NLP Project

Matthew LeBar

August 21, 2025

Project Description

I had originally planned to do a project based on theory of computing research on Transformers; while that topic was interesting, I ultimately decided I wanted something a little bit more practical, especially as I'm starting the job search imminently. I've developed some interest in RLVR (Reinforcement Learning with Verified Rewards), especially as I've been learning about using type theory to verify mathematical proofs or computer programs. RLVR is like RLHF, but rather than needing to train a reward model, rewards are verified with a deterministic function. I have some experience with RL from a couple classes, and since we covered most of RLHF, I figured it wouldn't be too hard to put together an RLVR project. Of course, I needed a dataset, and I pretty quickly found anopen-source RLVR dataset for math problems from AllenAI. In fact, this dataset was used for the first paper to develop RLVR (also from AllenAI). [3] I first tried to use PPO training, since that's what they used in the paper. However, after several hours of formatting data, getting the right value/reward model setup, etc. (made more difficult by some out of date Hugging Face documentation), it became clear that Hugging Face's PPOTrainer class was built to accept only neural network reward models, so I decided to switch to GRPO training, a common method for RLVR training. GRPO training came later than PPO training, and was developed specifically for mathematical reasoning.[5] GRPO was used in DeepSeek-r1, probably the most significant model trained with RLVR to date.[2] Hugging Face's GRPO Trainer was much easier to work with, and I ended up getting the code for the reward model largely from the Hugging Face documentation. The GRPO Trainer automatically took care of a reference model for KL-divergence (though I dropped it for my second training run, since this is not typically needed for GRPO training - potentially a mistake) as well as setting up tokenizing the input automatically, all of which made my life much easier.

I tried two training runs, neither of which (I thought) went well. On both, I ran on top of Allen AI's Supervised Fine-Tuned Llama 3.1 to match the paper as well as possible. I used several memory optimizations so that I could run the training on a RunPod GPU, most significantly QLoRA and LoRA (which as I understand are standard methods for reducing memory utilitization during fine-tuning).

The first run took about 31 hours to complete 1 epoch, at which point it had reward 0 and gibberish outputs (like "To ImGuiModel"Details.startDateDual Elementary Rule558rolled.onUsageIDORIA bytesFuse swappingSpe mtGenBean renegotusterity...."), which was interesting because about 2/3 through the epoch it had been producing answers that were sensible, except simply containing the wrong answer (like someone who generally understands the problem but makes some errors). I realized that some of the settings I'd set up to save memory (specifically, the number of generations in a group - see GRPO section for clarity on what this means) meant that GRPO wasn't really working. I figured this was the cause of the issue, so I started another run, along with some modifications that sped things up QUITE a bit at the cost of some memory (most sigificantly, changing from QLoRA to LoRA). The second run I let go just over 1 epoch, as I believed the training was not working. This took about 18 hours, so it was much faster. It was bouncing between a low reward rate and 0 reward without any consistent improvement. At step 750 (the first epoch), the responses were coherent but incorrect (sometimes including correct answers, but with other calculations in the box other than the answer itself), while by step 1000 they were incoherent in the style of the first run again. I don't understand how this happens. (I noticed I had some slight issues with truncation of inputs and with my maximum output length cutting off some productive chains-of-reasoning, but a) that is a tradeoff with memory utilization and b) while it would certainly impair model performance, it is hard to see how this could generate the total incoherence). The low rewards themselves are not necessarily a problem, as these were training rewards and with GRPO, eval rewards is what really matters (since the training generations are from the reference policy - again, see GRPO section for details). I did not appreciate the significance of the reference model in GRPO until after ending both training runs. Upon closer inspection, the second (but not the first!) training run was actually improving eval rewards, meaning the training was working. Knowing this now, I plan on re running the second run setup, with more frequency evaluation. Because of how I had the eval frequency set up, I don't know the model was doing at step 1000, so I'm unsure if the incoherence/gibberish is really an issue.
In the rest of this report, I explain PPO, GRPO, and the reward model I used, and share some thoughts on moving the project forward. These explanations are meant to cover the main methods I used in this project that I haven't learned before. I've also included some tables of the rewards and time for each of the runs from WandB, and also attached in my upload a Jupyter Notebook to show my code (though I ran a .py script version of the same code for my training). It's fairly simple since I ended up just using the Hugging Face GRPO Trainer.

