Final Report

Mu Junrong

Abstract

Large Language Models (LLMs) have demonstrated impressive capabilities in natural language understanding and generation. However, improving their performance (i.e., correctness and conciseness) on complex reasoning tasks remains challenging. Reinforcement Learning (RL) offers a promising approach to fine-tune LLMs by optimising specific reward functions aligned with desired outputs. This project wraps up the knowledge, experiments and results obtained in the project. This report summarises the underlying methods, experiments, and insights, contributing to the understanding of reward shaping for LLMs fine-tuning via RL.

1 Introduction to the Project

1.1 Project overview

The main goal of this project is to investigate how Reinforcement Learning (RL) can be used to post-train large language models (LLMs) to improve performance on structured reasoning tasks. Instead of relying solely on traditional supervised fine-tuning, we explore the application of reinforcement learning to directly optimise for desired output behaviours. Specifically, the project aims to shape LLM outputs to be:

- Correct: producing the mathematically accurate final answer
- Well-formatted – adhering to a clear and structured step-by-step reasoning process
- Concise – delivering the correct solution in as few words or steps as possible, without unnecessary verbosity

2 Reinforcement Learning (RL)

Reinforcement Learning (RL) is a framework in machine learning where an agent interacts with an environment to learn a policy that maximises cumulative reward over time (Sutton & Barto (2018)). The agent makes decisions (actions) based on observations (states) and receives feedback (rewards) that guide its learning. The Reinforcement Learning (RL) problems are typically modelled as Markov Decision Process (MDP) (Puterman (2014)), which can be found in Appendix A

2.1 Objective function of Reinforcement Learning (RL) and Policy Gradient

The objective function (which is the mathematical definition of the goal of the optimisation problem) can be defined as

$$J(\theta) = \int p_\theta(\tau) R(\tau)\, d\tau$$

By differentiating the objective function $J(\theta)$:

$$\nabla_\theta J(\theta) = \mathbb{E}_{\tau \sim \pi_\theta} \left[\nabla_\theta \log p_\theta(\tau) R(\tau) \right]$$

The full derivation of $\nabla_\theta J(\theta)$ can be found in Appendix B.

2.2 Algorithms of Reinforcement Learning (RL)

Several algorithms of Reinforcement Learning are designed, including REINFORCE (Williams (1992)), Proximal Policy Optimisation (PPO) (Schulman et al. (2017)), Generalised Reward Policy Optimisation (GRPO) (Yuan et al. (2024)) and Generalised Structured Policy Optimisation (GSPO) (Liu et al. (2024)).

Table 1: Comparison of Policy Gradient Methods

Algorithm	Description
REINFORCE	REINFORCE is a basic algorithm for training models with various reward functions. It is a Monte Carlo policy gradient method that uses full episodic rollouts to estimate the gradient: $$\nabla_\theta J(\theta) = \mathbb{E}\left[\sum_t \nabla_\theta \log \pi_\theta(a_t|s_t) \cdot R\right]$$ To reduce variance, a **baseline** b is often subtracted from the reward: $$\nabla_\theta J(\theta) = \mathbb{E}\left[\sum_t \nabla_\theta \log \pi_\theta(a_t|s_t) \cdot (R - b)\right]$$
PPO	Proximal Policy Optimisation (PPO) is a more stable and sample-efficient policy gradient method. It updates the policy using a clipped surrogate objective: $$L^{PPO}(\theta) = \mathbb{E}_t\left[\min\left(r_t(\theta)\hat{A}_t, \text{clip}(r_t(\theta), 1-\epsilon, 1+\epsilon)\hat{A}_t\right)\right]$$ where $r_t(\theta) = \frac{\pi_\theta(a_t|s_t)}{\pi_{\theta_{old}}(a_t|s_t)}$ is the probability ratio and $\hat{A}_t$ is the estimated advantage. PPO prevents large, unstable updates by restricting the change in policy per step, making it particularly effective for training large models like LLMs.
GRPO	Generalised Reward Policy Optimisation (GRPO) is a variant of policy gradient that incorporates a flexible reward transformation and KL regularisation: $$\pi^* = \arg\max_\pi \mathbb{E}_\pi[r(x)] - \beta \cdot \text{KL}(\pi \| \pi_{\text{ref}})$$ This balances task-specific optimisation with fidelity to the original LLM behaviour, reducing the risk of catastrophic forgetting.
GSPO	Generalised Structured Policy Optimisation (GSPO) incorporates structure-aware rewards at the token/segment level. It assigns rewards based on intermediate structures (e.g., reasoning steps) and propagates them backwards during learning.

