Extended Abstract

Motivation Modern paradigm for using language models often assumes downloading a large pre-trained model and fine-tuning it for downstream tasks. Supervised Fine-Tuning comprises of doing additional optimisation steps on the full set of model's weights starting from the pre-trained values and using a custom dataset, more representative of the desired specialised downstream task. Additional characteristics can be induced on the model, like presence of "reasoning" steps which breakdown the problem into steps and have been shown to increase model's performance. Reinforcement Learning techniques have been shown to be particularly useful for inducing these additional behaviours and tuning model further, especially when dealing with limited, but high quality data. For these reasons the RL element of LLM development is often dubbed "the cherry on top" in the "cake analogy"Mundt et al. (2025).

Method This project focused mainly on improving LLM performance on Math Reasoning tasks. The particular task considered in this work is the numerical version of a game of Countdown. In this game a participant is required to use the set of provided input numbers and basic arithmetic operations to arrive at a provided target number subject to certain additional constraints. A huge benefit of this task is that while it's non-trivial to solve, and when modelling it as a search problem the resulting tree has a high branching factor, the solutions to it are trivially verifiable.

Implementation The starting point for this project it the pre-trained **Qwen2.5-0.5B** modelHui et al. (2024). It features around half a billion parameters (0.36B excluding the embeddings layer), 24 Transformer layers, context length of 32,768 tokens and a multilingual support. Hence it offerers a very reasonable balance between portability and expressibility and is thus a great starting point for trying to get a reasonable performance on our task.

SFT:"WarmStart"Gandhi et al. (2025): reasoning-inducing dataset for the game of Countdown contained 1,000 query and response pairs. DPO: "TinyZero"'s datasetPan et al. (2025) containing 490,000 pairs of of a single target number and 3-4 input numbers for the game of Countdown.

Results We have observed a similar pattern across both tasks. SFT brought about the bulk of the improvements aligning the model more closely with the task at hand. With Math Reasoning task we would observe it directly from generated outputs. Before SFT there were no reasoning steps or answer formatting. The outputs made some vague references to maths and calculation, but the provided answers weren't useful in any way. After the SFT step the model consistently exhibited reasoning behaviour, though the actual reasoning steps were sometimes spurious or limited. The formatting started appearing, but not consistently. The optimisation results seemed to saturate well before reaching the final number of SFT steps. The RL methods were able to bring additional, if modest gains from that point. Within Math Reasoning task this translated into more sensible reasoning steps and better use of formatting. True breakthrough came from test-time compute use of a verifier and tuing of generation parameters. This shows that combining the LLM workflow with additional tooling may often be the most desirable course of action for tasks requiring specific formatting or NP problems with efficient verifiers.

Discussion Over the course of this project with have found that a lot can be done in terms of improving model's task-specific performance with reasonable amounts of data and reasonable compute. While not a consumer hardware, the AWS G6e instance featuring a 48GB GPU (NVIDIA L40S) which has been used for running the experiments is relatively affordable.
It was quite a journey which made us get hands-on with a lot of practical software engineering problems and gain a deeper, more intimate understanding of the fine-tuning process, the algorithms involved and the importance of appropriate data processing.

Conclusion This work has shown an importance of multi-pronged approach to fine-tuning and the relative strengths and weaknesses of different families of methods. The enormous potential of test-time methods shows that a thorough analysis of target uses cases of a fine-tuned model needs to be carried out to decide if the computational budget should be invested more towards optimisation or inference.

Default Project: Improving performance of a large language model on mathematical reasoning task through various fine-tuning and test-time compute methods.

Witold Gawlikowicz
Department of Computer Science
Stanford University
witold@stanford.edu

Abstract

This project focused on improving LLM performance on Math Reasoning tasks, specifically the numerical version of Countdown game where participants use provided input numbers and basic arithmetic operations to arrive at a target number. A nice characteristic of this problem is that it's trivially verifiable, in fact we use the same verifier in many parts of the project. This stylised problem brings useful insights into the usefulness of using LLMs combined with efficient verifiers on LP family of problems. We have also considered the Instruction Following task, although less emphasis was placed on it. We have developed a common computational framework for both tasks with supervised fine-tuning as the first step, followed by Reinforcement-Learning based methods as an additional step. For Math Reasoning we have used REINFORCE leave-one-out, an online algorithm using our reward function to construct advantage estimates of reduced variance. For Instruction following task we have used an offline Direct Policy Optimisation method and a corresponding preference data set. We concluded with exploration of test-time method which brought about enormous increase in the overall model performance on the Math Reasoning task.

1 Introduction

Modern paradigm for using language models often assumes downloading a large pre-trained large language model (LLM) and fine-tuning it for downstream tasks. Various techniques are employed to achieve this goal. Supervised Fine-Tuning (SFT) comprises of doing additional optimisation steps on model's weights starting from the pre-trained values and using a custom dataset, more representative of the specialised downstream task considered. With appropriately chosen dataset additional characteristics like "reasoning steps" (whereby a model breaks down the problem into smaller parts and solves them sequentially) can be induced into the pre-trained model.
SFT can either be done as full fine-tuning where all of model's weights get optimised or using various "Parameter Efficient Fine-Tuning" (PEFT) methods which typically focus on modifying the attention or feed-forward layers of a transformer-based in some way which uses significantly less parameters than the total number of parameters in the base pre-trained model. Both of these families of methods can lead to much better task-specific performance in many cases. However, since they all focus on minimising the next-token prediction objective they have their limitations, especially as far as more complex tasks are concerned.
For that reason they are often augment with Reinforcement Learning-based methods which allow more precise injection of user preferences into the optimisation objective. Lastly, aside from training,

significant performance gains can often also be realised at inference time by altering the default generation mechanism of the model. This project explores all of the aforementioned techniques with the aim of getting the best possible performance on the downstream tasks considered and understanding the contribution of these various elements.