Policy Gradient Methods

I assume the level of knowledge I had before doing this project, specifically basic RL terminology and definitions and KL-Divergence. In Policy Gradient Methods, the parameters for a policy function π are updated via gradient ascent.

Thus, if we have an action a, a state s, a set of parameters θ and represent the action, state, and parameters at time t with A_t and S_t respectively, we have [6]

$$\pi(a|s,\theta) = P[A_t = a|S_t = s, \theta_t = \theta]$$

This need not use a value function, though it can, in which case we have what is called an actor-critic model. We will see PPO is an actor-critic model, and GRPO is not (which saves quite a bit of memory, as the value model is large). In either case, the parameters are updated with gradient ascent (to maximize performance rather than to minimize loss):

$$\theta_{t+1} = \theta_t + \alpha\widehat{\nabla J(\theta_t)}$$

with $\widehat{\nabla J(\theta_t)}$ an estimate of the gradient of the performance of the policy with parameters θ_t.[6] The key to a gradient ascent method will be how that estimator works estimated. The most basic approach is to use $\hat{\mathbb{E}}[\nabla_\theta \log \pi_\theta(a|s)\hat{A}]$ where $\hat{A}$ is the advantage of action a at state s, i.e. $Q(s,a) - v(a)$ under policy π_θ (calculating this value function is where the value/"critic" model comes into play). Intuitively this is how much the log probability of an action in a state has changed, multiplied by how much better the policy is when it always takes that action at that state. The expectation is with respect to state action pairs, and is estimated empirically. This requires that several trajectories with the policy be sampled before updating the parameters, or the sample will not be large enough for a good estimate. In Transformer Reinforcement Learning (like I am working with in this project), each trajectory is just a prompt-completion pair, with the prompt the state and the completion the action. [4] The critic model is trained, often against TD estimates of the value.

PPO

PPO (Proximal Policy Optimization) algorithms modify the above basic approach to ensure the gradient ascent does not take too large of steps. They come in two varieties: PPO-Penalty, penalizing KL-Divergence against a reference model, and PPO-Clip, which clips the maximum amount the parameters can change by. [1] Formally, let $r(\theta,a) = \frac{\pi_\theta(a|s)}{\pi_{\theta_{\text{ref}}}(a|s)}$ (intuitively a measure of how much more likely action a is) and let

$$\text{clip}(z,x,y) = \begin{cases} x & \text{if } z < x \\ z & \text{if } x < z < y \\ y & \text{if } y < z \end{cases}$$

Then, with PPO-Clip, J (not ∇J) is estimated with

$$\hat{\mathbb{E}}[\min(r(\theta,a)\hat{A}, \text{clip}(r(\theta,a), 1-\epsilon, 1+\epsilon)\hat{A})]$$

where ϵ is a hyperparameter. The clipping ensures that the changes are never too large, while the min ensures that the more pessimistic expectation is used.

[4] PPO-Penalty (shockingly) uses a penalty KL-divergence term, yielding for as an estimate for J:

$$\hat{\mathbb{E}}\left[r(\theta, a)\hat{A} - \beta \cdot \mathrm{KL}[\pi_{\theta_{\mathrm{ref}}}(\cdot|s), \pi_\theta(\cdot|s)]\right]$$

where β is a hyperparameter. PPO-Clip and PPO-Penalty can be combined so we have

$$\hat{\mathbb{E}}\left[\min(r(\theta, a)\hat{A}, \mathrm{clip}(r(\theta, a), 1-\epsilon, 1+\epsilon)\hat{A}) - \beta \cdot \mathrm{KL}[\pi_{\theta_{\mathrm{ref}}}(\cdot|s), \pi_\theta(\cdot|s)]\right]$$

This is what was used in the Allen AI paper.[3]