3 Large Language Models (LLMs)

Large Language Models (LLMs) are deep neural networks trained to generate and understand natural language. Built primarily on the **transformer architecture**, LLMs have achieved remarkable performance across a wide range of tasks, including question answering, summarization, translation, and mathematical reasoning.

At their core, LLMs are autoregressive models that learn the probability distribution over sequences of tokens:

$$P(x_1, x_2, \ldots, x_T) = \prod_{t=1}^{T} P(x_t|x_{<t})$$

where x_t is the token generated at time step t, and $x_{<t}$ are the preceding tokens. This formulation allows LLMs to generate coherent, fluent text token-by-token, conditioned on the input context.

3.1 Transformer architecture

The transformer architecture can be found in Appendix D. The key components in the transformer architecture include (Vaswani et al. (2017)):

- Positional Encoding: adds information about token positions since transformers lack inherent sequence order.
- Multi-head Self-Attention: allows the model to attend to different parts of the input simultaneously, capturing syntactic and semantic relationships. Details of the self-attention can be found at Appendix C
- Add and norm layer: helps with stability, training speed, and gradient flow by applying Residual connection (Add) and Layer normalisation (Norm).
- Feedforward Layers: applied after attention, enabling nonlinear transformation of token representations. The final output of each iteration is given by

$$Output = LayerNorm(x + FFN(x))$$

where

$$\text{FFN}(x) = W_2 \cdot \text{ReLU}(W_1 \cdot x + b_1) + b_2$$

3.2 Prompt engineering

Prompt engineering refers to the art and science of crafting inputs (prompts) to guide large language models (LLMs) toward generating desired outputs. Since most LLMs, especially in a post-training or inference setting, operate in a zero-shot or few-shot paradigm, the prompt often determines how effectively the model performs a given task (Liu et al. (2023)). The structure, phrasing, and context within a prompt can significantly influence the quality, accuracy, and coherence of the model's response. There are few types of prompts, including:

- Zero-shot prompting: asking the model to perform a task with no examples. (e.g., "Translate this sentence to French: I love programming.")
- Few-shot prompting (Radford et al. (2019)): providing a few labelled examples before the actual task.
- Chain-of-thought prompting: encouraging the model to reason step-by-step.
- Instruction prompting: providing explicit instructions about the task. (e.g., Summarise the following paragraph in two sentences.)

4 Experiments and results

Three experiments were done, including

- Implementing and comparing the policy gradient variants on the CartPole-v0 environment
- Evaluate Reinforcement Learning post-training (i.e., GRPO) on LLMs to improve its performance and correctness
- Design the reward function of RL the LLM to produce concise outputs without sacrificing accuracy

The experiments have the setup as following:

Table 2: Experiment Setup for Reinforcement Learning and LLM Post-Training

Reinforcement Learning	Details
Reinforcement Learning	• Environment: CartPole (physics-based control problem) • Hardware: Single NVIDIA 8GB GPU
LLMs Post-Training with RL	• Model used: Qwen2.5-0.5B • Hardware: Single NVIDIA 8GB GPU • Temperature: 0.5

4.1 Experiment on Reinforcement Learning (RL)

Based on the experiment results which can found in Appendix E, the use of normalised advantages and reward-to-go significantly improves training performance, with the performance ranking as follows: `normalised advantages combined with reward-to-go > normalised advantages alone > reward-to-go alone > neither technique applied`

Furthermore, the experiments demonstrate that increasing the data batch size leads to more stable training outcomes, as reflected in the reduced performance fluctuations observed in larger batch settings.

4.2 Experiment on Large Language Models (LLMS) training with Reinforcement Learning (RL)

For the Countdown task, quantitatively speaking, during training, there was an increase in the average reward from 0.01 at first to 0.063 and the reward curve obtained demonstrates that the model gradually learned to generate more accurate answers as the policy was optimised using GRPO.

Qualitatively speaking, the model generates non-sensical and incorrect outputs that do not follow the template at first, with many foreign keys. After 35 iterations, the output is correct, although the phrasing can be insignificant and misleading (i.e., which is the cause of the very low reward obtained).

For the GSM8K dataset (i.e., school-level mathematical problems) (Cobbe et al. (2021)), the reward increased steadily from an initial 0.01 to approximately 0.054, indicating that the model was gradually learning to produce more correct answers GRPO. This shows that the reinforcement learning derived from string matching the generated answer with the ground truth is effective, despite being sparse and binary.

4.3 Experiment on conciseness improvement

Continuing GRPO-based fine-tuning of Qwen2.5-0.5B on GSM8K, a new reward function is explored to ensure correctness and proper format, as well as to improve the conciseness of answers.