2 Related Work

The Math Reasoning task considered in this work will be the numerical version of a game of Countdown[1]. A huge benefit of this task is that while it's non-trivial to solve, and when modelling it as a search problem the resulting tree has a high branching factor, the solutions to it are trivially verifiable. Since this is basically the definition of NP problems Wigderson (2019), this stylised problem can bring useful insights into the usefulness of using LLMs combined with efficient verifiers on this family of problems. Gandhi et al. (2024) have studied the performance of an LLM on this task. The authors have constructed a synthetic dataset where they generated a random set of input numbers and targets and then used a few different simple heuristics to generate solutions. They have used a domain specific language to represent the process of solving the problem. Importantly, heuristics were deliberately chosen to be simple and not all of them yielded a correct answer (accuracy of around 57%). They have also used another dataset comprised of optimal solutions without any additional reasoning and backtracking steps present it the first dataset. Interestingly, the model achieved a higher performance when trained with the former dataset (accuracy of 51.27% vs 25.73%). They have then applied Reinforcement Learning (RL) techniques to further increase the performance of the model by 6%. It would have been interesting if the authors have varied the accuracy of the set of heuristic solutions used as input data and presented how it impacted the accuracy of the trained model. The strategy used by the authors is an example of inducing Chain-of-Thought reasoning capabilities to a LLM. This technique encourages the model to output a reasoning process whilst generating an answer and has been shown to improve performance on variety of tasks including Math Reasoning. It's an example of a wider set of techniques called "test time inference" where additional computational resources are used by the model at inference time in order to generate a higher quality answer. It is these methods that we would like to explore in the extension part of the project. RL is a key ingredient in successfully implementing many of these techniques as recently exemplified by the immensely popular DeepSeek model DeepSeek-AI et al. (2025).

Snell et al. (2024) study optimal allocation of test-time compute depending on the type and difficulty of a task. They have estimated that an optimal strategy brings about a 2-4 times improvement in the efficiency of test time compute. This highlights that the often neglected generation parameters should be treated on par with other hyperparameters when trying to improve model's overall performance on a task. They have also studied the trade-off between using a larger model as is versus a smaller model with effectively implemented test time inference and found that the later can still outperform a vanilla model that is up to 14 times larger. While deciding optimal test-compute strategy may not be applicable to our task at hand (the task is fairly uniform, and modifying strategy based on input and target numbers may have very limited impact on performance not justifying the additional complexity) these findings solidified our view that it's a worthy area of research as it can be used by relatively small entities with more limited budget to still achieve state of the art results on certain tasks. The authors divide test time computation approaches into two categories: input level interventions which modify the distribution of LLMs outputs and output level techniques which use learned reward models (called verifiers) to perform post-hoc output modifications.

Zhang et al. (2025) study verifiers in more detail. They contrast their typical use case of assigning a score to a candidate solution and choosing the final solution based on that with verifiers trained on next-token prediction which integrate better with the underlying architecture of LLMs and established instruction fine-tuning techniques. They also propose a variant particularly well-suited to models using Chain-of-Thought technique. Wang et al. (2023) show that extending CoT technique with self-consistency can further improve performance. Their approach uses a simple assumption that for tasks with well defined solutions all the correct reasoning paths should converge on the same answer.

[1] In this game a participant is required to use the set of provided input numbers and basic arithmetic operations to arrive at a provided target number subject to certain additional constraints.

3 Method

3.1 Tasks

In this project we consider two downstream tasks: Mathematical Reasoning and Instruction Following. The inputs to the **Math Reasoning** are an array of 3 or 4 input integers as well as a target integer. The task is to construct and equation using the basic arithmetic operations and using each input integer exactly once so that the result equals the target number. The focus is on evaluating and improving model's ability to perform multi-step mathematical reasoning using natural language. Additionally, the task is to put the resulting equation between the <answer> and </answer> HTML-style tags. This presents additional challenge for the model, however it also improves it's usefulness and potential ease of integration with downstream systems.
Instruction Following task is to generate coherent, relevant, and helpful responses to the presented prompts spanning variety of tasks including, but not limited to: translation, summarization, guidance, knowledge checks and information retrieval.
We consider each of the tasks separately, hence the models customized for each of the tasks are never used with the other task. While pre-trained models try to strike a balance between a variety of tasks and maintain a good overall performance, our approach here is to use the LLM as a tool to solve a problem at hand as well as possible with no regard to how the model generalises to other tasks or even the overall language modelling abilities.

3.2 Metrics

Math Reasoning: the model gets a score of 0 for a generated answer if it doesn't contain these tags irrespective of presence or correctness of the equation. If the equation presented between the tags correctly evaluates to the target number and uses all of the input numbers exactly once then a score of 1 is awarded for the entire generation (including the formatting score). Since formatting part is an enabler for potentially receiving the higher score for correctness, we track the fraction of malformatted answers across our validation data set.

Instruction Following: we use an external model to assess model's responses and compare them to baselines. Baseline for SFT is the pre-trained model, baseline for DPO is the result of SFT fine tuning. We then construct prompt-by-prompt win indicator function (1 if the answer of the current model gets a higher score than that of a reference model and 0 otherwise) and average them across the validation set to get the win-rate.