GRPO

GRPO (Group Relative Policy Optimization) forgoes the need for a value model by computing advantage a different way. For each prompt, GRPO will generate a number of possible responses (sampled from the reference policy) and compute the advantage of these responses against each other. Thus, if G is the size of the group of possible responses and a_i response i, GRPO yields an estimate for J of [2]:

$$\frac{1}{G}\sum_{i=1}^{G}\hat{\mathbb{E}}\left[\min(r(\theta, a_i)\hat{A}_i, \mathrm{clip}(r(\theta, a_i), 1-\epsilon, 1+\epsilon)\hat{A}_i) - \beta \cdot \mathrm{KL}[\pi_{\theta_{\mathrm{ref}}}(\cdot|s), \pi_\theta(\cdot|s)]\right]$$

(Of course we could shove the averaging over the group inside the expectation, but I thought this way of putting it would be clearer). The simplest definition for $\hat{A}_i$ (and the one most relevant to my setup - the alternative involves multiple reasoning steps), given that the reward for completion i is r_i and $\boldsymbol{r}$ the random variable representing the distribution of rewards within the group, is $\hat{A}_i = \frac{r_i - \mu(\boldsymbol{r})}{\sigma(\boldsymbol{r})}$, just the reward of an individual completion normalized against its group.

Reward Model

The reward model I used quite simply yields reward of 1 if the answer is exactly correct, and 0 otherwise. The training examples include all answers in "
boxed", training the model to provide answers in this format. This is following Allen AI's approach again; perhaps it would be better to follow the DeepSeek approach more fully since they used GRPO. They also include a format reward to encourage the model to put <think> </think> tags around its thought process, which I definitely think could be helpful. [2]

Moving Forward

This section originally contained a wide variety of hypotheses for why my model was performing so poorly, since I was looking at training rewards without appreciating that they are from the reference model. Still, the gibberish is odd, but the evals (as you'll see in the results section) are appreciably improving, so I think I need to just let it run for a while with more frequent eval results and see how it looks. There are definitely still some warts with this version but I'd like to see if the basic version of it works. Without the project deadline hopefully I can use cheaper/slower alternative to the RunPod GPU I was using (suggestions welcome!).

Results

Had to put these at the end due to the difficulty of LaTeX image insertion. Note "likely-sun-12" (the orange) was the second training run and "peach-blaze-11" (the green) the first training run. I wish I had run the second training run up until the next eval stage; that would let me know if the gibberish is really a problem, as up until step 750 the eval rewards are clearly and significantly increasing. Re-running with much more frequent evaluation I think is my very next move.

