Fig. 6. Maximum and (smoothed) minimum rewards across six PPO training runs on the arithmetic task without entropy or batch-entropy regularization.

5.3 Ludii Game Synthesis Task

The grammar of Ludii's DSL is complex enough that an untrained policy will face a flat reward landscape. This sets it apart from the arithmetic task, where it was feasible to start PPO training from an untrained model. In this task it is instead critical to first pre-train a minimally proficient model using a supervised MLM objective. We define a GPT-2 model with a custom tokenizer made up of all possible ludemes and primitives in the Ludii DSL. The GPT-2 variant was trained to convergence on the Ludii fill-in-the-middle dataset. However, despite efforts to simplify the tasks' representation using string masking (masking arbi-

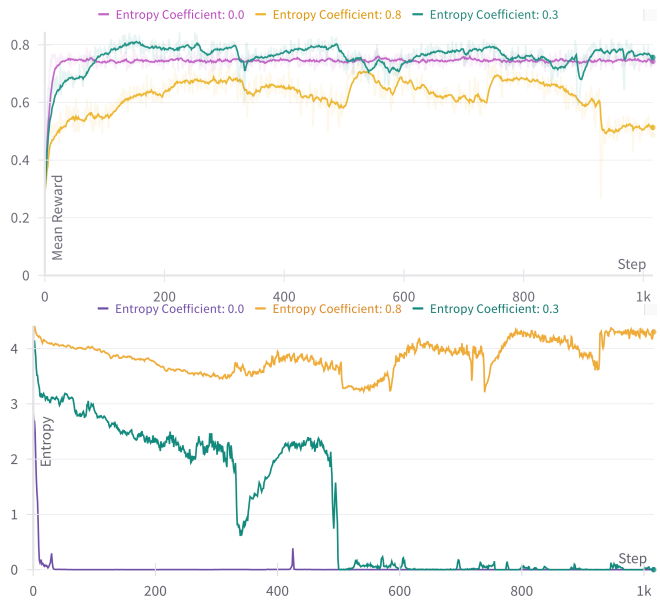

Fig. 7. Comparison between training with PPO using three different values for β_{ENT} on the arithmetic task. Rewards were smoothed to improve readability.

trary strings such as names of games and pieces) and a custom tokenizer for the Ludii DSL, the model consistently failed to obtain a non-zero reward.

Fine-tuning Pythia 410M was more effective. Training this model to convergence on the training split of the Ludii dataset led to a mean reward above 0.9 out of 1 for both the training and validation splits. This is largely possible because the fill-in-the-middle dataset overrepresents smaller parentheticals. Filtering the dataset to only games where at least 20% of the game description has been masked, the model's validation reward averages around 0.3, offering ample space for improvement with reward-based training. While the Llama Code 13B model also achieved comparable pre-training performance, we were forced to exclude it from further reward-based training since Fig. 1 suggests that Llama Code 13B is incompatible with the version of TRL that we used.

None of the 11 PPO training runs that we conducted were able to improve the policy on the Ludii game synthesis task (see Fig. 9). Runs with larger entropy and batch-entropy regularization coefficients also appear to diverge more quickly. We tried reintroducing the KL divergence penalty, but noted no difference in behavior. The fact that neither form of entropy regularization improved the performance of PPO on this task suggests that the observed instability may not be (solely) attributed to a lack of exploration. One possibility is that this task is too complex and requires a substantially larger model than Pythia 410M. It is also possible that PPO, or reinforcement learning more generally, may not be sufficiently stable for LLM training tasks that go beyond RLHF-style align-

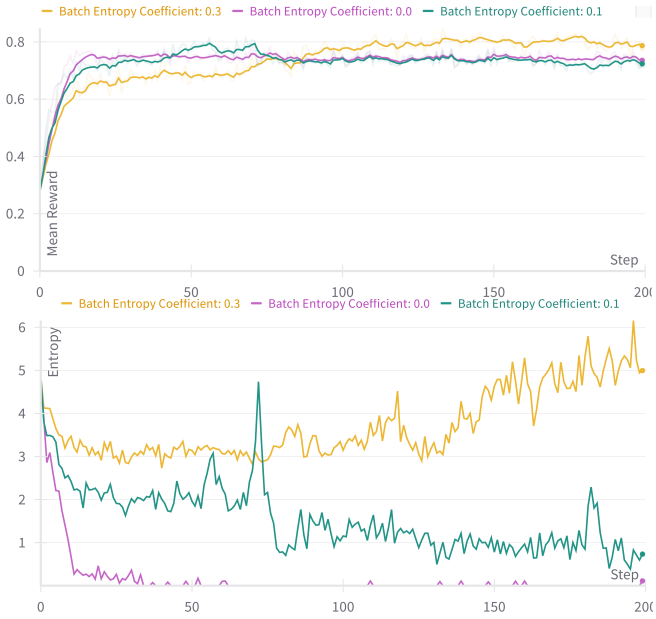

Fig. 8. Comparison between training with PPO using three different values for β_{BENT} on the arithmetic task. Rewards were smoothed to improve readability.