We consider both tasks separately, while they both start from the common base model the models used diverge from that point onwards. There's a separate SFT step for each of the tasks. Next, the result of that step gets fed into the RL step which is RLOO for Math Reasoning and DPO for Instruction Following. For Math Reasoning we also take the best model from RL step and try to increase its performance further using test-time compute methods.

3.3 Supervised Fine-Tuning (SFT)

Like most modern workflows for customizing a general pre-trained LLM to a specific task at hand, we start with SFT. For both tasks we consider the full fine-tuning as well as Low-Rank Adaption (LoRA) methods. We use the same number of optimiser steps for each of these methods and track the loss function as well as validation metrics appropriate for each of the tasks across these steps. The loss function is given by:

$$\mathcal{L}_{\text{SFT}}(\pi_\theta) := \max_\theta \mathbb{E}_{x,y\in D} \sum_{t=1}^{|y|} \log \pi_\theta(y_t|x, y_{<t}), \tag{1}$$

where θ represents the model parameters being optimised, D is the dataset of prompt-response pairs (x, y), $|y|$ is the length of the response sequence, y_t is the token at position t in the sequence, $y_{<t}$ denotes all the tokens before it, and $\pi_\theta(y_t|x, y_{<t})$ is the probability assigned by the model to token y_t conditional on the prompt tokens x and preceding response tokens $y_{<t}$.

3.4 REINFORCE Leave-One-Out (RLOO)

Ahmadian et al. (2024) looked into Proximal Policy Optimisation (PPO) algorithm applied to LLM fine-tuning. The authors argued that the algorithm is actually not that well suited to practical fine-tuning applications, where the base model is already fairly well-aligned with the task. They demonstrated that per token reward structure is sparse and often spurious as it's really the end of sequence token that brings about most rewards. Therefore they argue that a reward for entire generation is a better approach in many applications. The authors have also shown that PPO is heavily sensitive to hyperparameter choice making practical use more challenging. They have combined those observations into a variant of popular REINFORCE Williams (1992) algorithm from Reinforcement Learning with a leave one-out approach to advantage estimation which mitigates the variance of the estimate. The resulting loss function is given below:

$$\mathcal{L}_{\text{RLOO}}(\pi_\theta; \pi_{\text{ref}}) := \frac{1}{k} \sum_{i=1}^{k} \left[R(y_{(i)}, x) - \frac{1}{k-1} \sum_{j \neq i} R(y_{(j)}, x) - \alpha \log \frac{\pi_\theta(y_{(i)}|x)}{\pi_{\text{ref}}(y_{(i)}|x)} \right] \nabla \log \pi_\theta(y_{(i)}|x) \tag{2}$$

where x is the input prompt, k is the number of samples generated per x, $y_{(i)}$ is the i-th response sampled from the target model, $R(y, x)$ is the reward function, $\pi_\theta(\cdot|x)$ is the probability distribution of the target model parametrised by θ, $\pi_{\text{ref}}(\cdot|x)$ is the probability distribution of the reference model, α is the weight controlling the strength of the Kullback–Leibler divergence penalty between the current model and the reference model aimed at preventing the target model from drifting too far away from the reference model to mitigate possibility of catastrophic forgetting, and ∇ is the gradient operator.
It is an online RF algorithm (it requires generations from the model being optimised). In many applications $R(y, x)$ needs to be learned, however in our case we can just use the Countdown reward function used for evaluation. We skip the prompts for which all generations result in equal rewards from the loss function as that brings no additional information. We considered adding a length penalty to the reward which penalises divergence from mean response observed in the "WarmStart" set to enrich the reward signal, however since this approach has not improved the performance of the algorithms we skip its details for the sake of brevity.

3.5 Direct Policy Optimisation (DPO)

DPO is an RL-based algorithm which relies on the target model and the reference model, as well of pair of preferred (y_w) and rejected (y_l) responses per prompt x. These responses are pre-generated and can come from any model or other source and ofen involve human or model-based labelling. Hence it's considered an offline algorithm. The loss function is given below:

$$\mathcal{L}_{\text{DPO}}(\pi_\theta; \pi_{\text{ref}}) := -\mathbb{E}_{(x, y_w, y_l) \sim \mathcal{D}} \left[\log \sigma \left(\beta \log \frac{\pi_\theta(y_w|x)}{\pi_{\text{ref}}(y_w|x)} - \beta \log \frac{\pi_\theta(y_l|x)}{\pi_{\text{ref}}(y_l|x)} \right) \right], \tag{3}$$

where $\pi_\theta(\cdot|x)$ is the probability distribution of the target model parametrised by θ, $\pi_{\text{ref}}(\cdot|x)$ is the probability distribution of the reference model, $\mathcal{D}$ is a dataset of preference tuples (x, y_w, y_l), σ is the logistic function, and β is the parameter controlling the level of adherence to reference model's behaviour.

3.6 Test-time compute

Finally, we turn our attention away from modifying model parameters and explore the performance improvements that can be achieved by taking closer look at the inferences process and how it can be modified. We only consider the Math Reasoning task in this part.
We being by exploring impact of various generation parameters (maximum response length, Softmax temperature, repetition penalty) We also consider the following sampling schemes when generating each next token. Standard multinomial sampling: consider all tokens proportionally to model's probability distribution. Top-K: consider only top k most probable tokens. Top-P: consider only the smallest subset of tokens whose cumulative probability if greater than or equal to p. Beam search expand n sequences in parallel, return sorted by cumulative probability (highest first).