4.3.1 Experiment details

The new reward function is defined as a weighted combination of three components:

$$R = R_c + R_f - \lambda \cdot L$$

where:

- $R_c = \begin{cases} 1 & \text{if answer matches gold standard} \\ 0 & \text{otherwise} \end{cases}$ is the correctness reward
- $R_f = \begin{cases} 1 & \text{if format is “The answer is \#.”} \\ 0 & \text{otherwise} \end{cases}$ is the format reward

- $L = \frac{n}{n_{\max}}$ is the length penalty (normalized token count)
- $\lambda \in \mathbb{R}^+$ is the length penalty weight (typically $\lambda = 0.3$)

This reward structure prioritizes correctness (R_c) over formatting (R_f), and penalizes verbose responses through the term $-\lambda L$ to ensure the conciseness of the answer.

4.3.2 Experiment results

With the new reward function, the length of the output reduces while remaining correct. For example, for the prompt

```
Natalia sold clips to 48 of her friends in April, and then she sold half as many
   clips in May. How many clips did Natalia sell altogether in April and May?
```

the original output is

```
Natalia sold 48/2 = 24 clips in May. Natalia sold 48+24 = 72 clips altogether in
                    April and May. Thus, the answer is 72.
```

which has 23 words. With the new reward function, the new output is

```
    48/2 = 24, 48+24 = 72. In total, Natalia sold 72 clips in April and May
```

which has 15 words and is still correct.

5 Discussion and future works

Despite these successes, there are several limitations and challenges. In the classical RL experiments, computational constraints limited the exploration of more complex environments. For the LLM-related experiments, despite the relatively small model sizes and limited training iterations meant that improvements, there are still improvements shown in the performance, showing the possibility of Small Language Models (SLMs). Additionally, reward function design remains a delicate and labour-intensive process—poorly specified rewards can result in undesired optimisation paths.

In the future, RL on LLMs with smaller size or even Small Language Models (SLMs) can be explored, and alternative RL frameworks can be explored.

- Scaling RL to Small Language Models (SLMs): There is growing interest in Small Language Models (SLMs) that are lightweight yet competitive. Combining efficient RL algorithms with parameter-efficient fine-tuning techniques (e.g., LoRA, QLoRA) could enable high-quality alignment on edge devices.
- Exploring Alternative RL Paradigms: Beyond conventional policy gradient methods, frameworks such as the Free-Energy Principle or Active Inference offer biologically inspired alternatives that could improve sample efficiency and robustness. Integrating such methods with LLM post-training may yield novel forms of alignment less dependent on large-scale datasets.
- Automated Reward Function Design: Leveraging meta-learning or evolutionary search to automatically generate and refine reward functions could reduce the trial-and-error nature of current RLHF workflows, improving efficiency and reliability.

Overall, these experiments reaffirmed RL's role as a powerful and general-purpose optimization framework. From improving policy stability in classical environments to fine-tuning the stylistic behavior of language models, RL methods consistently demonstrated their adaptability. As hardware efficiency and algorithmic sophistication advance, the integration of RL into both large- and small-scale AI systems will likely accelerate, enabling more controllable, interpretable, and human-aligned models.

A Markov Decision Process (MDP)

The Markov Decision Process (MDP) is a mathematical model of the decision-making process, extensively used in the field of Reinforcement Learning (RL). It formally describes a fully observable environment where an agent performs actions, and outcomes are partly random. In other words, a Markov Decision Process (MDP) is a Markov Reward Process (MRP) with decisions.

A Markov Decision Process (MDP) is a Markov Reward Process (MRP) with the agent making decisions. It is under an environment where all states are Markov (i.e. every state has the Markov property: the state in which the future is independent of the past, given the present).

MDP is a tuple $\langle S, A, P, R, \gamma \rangle$, where

- S is a finite set of states. In the context of LLM post-training, the "state" can be interpreted as the current context or prompt, the "action" is the next token generated, and the "reward" is based on the quality of the full generated output (e.g., correctness, formatting, conciseness).
- A is a finite set of actions
- P is a state transition probability matrix, which is defined as $P_{ss'} = \mathbb{P}[S_{t+1} = s' \mid S_t = s]$
- R is a reward function with the reward R_s at state s being $R_s = \mathbb{E}[R_{t+1} \mid S_t = s]$
- $\gamma \in [0, 1]$ is a discount factor