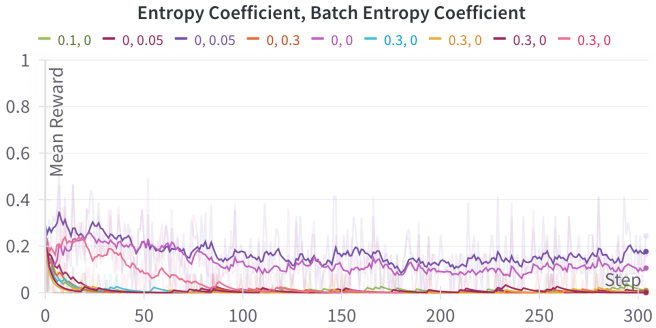

Fig. 9. Comparison between training with PPO using 11 different values for the β_{ENT} and β_{BENT} coefficients on the Ludii game synthesis task. Runs are labeled as $\beta_{\text{ENT}}, \beta_{\text{BENT}}$. Rewards were smoothed to improve readability.

ment, where minor parameter adjustments are made to encourage or discourage previously acquired capabilities, possibly due to a loss of plasticity. [19,11].

6 Related Work

Prior work on using direct RL (as opposed to RL with trained reward models) to fine-tune LLMs for formal language tasks tends to focus on settings for which

ample training data is available, and pre-trained models are already highly capable, such as commonly used programming languages [15,29]. Outside of pure RL methods, researchers have also looked towards novel inference strategies to address similar shortcomings to those considered in this paper. Examples include iterative procedures for program synthesis that feed results or error reports from unit tests back into an LLM via additional prompts [17], adding lookahead search on top of RL to improve the math abilities of LLMs [32], and combining an LLM with evolutionary search for Ludii-based game synthesis [31].

7 Conclusions & Future Work

This paper explores the feasibility of using direct Reinforcement Learning (RL) with programmed reward functions (as opposed to the more common trained reward models used in Reinforcement Learning from Human Feedback) to fine-tune LLMs for formal-language tasks which the model was not exposed to during pre-training.

Our first experiments replicate prior work on sentiment analysis [12] and validate the correctness of our TRL-based implementation [33]. We then designed an arithmetic task that could not be effectively learned through supervised learning alone. RL-based training proved to be more effective; however, without entropy regularization, the model consistently converged to a naive local optimum. Both classical entropy regularization and our novel form of batch-entropy regularization improved upon this local optimum. While, theoretical reasoning and our empirical results suggest that batch-entropy regularization provides greater stability, a comprehensive hyperparameter sweep would be needed to confirm this observation. Our final experiments found that reward-based training of GPT-2 and Pythia 410M for the complex task of generating board games in Ludii's game description language was unstable.

Our initial observation that PPO is effective at model alignment, such as encouraging a pre-trained model to write exclusively positive reviews, is consistent with the literature. However, we found that this performance does not generalize to unseen tasks, like learning to design board games, or even simply summing numbers. Since PPO is a state-of-the-art RL training algorithm, our findings highlight the need for fundamental improvements in RL training algorithms for reward-based training of LLMs. Potential avenues worth exploring include simpler methods like RLOO [2] and incorporating better domain-specific inductive biases, such as task-specific positional encodings. For complex tasks, such as game synthesis, it is also plausible that substantially larger models or more computationally expensive search algorithms [32,31] are required. Nevertheless, exploring the limits of improving RL training before resorting to such resource-intensive methods remains a compelling area of investigation.

Acknowledgments. We thank Aki Härmä for feedback on an early draft of this work.

Disclosure of Interests. The authors have no competing interests to declare that are relevant to the content of this article.