Lastly, we choose the most promising schemes and parameters discussed above and extend the inference logic with a verifier which can score the multiple responses provided per same input prompt. We provide two thresholds: primary and secondary. First we consider a primary threshold and narrow down the results to the ones which meet or exceed it. If there's at least one match we return the top match (preserving the ordering output by the model, sorted by sum of log probabilities). If there's no match we consider the secondary threshold with a lower value and act analogously. If this also yields no results we return the first generation returned by the model (analogous to no verifier).

4 Experimental Setup

We use 1,000 optimiser steps for SFT part and 50 steps for RL part for each of the tasks. We set these limits arbitrarily to strike a balance between sensible exploration of the evolution of the loss function and reasonable runtime. We have chosen to use the these relative values as RL techniques can much more computationally expensive (especially the online ones) and usually require more carefully curated datasets, hence we believe this split to be realistic of actual tasks solved this way within the academia and the industry.
During evaluation of SFT and RL algorithms we always use greedy decoding (Softmax temperature of 0). The main reason is to reduce the variance of the scores based on the corresponding generations and hence make them more comparable across optimiser steps. A side benefit is that it makes the generation quicker and more memory-efficient. We use vLLM (a local language server) for generation during evaluation as it's more efficient. In RLOO we still use Hugging Face's models directly to generate as it simplifies the codebase and makes log-probabilities easier to calculate from logits output by the model. Also, the small batch sizes that we use likely wouldn't justify incurring the overhead for spinning up a new vLLM instance after each optimiser step. One problem that we have noticed with vLLM is that it introduces some stochasticity into the generations even when done on the same set of model weights with all random number generators seeded right before the generation. The problem is exacerbated by the use of `BFloat16` format, hence we ended up using `Float16` when quantizing models to lower precision to aid the performance. We have also implemented gradient accumulation for the same reason. To help with stability we have also implemented gradient clipping and learning rate scheduling.

4.1 Data

For Math Reasoning task we have used the following datasets:

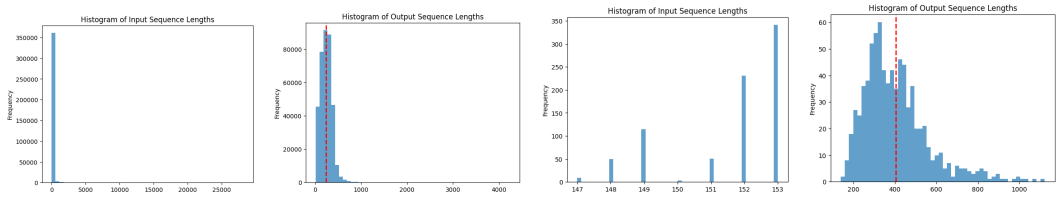
- "WarmStart"[2]: reasoning-inducing dataset for the game of Countdown.
  - Resulting splits: training - 800, development - 200, test - 200.
  - Used for: SFT (training split).
- "Countdown-Tasks-3to4"[3]
  - Resulting splits: training - 343,254, development - 98,073, test - 49,037.
  - Used for: RLOO (training split) and validation (development split subsets).

For Instruction Following task we have used the following datasets:

- "SmolTalk"[4]: high-quality chat responses from GPT-4o
  - Resulting splits: training - 368,272, development - 92,069, test - 24,229.
  - Used for: SFT (training split).
- "Ultrafeedback"[5]:instruction following preference dataset
  - Resulting splits: training - 48,908, development - 12,227, test - 2,000.
  - Used for: DPO (training split) and validation (development split subsets).

[2]Gandhi et al. (2025),`https://huggingface.co/datasets/Asap7772/cog_behav_all_strategies`
[3]Pan et al. (2025) `https://huggingface.co/datasets/Jiayi-Pan/Countdown-Tasks-3to4`
[4]Allal et al. (2025),`https://huggingface.co/datasets/HuggingFaceTB/smol-smoltalk`
[5]Dubois et al. (2024), `www.huggingface.co/datasets/HuggingFaceH4/ultrafeedback_binarized`

We have implemented pipeline for downloading datasets from HuggingFace (HF) and preparing local splits saved as a parquet files. These can then be loaded into memory and encoded in one go, or streamed from disk and encoded on the fly as batches get traversed[6]. We have put aside 20% of each training set as a development set to be used during subsequent evaluations when comparing different methods, and for hyperparameter optimisation.
Next step was to determine the optimal maximum sizes for tokenized representation of queries and responses from the training set. Our analysis (see Figure 1) revealed that for "SmolTalk" 260 tokens for query and 426 tokens for response were enough to fully cover 95% of the dataset. For "WarmStart" the corresponding figures were 153 and 730[7]. Performance test has shown that our data pipeline can handle $1e5$ examples from both training sets (encoding both query and token) in around 100s. Tests with sets used for evaluation in this part of the project - "UltrafeedBack" and "Countdown" - yielded respectively 45s and 32s when encoding query only. We use the response limits established during the analysis of training sets during evaluation with our fine-tuned models unless stated otherwise.



(a) SmolTalk Queries (b) SmolTalk Responses (c) WarmStart Queries (d) WarmStart Responses

Figure 1: Dataset tokenized context length distributions, red vertical line indicates the mean.

4.2 Models

The starting point for both tasks is it the pre-trained `Qwen2.5-0.5B` model Hui et al. (2024). Hugging Face implementation of the model is used along with model weights downloaded from the same source. We also use Hugging face to store model weights from our experiments for reproducibility and inference. The version of `Qwen2.5` model which we used features around half a billion parameters (0.36B excluding the embeddings layer), 24 Transformer layers, maximum context length of 32,768 tokens and a multilingual support. Hence it offerers a very reasonable balance between portability and expressibility and is thus a great starting point for our fine-tuning experiments. This version of the model has not been explicitly tuned for instruction following before our fine-tuning process.