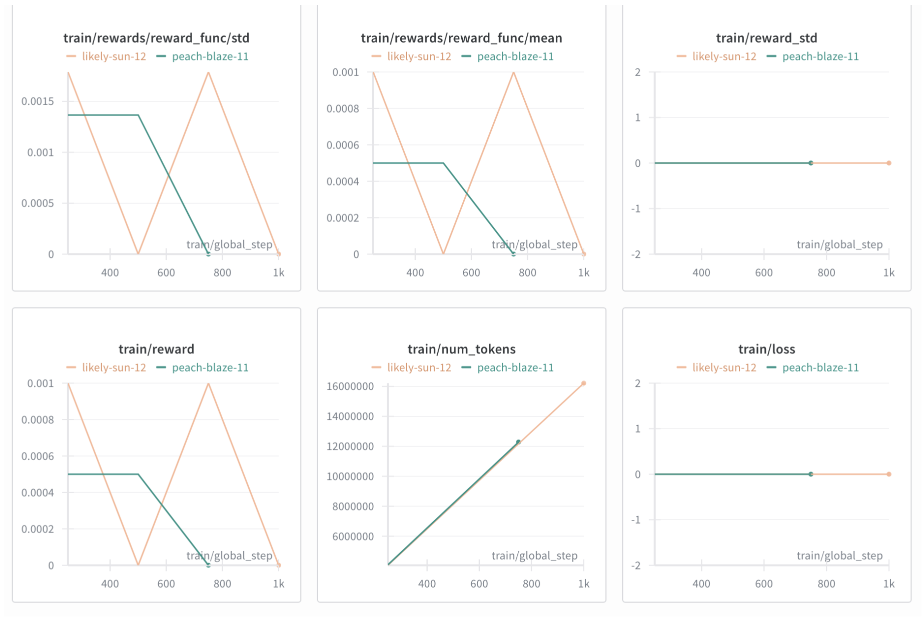

Figure 1: Training rewards

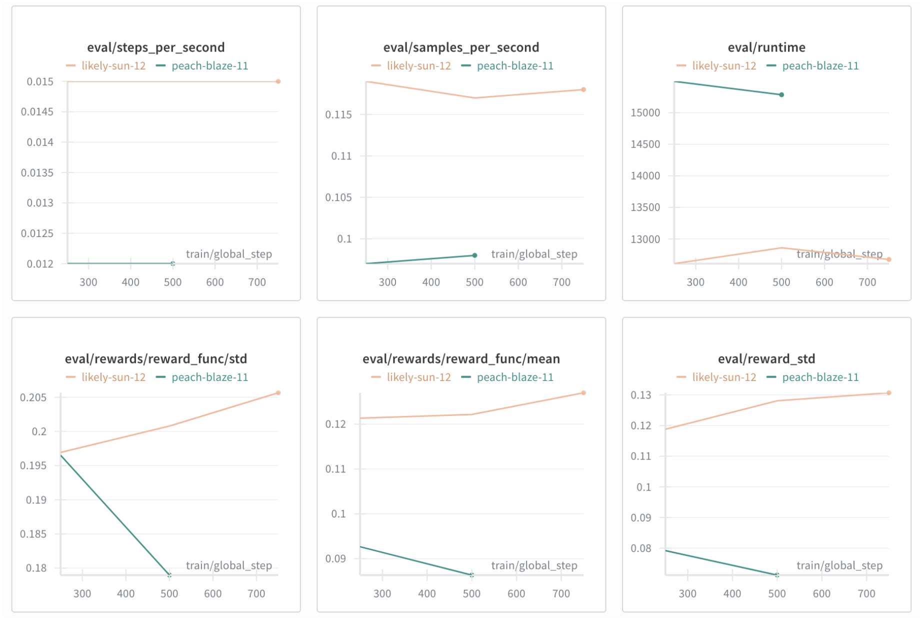

Figure 2: Eval rewards

References

[1] Joshua Achiam. Spinning Up in Deep Reinforcement Learning. 2018.