B Policy Gradient Derivations

The main objective of Reinforcement Learning (RL) is to find an optimal policy π that maximizes the reward (i.e. maximizes the state-value function and action-value function):

$$\pi^* = \arg\max_{\pi} \mathbb{E}_{\pi} \left[\sum_{t=0}^{\infty} \gamma^t R_{t+1} \right]$$

Let $\pi_\theta(a|s)$ be a stochastic policy parameterized by θ (where θ can be parameters of the distribution), and let τ represent a trajectory (i.e., the sequence of states, actions, rewards):

$$\tau = (s_0, a_0, r_1, s_1, a_1, r_2, \ldots)$$

The objective function (which is the mathematical definition of the goal of the optimization problem) can be defined as

$$\begin{aligned} J(\theta) &= \mathbb{E}_{\tau \sim \pi_\theta} \left[\sum_{t=0}^{T} \gamma^t r_t \right] \\ &= \int p_\theta(\tau) R(\tau) \, d\tau \end{aligned}$$

By differentiating the objective function $J(\theta)$:

$$\begin{aligned} \nabla_\theta J(\theta) &= \nabla_\theta \int p_\theta(\tau) R(\tau) \, d\tau \\ &= \int \nabla_\theta p_\theta(\tau) R(\tau) \, d\tau \\ &= \int p_\theta(\tau) \nabla_\theta \log p_\theta(\tau) R(\tau) \, d\tau \\ &= \mathbb{E}_{\tau \sim \pi_\theta} \left[\nabla_\theta \log p_\theta(\tau) R(\tau) \right] \end{aligned}$$

C Multi-head Self-Attention

The transformer architecture, introduced by Vaswani et al. (2017), forms the backbone of modern LLMs. It replaces recurrence with self-attention, allowing models to process sequences in parallel and learn long-range dependencies efficiently.

The core of the transformer architecture is the multi-head self attention mechanism, which helps a model understand how each word in a sentence relates to the others, by assigning attention scores. These scores decide how much focus each word should give to the rest. Each token is turned into three vectors, Query (Q), Key (K) and Value (V), by $Q = XW^Q, K = XW^K, V = XV^V$, where $W^Q, W^K, W^V \in \mathbb{R}^{d_{\text{model}} \times d_k}$ are trainable weight matrices.

The scaled attention score between token i and token j is the dot product of their Q and K vectors, since dot product represents the similarity between two vectors. The attention score is scaled by $\sqrt{d_k}$ to prevents large dot products (for stable gradients).

$$\text{Attention score}_{i,j} = \frac{Q_i \cdot K_j^T}{\sqrt{d_k}}$$

The attention matrix is given by:

$$\text{Attention}(Q, K, V) = \text{softmax}\left(\frac{QK^T}{\sqrt{d_k}}\right) V$$

Where QK^T computes the similarity scores between each token pair, and softmax turns scores into probabilities (attention weights). By doing so, a matrix of the same shape as the input is computed, which represents new context-aware vectors for each token.

In multi-head self-attention, multiple sets (heads) are used, each head pays attention to different aspects of the input. Each head is given by

$$\text{head}_i = \text{Attention}(XW_i^Q, XW_i^K, XW_i^V)$$

for each $i \in \{1, \ldots, h\}$

All heads are then concatenated and applied to compute the final linear transformation

$$\text{MultiHead}(Q, K, V) = \text{Concat}(\text{head}_1, \ldots, \text{head}_h)W^O$$

Where: $W^O \in \mathbb{R}^{hd_v \times d_{\text{model}}}$ is a learnable projection matrix.