Additionally, `Llama-3.1-nemotron-70b-reward` Bercovich et al. (2025) model has been used for the evaluation of the Instruction Following task performance. We use the NVIDIA-hosted version[8] to get the score for completion of a given prompt. We compare these against corresponding scores for the reference model (for SFT this is the base `Qwen2.5-0.5B` model, for DPO it's the result of the SFT step).

4.3 Hyperparameters

We use the standard implementation of `AdamW` from `PyTorch` as our optimiser. We fix `weight_decay` at 0.01 and vary learning rate depending on the use case, but use consistent setting across both tasks. For SFT we use a learning rate scheduler with 100 warm-up steps during which the learning rate is increased linearely from 0 to a target rate. Over the next 900 steps we use the `CosineAnnelingLR` scheduler from PytTorch to decrease the learning rate to tenth of the target value. The target value for full fine-tuning is 2×10^{-5}, for LoRA it's 5×10^{-5}. Both RLOO and DPO used a constant learning rate of 1×10^{-5}. SFT used a batch size of 4 with 5 accumulation steps (effective batch size of 20),

[6]The second option is mainly useful when using MPS framework for rapid development on a local machine as it uses unified memory and any memory used-up before the training loop won't be available to GPU cores during training. For GPU-enabled VMs this isn't as useful as tokenizers typically use CPU and RAM rather than GPU and VRAM anyway

[7]For both datasets we have applied the chat template. For "SmolTalk" we included system messages (15%) of the dataset and reduced each example to just one user-assistant exchange.

[8]`https://build.nvidia.com/nvidia/llama-3_1-nemotron-70b-reward/modelcard`

both RL algorithms used a batch size of 2 with 2 accumulation steps (effective batch size of 4). The dropout rate was left at the standard 0.1 from the base model.

Both LoRA configurations used α of 16 and dropout rate of 0.05. Other parameters corresponding to the two sets which yielded best scores on MultiNLI Williams et al. (2018) dataset in the original LoRA paper Hu et al. (2021). Namely, `LORA_1` uses a rank of 8 and targets W_Q, W_V in each transformer layer, while `LORA_2` uses a rank of 2 and targets W_Q, W_K, W_V, W_O in each transformer layer. Same LoRA configurations are applied to both tasks. RLOO achieved best results with $\alpha = 0$ and $k = 4$.
DPO achieved best results with $\beta = 0.01$.
Test-time compute without a verifier achieved best results with beam search with 8 beams (maximum number considered), no repetition penalty and double the response context length compared to what the model was trained with.
When extending test-time compute with verifier function we achieve best results with standard multinomial sampling (neither Top-K nor Top-P sampling) with 16 samples (highest number considered), a Softmax temperature of 1 and the double the same response length as what the model was trained on.
Please refer to the Appendix for the details of hyperparamter tuning experiments conducted. Please note that due to the long runtimes we have carried out opportunistic hyperparameter tuning where we have concentrated on a selected subset of parameters, ordered them by the expected impact and then for each of them considered a few candidates, selected the one yielding best results, fixed that parameter and moved to consider the next. Clearly, this isn't the best possible approach as relationships between these parameters are definitely non-linear, however it's the best we could achieve under the limited computational resources we were able to devote to this project.
Please refer to Section B of the Appendix for additional implementation details.

5 Results

5.1 Qualitative Analysis

Let's consider Instruction Following task first. In Figure 2 we can see initial increase in the loss function which soon changes into a mild, decreasing in a sustained way, slope (amid all the noise coming from each mini-batch step considering a different subset of data) despite having covered only a fraction of the epoch. LoRA fares marginally better here. However, when looking at DPO results, using the full set of model weights rather than the LoRA model yields much better results. This indicates that additional degrees of freedom are beneficial in modelling the relative preferences while not diverging too far from the reference model. We notice that a very mild $\beta = 0.01$ copes best. The slope of the DPO loss and resulting average win rate increase is similarly shallow and fluctuating.
Moving on to Math Reasoning in Figure 3 we notice a similarly noisy loss function, however with a much more rapidly decreasing slope in the earlier steps of the optimiser. We can observe that while both LoRA methods managed to decrease the loss objective substantially, this has not fed into any increase in the training score or drop in the number of malformatted responses. It may be that additional expressibility coming from embeddings layer or the later model layers immediately preceding the logits calculation is needed to induce the correct formatting behaviour. While the RLOO loss function is quite noisy, the algorithm did manage to noticeable improve the score on the validation subset in just 50 iterations.