References

1. Achiam, J.: Spinning Up in Deep Reinforcement Learning (2018), https://spinningup.openai.com/en/latest/algorithms/vpg.html
2. Ahmadian, A., Cremer, C., Gallé, M., Fadee, M., Kreutzer, J., Pietquin, O., Üstün, A., Hooker, S.: Back to basics: Revisiting REINFORCE-style optimization for learning from human feedback in LLMs. In: Ku, L.W., Martins, A., Srikumar, V. (eds.) Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers). pp. 12248–12267. Association for Computational Linguistics (2024)
3. Ahmed, Z., Roux, N.L., Norouzi, M., Schuurmans, D.: Understanding the impact of entropy on policy optimization. In: Proceedings of the 36th International Conference on Machine Learning. PMLR, vol. 97, pp. 151–160 (2019)
4. Austin, J., Odena, A., Nye, M., Bosma, M., Michalewski, H., Dohan, D., Jiang, E., Cai, C., Terry, M., Le, Q., Sutton, C.: Program synthesis with large language models (2021), https://arxiv.org/abs/2108.07732
5. Bellman, R.: Dynamic Programming. Dover Publications (1957)
6. Bird, S., Klein, E., Loper, E.: Natural Language Processing with Python: Analyzing Text with the Natural Language Toolkit. O'Reilly, Beijing (2009). https://doi.org/http://my.safaribooksonline.com/9780596516499, http://www.nltk.org/book
7. Browne, C., Soemers, D.J.N.J., Piette, É., Stephenson, M., Crist, W.: Ludii language reference. ludii.games/downloads/LudiiLanguageReference.pdf (2020)
8. Browne, C.B.: Automatic Generation and Evaluation of Recombination Games. Phd thesis, Faculty of Information Technology, Queensland University of Technology, Queensland, Australia (2009)
9. Chen, M., Tworek, J., Jun, H., Yuan, Q., de Oliveira Pinto, H.P., Kaplan, J., Edwards, H., Burda, Y., Joseph, N., Brockman, G., Ray, A., Puri, R., Krueger, G., Petrov, M., Khlaaf, H., Sastry, G., Mishkin, P., Chan, B., Gray, S., Ryder, N., Pavlov, M., Power, A., Kaiser, L., Bavarian, M., Winter, C., Tillet, P., Such, F.P., Cummings, D., Plappert, M., Chantzis, F., Barnes, E., Herbert-Voss, A., Guss, W.H., Nichol, A., Paino, A., Tezak, N., Tang, J., Babuschkin, I., Balaji, S., Jain, S., Saunders, W., Hesse, C., Carr, A.N., Leike, J., Achiam, J., Misra, V., Morikawa, E., Radford, A., Knight, M., Brundage, M., Murati, M., Mayer, K., Welinder, P., McGrew, B., Amodei, D., McCandlish, S., Sutskever, I., Zaremba, W.: Evaluating large language models trained on code (2021)
10. Dereventsov, A., Starnes, A., Webster, C.: Examining policy entropy of reinforcement learning agents for personalization tasks. https://arxiv.org/abs/2211.11869 (2024)
11. Dohare, S., Hernandez-Garcia, J.F., Lan, Q., Rahman, P., Mahmood, A.R., Sutton, R.S.: Loss of plasticity in deep continual learning. Nature **632**, 768–774 (2024)
12. Face, H.: Tune gpt2 to generate positive reviews (May 2022), https://huggingface.co/docs/trl/v0.1.1/en/sentiment_tuning
13. Hutto, C.J., Hutto, C.J., Éric Gilbert, Gilbert, E.: Vader: A parsimonious rule-based model for sentiment analysis of social media text. International Conference on Web and Social Media (2014). https://doi.org/10.1609/icwsm.v8i1.14550
14. Kullback, S., Leibler, R.A.: On information and sufficiency. The Annals of Mathematical Statistics **22**(1), 79–86 (1951). https://doi.org/10.1214/aoms/1177729694, https://projecteuclid.org/euclid.aoms/1177729694
15. Le, H., Wang, Y., Gotmare, A.D., et al.: Coderl: Mastering code generation through pretrained models and deep reinforcement learning. Advances in Neural Information Processing Systems **35**, 21314–21328 (2022)