D Transformer architecture

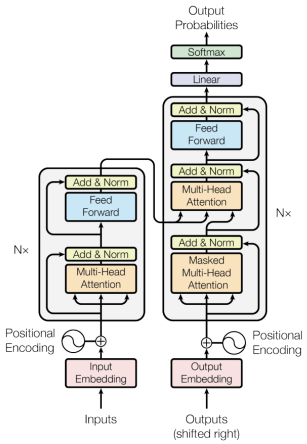

Figure 1: Transformer architecture

E RL experiment plots

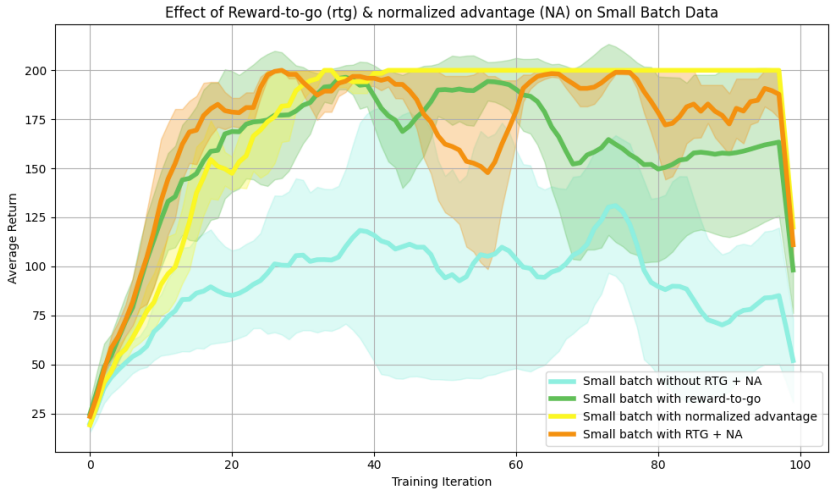

Figure 2: Comparing the performance of different setups with a small batch of data. Normalised advantages combined with reward-to-go show the best performance, and normalised advantage alone is better than reward-to-go alone.

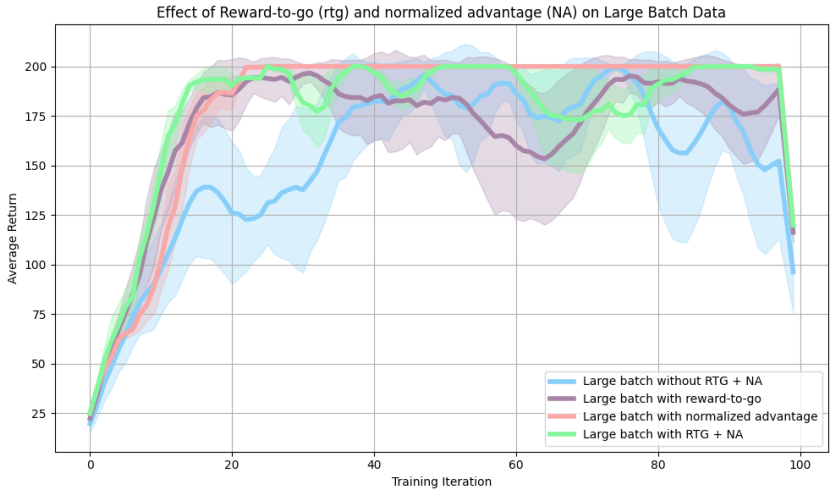

Figure 3: Comparing the performance of different setups with a large batch of data. Larger batch data shows better result than small batch.

References

Karl Cobbe, Vineet Kosaraju, Mohammad Bavarian, Mark Chen, Heewoo Jun, Lukasz Kaiser, Matthias Plappert, Jerry Tworek, Jacob Hilton, Reiichiro Nakano, et al. Training verifiers to solve math word problems. *arXiv preprint arXiv:2110.14168*, 2021.

Pengfei Liu, Weizhe Yuan, Jinlan Fu, Zhengbao Jiang, Hiroaki Hayashi, and Graham Neubig. Pre-train, prompt, and predict: A systematic survey of prompting methods in natural language processing. *ACM Computing Surveys*, 55(9):1–35, 2023. doi: 10.1145/3560815.

Tianqi Liu, Ziyu Luo, Yihao Zhang, Yuanjun Li, Zhewei Hu, Yujing Wang, Jing Chen, and Hang Zhang. Generalized structured policy optimization for language models. *arXiv preprint arXiv:2403.08235*, 2024.

Martin L. Puterman. Markov decision processes: Discrete stochastic dynamic programming. 2014.

Alec Radford, Jeffrey Wu, Rewon Child, David Luan, Dario Amodei, and Ilya Sutskever. Language models are few-shot learners. *arXiv preprint arXiv:2005.14165*, 2019.

John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, and Oleg Klimov. Proximal policy optimization algorithms. *arXiv preprint arXiv:1707.06347*, 2017.

Richard S. Sutton and Andrew G. Barto. *Reinforcement Learning: An Introduction*. MIT Press, 2018. URL http://incompleteideas.net/book/RLbook2020.pdf.

Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Lukasz Jones, Aidan N. Gomez, Łukasz Kaiser, and Illia Polosukhin. Attention is all you need. *Advances in Neural Information Processing Systems (NeurIPS)*, 2017.

Ronald J. Williams. Simple statistical gradient-following algorithms for connectionist reinforcement learning. *Machine Learning*, 8:229–256, 1992. doi: 10.1007/BF00992696.

Zheng Yuan, Honglei Xue, Jian Yuan, Yiding Wang, Yibin Guo, and Shen Gao. Generalized reward policy optimization. *arXiv preprint arXiv:2404.13228*, 2024.