5.2 Quantitative Evaluation

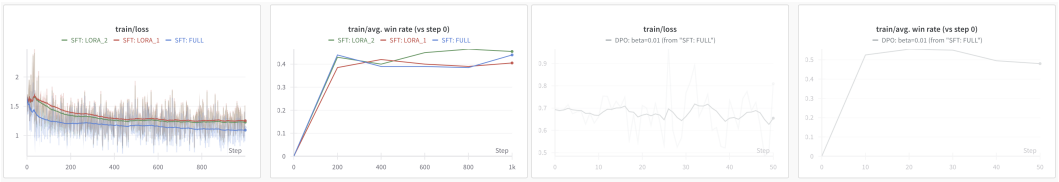
Consulting Table 1 we can see that SFT increased the average win rate quite substantially compared to the baseline model (45.5%). DPO managed to add another 7 percentage points to this, despite doing a small fraction of the iteration steps. However the training time was very similar to SFT at around 30 minutes.
Moving on to Table 2 we observe a similar pattern, however the gains from RLOO are smaller compared to DPO (just over 3 percentage), despite over 50% longer training time compared to SFT for this task.
However, the most remarkable gains in performance come from test-time compute extensions to the model which increase the average score by over 40 percentage points at a cost of just over 3 times longer inference time. Figure 4 dissects various candidate test-time compute methods in more detail. We see a large range of performances, with beam-based methods usually performing worse than

Table 1: Instruction Following

Method	Avg. win-rate[1]	Training time[2]	Inference time[3]	Final leaderboard score
Base Model	-	-	28s	-
SFT	0.4550	33m00s	28s	-
DPO	0.5250	30m32s	28s	[NOT PROCESSED IN TIME]

[1] Average score win-rates for development split of "Ultrafeedback" dataset reduced to 200 observations. Base model is treated as a baseline in both cases and win-rates are computed w.r.t. relative scores assigned by an external model the the base model and the target model.
[2] Includes 6 evaluation steps of 200 data points each.
[3] Per 200 data points, includes generation by the model only, not scoring with the external model.

standard multinomial sampling (despite much longer generation times) in the presence of a verifier. Without the verifier it was actually beam methods which performed better. The overall performance of the model with different test-time compute methods varies from just over 52% to over 90%. More details on the effect of different generation hyperparameters can be found in Section A.1.3 of the Appendix.



(a) SFT (b) DPO

Figure 2: Instruction Following: SFT and DPO loss function and avg. win rate evolution (current optimiser step vs model at step 0) for development split of "Ultrafeedback" dataset reduced to 200 observations.

(a) SFT (b) RLOO

Figure 3: Math Reasoning: SFT and RLOO loss function and average score across generations for development split of "Countdown-Tasks-3to4" dataset reduced to 200 observations.

6 Discussion

The results in this project are broadly in line with wider literature. SFT brings about steady improvements up to a certain point, RL-based methods can be used to increase the performance by additional margins from there. The beauty of these methods is that they allow easier injection of user preferences into the optimisation process. Here our evaluation metrics and preferences were not just post hoc reported numbers, but in a way also inputs into the optimisation process itself. We would've liked to carry out RL-based optimisation for both tasks with larger batch sizes and for more optimisation steps, but due to the high computational cost and memory saturation that characterises these methods we struggled to do more while maintaining a sensible computational budget.
The often neglected generation parameter hypertuning and test-time compute methods brought about an impressive performance boost.

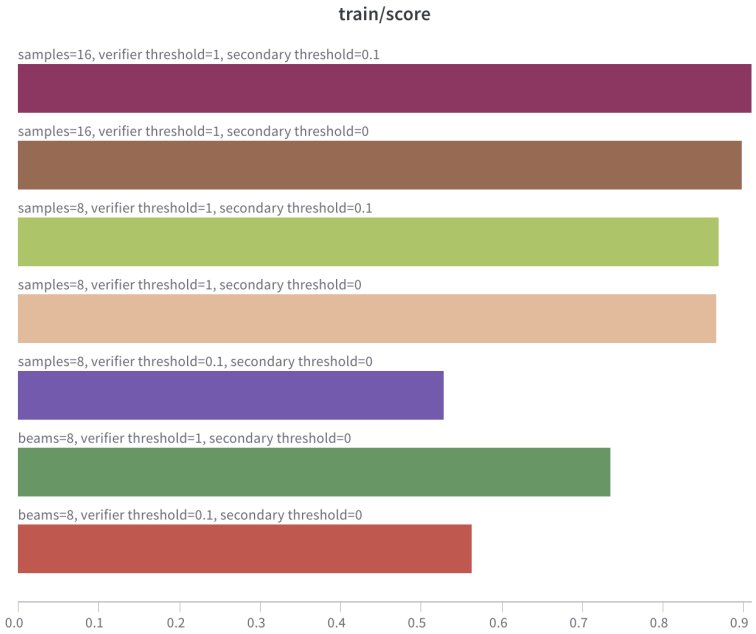


Figure 4: Test-time compute: impact of different verifier based strategies on average score across generations for development split of "Countdown-Tasks-3to4" dataset reduced to 200 observations.

Table 2: Math Reasoning

Method	Avg. score[1]	Training time	Inference time[3]	Final leaderboard score
Base Model	0.0000	-	30s	-
SFT	0.4700	20m35s[2]	30s	-
RLOO	0.5085	31m25s[2]	30s	0.3160
Test-time compute	0.9100	0s	98s	0.6715

[1] Average score across generations for development split of "Countdown-Tasks-3to4" dataset reduced to 200 observations.
[2] Includes 6 evaluation steps of 200 data points each.
[3] Per 200 data points.

7 Conclusion

This work has shown the importance of multi-pronged approach to fine-tuning and the relative strengths and weaknesses of different families of methods. The enormous potential of test-time methods shows that a thorough analysis of target uses cases of a fine-tuned model needs to be carried out to decide if the computational budget should be invested more towards optimisation or inference.

8 Team Contributions

- **Witold Gawlikowicz:** This was a solo project, hence all contributions are mine.

Changes from Proposal Have not managed to look into these stretch goals mentioned in the proposal: few shot learning using the base model without any additional fine-tuning, Chain-of-Thought verifiers (GenRM-CoT) Zhang et al. (2025), oracle-based approach where LLM is given an access to a simple algebraic calculator to verify the solution, preferably via a formalised tool use framework in a similar fashion to Andor et al. (2019) (would require re-training and we were already low on AWS credit and time towards the end of the project).