16. Li, R., Allal, L.B., Zi, Y., Muennighoff, N., Kocetkov, D., Mou, C., Marone, M., Akiki, C., Li, J., Chim, J., Liu, Q., Zheltonozhskii, E., Zhuo, T.Y., Wang, T., Dehaene, O., Davaadorj, M., Lamy-Poirier, J., Monteiro, J., Shliazhko, O., Gontier, N., Meade, N., Zebaze, A., Yee, M.H., Umapathi, L.K., Zhu, J., Lipkin, B., Oblokulov, M., Wang, Z., Murthy, R., Stillerman, J., Patel, S.S., Abulkhanov, D., Zocca, M., Dey, M., Zhang, Z., Fahmy, N., Bhattacharyya, U., Yu, W., Singh, S., Luccioni, S., Villegas, P., Kunakov, M., Zhdanov, F., Romero, M., Lee, T., Timor, N., Ding, J., Schlesinger, C., Schoelkopf, H., Ebert, J., Dao, T., Mishra, M., Gu, A., Robinson, J., Anderson, C.J., Dolan-Gavitt, B., Contractor, D., Reddy, S., Fried, D., Bahdanau, D., Jernite, Y., Ferrandis, C.M., Hughes, S., Wolf, T., Guha, A., von Werra, L., de Vries, H.: Starcoder: may the source be with you! (2023)
17. Liventsev, V., Grishina, A., Härmä, A., Moonen, L.: Fully autonomous programming with large language models. In: Proceedings of the Genetic and Evolutionary Computation Conference. p. 1146–1155. GECCO '23, Association for Computing Machinery, New York, NY, USA (2023). https://doi.org/10.1145/3583131.3590481, https://doi.org/10.1145/3583131.3590481
18. Luo, Z., Xu, C., Zhao, P., Sun, Q., Geng, X., Hu, W., Tao, C., Ma, J., Lin, Q., Jiang, D.: Wizardcoder: Empowering code large language models with evol-instruct (2023)
19. Lyle, C., Zheng, Z., Nikishin, E., Pires, B.A., Pascanu, R., Dabney, W.: Understanding plasticity in neural networks. In: Krause, A., Brunskill, E., Cho, K., Engelhardt, B., Sabato, S., Scarlett, J. (eds.) Proceedings of the 40th International Conference on Machine Learning. Proceedings of Machine Learning Research, vol. 202, pp. 23190–23211. PMLR (2023)
20. Maas, A.L., Daly, R.E., Pham, P.T., Huang, D., Ng, A.Y., Potts, C.: Learning word vectors for sentiment analysis. In: Proceedings of the 49th Annual Meeting of the Association for Computational Linguistics: Human Language Technologies. pp. 142–150. Association for Computational Linguistics, Portland, Oregon, USA (June 2011), http://www.aclweb.org/anthology/P11-1015
21. Meta: Introducing meta llama 3: The most capable openly available llm to date (Apr 2024), https://ai.meta.com/blog/meta-llama-3/
22. OpenAI: Aligning language models to follow instructions. https://openai.com/research/instruction-following (2021)
23. OpenAI: Chatgpt: Optimizing language models for dialogue. https://openai.com/research/chatgpt (2022)
24. Piette, É., Soemers, D.J.N.J., Stephenson, M., Sironi, C.F., Winands, M.H.M., Browne, C.: Ludii – the ludemic general game system. In: Giacomo, G.D., Catala, A., Dilkina, B., Milano, M., Barro, S., Bugarín, A., Lang, J. (eds.) Proceedings of the 24th European Conference on Artificial Intelligence (ECAI 2020). Frontiers in Artificial Intelligence and Applications, vol. 325, pp. 411–418. IOS Press (2020)
25. Radford, A., Wu, J., Child, R., Luan, D., Amodei, D., Sutskever, I.: Language models are unsupervised multitask learners (2019)
26. Rozière, B., Gehring, J., Gloeckle, F., Sootla, S., Gat, I., Tan, X.E., Adi, Y., Liu, J., Sauvestre, R., Remez, T., Rapin, J., Kozhevnikov, A., Evtimov, I., Bitton, J., Bhatt, M., Ferrer, C.C., Grattafiori, A., Xiong, W., Défossez, A., Copet, J., Azhar, F., Touvron, H., Martin, L., Usunier, N., Scialom, T., Synnaeve, G.: Code llama: Open foundation models for code (2024)
27. Sanh, V., Debut, L., Chaumond, J., Wolf, T.: Distilbert, a distilled version of bert: smaller, faster, cheaper and lighter (2020)
28. Schulman, J., Wolski, F., Dhariwal, P., Radford, A., Klimov, O.: Proximal policy optimization algorithms (2017)

29. Shojaee, P., Jain, A., Tipirneni, S., et al.: Execution-based code generation using deep reinforcement learning. arXiv preprint arXiv:2301.13816 (2023)
30. Sutton, R.S., Barto, A.G.: Reinforcement Learning: An Introduction. MIT Press, Cambridge, MA, 2 edn. (2018)
31. Todd, G., Padula, A., Stephenson, M., Piette, É., Soemers, D.J.N.J., Togelius, J.: GAVEL: Generating games via evolution and language models. In: Advances in Neural Information Processing Systems 37 (NeurIPS 2024) (2024), accepted
32. Uesato, J., Kushman, N., Kumar, R., et al.: Solving math word problems with process-and outcome-based feedback. arXiv preprint arXiv:2211.14275 (2022)
33. von Werra, L., Belkada, Y., Tunstall, L., Beeching, E., Thrush, T., Lambert, N., Huang, S.: Trl: Transformer reinforcement learning. https://github.com/huggingface/trl (2020)
34. Ziegler, D.M., Stiennon, N., Wu, J., Brown, T.B., Radford, A., Amodei, D., Christiano, P., Irving, G.: Fine-tuning language models from human preferences (2020)