[2] DeepSeek-AI, Daya Guo, Dejian Yang, Haowei Zhang, Junxiao Song, Ruoyu Zhang, Runxin Xu, Qihao Zhu, Shirong Ma, Peiyi Wang, Xiao Bi, Xiaokang Zhang, Xingkai Yu, Yu Wu, Z. F. Wu, Zhibin Gou, Zhihong Shao, Zhuoshu Li, Ziyi Gao, Aixin Liu, Bing Xue, Bingxuan Wang, Bochao Wu, Bei Feng, Chengda Lu, Chenggang Zhao, Chengqi Deng, Chenyu Zhang, Chong Ruan, Damai Dai, Deli Chen, Dongjie Ji, Erhang Li, Fangyun Lin, Fucong Dai, Fuli Luo, Guangbo Hao, Guanting Chen, Guowei Li, H. Zhang, Han Bao, Hanwei Xu, Haocheng Wang, Honghui Ding, Huajian Xin, Huazuo Gao, Hui Qu, Hui Li, Jianzhong Guo, Jiashi Li, Jiawei Wang, Jingchang Chen, Jingyang Yuan, Junjie Qiu, Junlong Li, J. L. Cai, Jiaqi Ni, Jian Liang, Jin Chen, Kai Dong, Kai Hu, Kaige Gao, Kang Guan, Kexin Huang, Kuai Yu, Lean Wang, Lecong Zhang, Liang Zhao, Litong Wang, Liyue Zhang, Lei Xu, Leyi Xia, Mingchuan Zhang, Minghua Zhang, Minghui Tang, Meng Li, Miaojun Wang, Mingming Li, Ning Tian, Panpan Huang, Peng Zhang, Qiancheng Wang, Qinyu Chen, Qiushi Du, Ruiqi Ge, Ruisong Zhang, Ruizhe Pan, Runji Wang, R. J. Chen, R. L. Jin, Ruyi Chen, Shanghao Lu, Shangyan Zhou, Shanhuang Chen, Shengfeng Ye, Shiyu Wang, Shuiping Yu, Shunfeng Zhou, Shuting Pan, S. S. Li, Shuang Zhou, Shaoqing Wu, Shengfeng Ye, Tao Yun, Tian Pei, Tianyu Sun, T. Wang, Wangding Zeng, Wanjia Zhao, Wen Liu, Wenfeng Liang, Wenjun Gao, Wenqin Yu, Wentao Zhang, W. L. Xiao, Wei An, Xiaodong Liu, Xiaohan Wang, Xiaokang Chen, Xiaotao Nie, Xin Cheng, Xin Liu, Xin Xie, Xingchao Liu, Xinyu Yang, Xinyuan Li, Xuecheng Su, Xuheng Lin, X. Q. Li, Xiangyue Jin, Xiaojin Shen, Xiaosha Chen, Xiaowen Sun, Xiaoxiang Wang, Xinnan Song, Xinyi Zhou, Xianzu Wang, Xinxia Shan, Y. K. Li, Y. Q. Wang, Y. X. Wei, Yang Zhang, Yanhong Xu, Yao Li, Yao Zhao, Yaofeng Sun, Yaohui Wang, Yi Yu, Yichao Zhang, Yifan Shi, Yiliang Xiong, Ying He, Yishi Piao, Yisong Wang, Yixuan Tan, Yiyang Ma, Yiyuan Liu, Yongqiang Guo, Yuan Ou, Yuduan Wang, Yue Gong, Yuheng Zou, Yujia He, Yunfan Xiong, Yuxiang Luo, Yuxiang You, Yuxuan Liu, Yuyang Zhou, Y. X. Zhu, Yanhong Xu, Yanping Huang, Yaohui Li, Yi Zheng, Yuchen Zhu, Yunxian Ma, Ying Tang, Yukun Zha, Yuting Yan, Z. Z. Ren, Zehui Ren, Zhangli Sha, Zhe Fu, Zhean Xu, Zhenda Xie, Zhengyan Zhang, Zhewen Hao, Zhicheng Ma, Zhigang Yan, Zhiyu Wu, Zihui Gu, Zijia Zhu, Zijun Liu, Zilin Li, Ziwei Xie, Ziyang Song, Zizheng Pan, Zhen Huang, Zhipeng Xu, Zhongyu Zhang, and Zhen Zhang. Deepseek-r1: Incentivizing reasoning capability in llms via reinforcement learning, 2025.

[3] Nathan Lambert, Jacob Morrison, Valentina Pyatkin, Shengyi Huang, Hamish Ivison, Faeze Brahman, Lester James V. Miranda, Alisa Liu, Nouha Dziri, Shane Lyu, Yuling Gu, Saumya Malik, Victoria Graf, Jena D. Hwang, Jiangjiang Yang, Ronan Le Bras, Oyvind Tafjord, Chris Wilhelm, Luca Soldaini, Noah A. Smith, Yizhong Wang, Pradeep Dasigi, and Hannaneh

Hajishirzi. Tulu 3: Pushing frontiers in open language model post-training, 2025.

[4] John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, and Oleg Klimov. Proximal policy optimization algorithms, 2017.

[5] Zhihong Shao, Peiyi Wang, Qihao Zhu, Runxin Xu, Junxiao Song, Xiao Bi, Haowei Zhang, Mingchuan Zhang, Y. K. Li, Y. Wu, and Daya Guo. Deepseekmath: Pushing the limits of mathematical reasoning in open language models, 2024.

[6] Richard S. Sutton and Andrew G. Barto. *Reinforcement Learning: An Introduction*. The MIT Press, second edition, 2018.