References

Arash Ahmadian, Chris Cremer, Matthias Gallé, Marzieh Fadaee, Julia Kreutzer, Olivier Pietquin, Ahmet Üstün, and Sara Hooker. 2024. Back to Basics: Revisiting REINFORCE Style Optimization for Learning from Human Feedback in LLMs. arXiv:2402.14740 [cs.LG] https://arxiv.org/abs/2402.14740

Loubna Ben Allal, Anton Lozhkov, Elie Bakouch, Gabriel Martín Blázquez, Guilherme Penedo, Lewis Tunstall, Andrés Marafioti, Hynek Kydlíček, Agustín Piqueres Lajarín, Vaibhav Srivastav, Joshua Lochner, Caleb Fahlgren, Xuan-Son Nguyen, Clémentine Fourrier, Ben Burtenshaw, Hugo Larcher, Haojun Zhao, Cyril Zakka, Mathieu Morlon, Colin Raffel, Leandro von Werra, and Thomas Wolf. 2025. SmolLM2: When Smol Goes Big – Data-Centric Training of a Small Language Model. arXiv:2502.02737 [cs.CL] https://arxiv.org/abs/2502.02737

Daniel Andor, Luheng He, Kenton Lee, and Emily Pitler. 2019. Giving BERT a Calculator: Finding Operations and Arguments with Reading Comprehension. In *Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing and the 9th International Joint Conference on Natural Language Processing (EMNLP-IJCNLP)*, Kentaro Inui, Jing Jiang, Vincent Ng, and Xiaojun Wan (Eds.). Association for Computational Linguistics, Hong Kong, China, 5947–5952. https://doi.org/10.18653/v1/D19-1609

Akhiad Bercovich, Itay Levy, Izik Golan, Mohammad Dabbah, Ran El-Yaniv, Omri Puny, Ido Galil, Zach Moshe, Tomer Ronen, Najeeb Nabwani, Ido Shahaf, Oren Tropp, Ehud Karpas, Ran Zilberstein, Jiaqi Zeng, Soumye Singhal, Alexander Bukharin, Yian Zhang, Tugrul Konuk, Gerald Shen, Ameya Sunil Mahabaleshwarkar, Bilal Kartal, Yoshi Suhara, Olivier Delalleau, Zijia Chen, Zhilin Wang, David Mosallanezhad, Adi Renduchintala, Haifeng Qian, Dima Rekesh, Fei Jia, Somshubra Majumdar, Vahid Noroozi, Wasi Uddin Ahmad, Sean Narenthiran, Aleksander Ficek, Mehrzad Samadi, Jocelyn Huang, Siddhartha Jain, Igor Gitman, Ivan Moshkov, Wei Du, Shubham Toshniwal, George Armstrong, Branislav Kisacanin, Matvei Novikov, Daria Gitman, Evelina Bakhturina, Jane Polak Scowcroft, John Kamalu, Dan Su, Kezhi Kong, Markus Kliegl, Rabeeh Karimi, Ying Lin, Sanjeev Satheesh, Jupinder Parmar, Pritam Gundecha, Brandon Norick, Joseph Jennings, Shrimai Prabhumoye, Syeda Nahida Akter, Mostofa Patwary, Abhinav Khattar, Deepak Narayanan, Roger Waleffe, Jimmy Zhang, Bor-Yiing Su, Guyue Huang, Terry Kong, Parth Chadha, Sahil Jain, Christine Harvey, Elad Segal, Jining Huang, Sergey Kashirsky, Robert McQueen, Izzy Putterman, George Lam, Arun Venkatesan, Sherry Wu, Vinh Nguyen, Manoj Kilaru, Andrew Wang, Anna Warno, Abhilash Somasamudramath, Sandip Bhaskar, Maka Dong, Nave Assaf, Shahar Mor, Omer Ullman Argov, Scot Junkin, Oleksandr Romanenko, Pedro Larroy, Monika Katariya, Marco Rovinelli, Viji Balas, Nicholas Edelman, Anahita Bhiwandiwalla, Muthu Subramaniam, Smita Ithape, Karthik Ramamoorthy, Yuting Wu, Suguna Varshini Velury, Omri Almog, Joyjit Daw, Denys Fridman, Erick Galinkin, Michael Evans, Shaona Ghosh, Katherine Luna, Leon Derczynski, Nikki Pope, Eileen Long, Seth Schneider, Guillermo Siman, Tomasz Grzegorzek, Pablo Ribalta, Monika Katariya, Chris Alexiuk, Joey Conway, Trisha Saar, Ann Guan, Krzysztof Pawelec, Shyamala Prayaga, Oleksii Kuchaiev, Boris Ginsburg, Oluwatobi Olabiyi, Kari Briski, Jonathan Cohen, Bryan Catanzaro, Jonah Alben, Yonatan Geifman, and Eric Chung. 2025. Llama-Nemotron: Efficient Reasoning Models. arXiv:2505.00949 [cs.CL] https://arxiv.org/abs/2505.00949

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. 2025. DeepSeek-R1: Incentivizing Reasoning Capability in LLMs via Reinforcement Learning. arXiv:2501.12948 [cs.CL] https://arxiv.org/abs/2501.12948

Yann Dubois, Xuechen Li, Rohan Taori, Tianyi Zhang, Ishaan Gulrajani, Jimmy Ba, Carlos Guestrin, Percy Liang, and Tatsunori B. Hashimoto. 2024. AlpacaFarm: A Simulation Framework for Methods that Learn from Human Feedback. arXiv:2305.14387 [cs.LG] https://arxiv.org/abs/2305.14387

Kanishk Gandhi, Ayush Chakravarthy, Anikait Singh, Nathan Lile, and Noah D. Goodman. 2025. Cognitive Behaviors that Enable Self-Improving Reasoners, or, Four Habits of Highly Effective STaRs. arXiv:2503.01307 [cs.CL] https://arxiv.org/abs/2503.01307

Kanishk Gandhi, Denise Lee, Gabriel Grand, Muxin Liu, Winson Cheng, Archit Sharma, and Noah D. Goodman. 2024. Stream of Search (SoS): Learning to Search in Language. arXiv:2404.03683 [cs.LG] https://arxiv.org/abs/2404.03683

Kanishk Gupta. 2023. Countdown Function in Reward Score Module. https://github.com/kanishkg/cognitive-behaviors/blob/main/verl/utils/reward_score/countdown.py#L59.

Edward J. Hu, Yelong Shen, Phillip Wallis, Zeyuan Allen-Zhu, Yuanzhi Li, Shean Wang, Lu Wang, and Weizhu Chen. 2021. LoRA: Low-Rank Adaptation of Large Language Models. arXiv:2106.09685 [cs.CL] https://arxiv.org/abs/2106.09685

Edward J Hu, Yelong Shen, Phillip Wallis, Zeyuan Allen-Zhu, Yuanzhi Li, Shean Wang, Lu Wang, and Weizhu Chen. 2022. LoRA: Low-Rank Adaptation of Large Language Models. In *International Conference on Learning Representations*. https://openreview.net/forum?id=nZeVKeeFYf9

Binyuan Hui, Jian Yang, Zeyu Cui, Jiaxi Yang, Dayiheng Liu, Lei Zhang, Tianyu Liu, Jiajun Zhang, Bowen Yu, Keming Lu, Kai Dang, Yang Fan, Yichang Zhang, An Yang, Rui Men, Fei Huang, Bo Zheng, Yibo Miao, Shanghaoran Quan, Yunlong Feng, Xingzhang Ren, Xuancheng Ren, Jingren Zhou, and Junyang Lin. 2024. Qwen2.5-Coder Technical Report. arXiv:2409.12186 [cs.CL] https://arxiv.org/abs/2409.12186

Ilya Loshchilov and Frank Hutter. 2019. Decoupled Weight Decay Regularization. arXiv:1711.05101 [cs.LG] https://arxiv.org/abs/1711.05101

Martin Mundt, Anaelia Ovalle, Felix Friedrich, A Pranav, Subarnaduti Paul, Manuel Brack, Kristian Kersting, and William Agnew. 2025. The Cake that is Intelligence and Who Gets to Bake it: An AI Analogy and its Implications for Participation. arXiv:2502.03038 [cs.AI] https://arxiv.org/abs/2502.03038

Long Ouyang, Jeff Wu, Xu Jiang, Diogo Almeida, Carroll L. Wainwright, Pamela Mishkin, Chong Zhang, Sandhini Agarwal, Katarina Slama, Alex Ray, John Schulman, Jacob Hilton, Fraser Kelton, Luke Miller, Maddie Simens, Amanda Askell, Peter Welinder, Paul Christiano, Jan Leike, and Ryan Lowe. 2022. Training language models to follow instructions with human feedback. arXiv:2203.02155 [cs.CL]

Jiayi Pan, Junjie Zhang, Xingyao Wang, Lifan Yuan, Hao Peng, and Alane Suhr. 2025. TinyZero. https://github.com/Jiayi-Pan/TinyZero. Accessed: 2025-04-19.

Charlie Snell, Jaehoon Lee, Kelvin Xu, and Aviral Kumar. 2024. Scaling LLM Test-Time Compute Optimally can be More Effective than Scaling Model Parameters. arXiv:2408.03314 [cs.LG] https://arxiv.org/abs/2408.03314

Xuezhi Wang, Jason Wei, Dale Schuurmans, Quoc Le, Ed Chi, Sharan Narang, Aakanksha Chowdhery, and Denny Zhou. 2023. Self-Consistency Improves Chain of Thought Reasoning in Language Models. arXiv:2203.11171 [cs.CL] https://arxiv.org/abs/2203.11171

A. Wigderson. 2019. *Mathematics and Computation: A Theory Revolutionizing Technology and Science*. Princeton University Press. https://books.google.pl/books?id=-WCqDwAAQBAJ

Adina Williams, Nikita Nangia, and Samuel Bowman. 2018. A Broad-Coverage Challenge Corpus for Sentence Understanding through Inference. In *Proceedings of the 2018 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long Papers)* (New Orleans, Louisiana). Association for Computational Linguistics, 1112–1122. http://aclweb.org/anthology/N18-1101

Ronald J. Williams. 1992. Simple statistical gradient-following algorithms for connectionist reinforcement learning. *Machine Learning* 8, 3 (1992), 229–256. https://doi.org/10.1007/BF00992696

Lunjun Zhang, Arian Hosseini, Hritik Bansal, Mehran Kazemi, Aviral Kumar, and Rishabh Agarwal. 2025. Generative Verifiers: Reward Modeling as Next-Token Prediction. arXiv:2408.15240 [cs.LG] https://arxiv.org/abs/2408.15240

A Additional Experiments

A.1 Math Reasoning

A.1.1 SFT

Figure 5 shows SFT results for Math Reasoning along with malformatted fraction of responses.

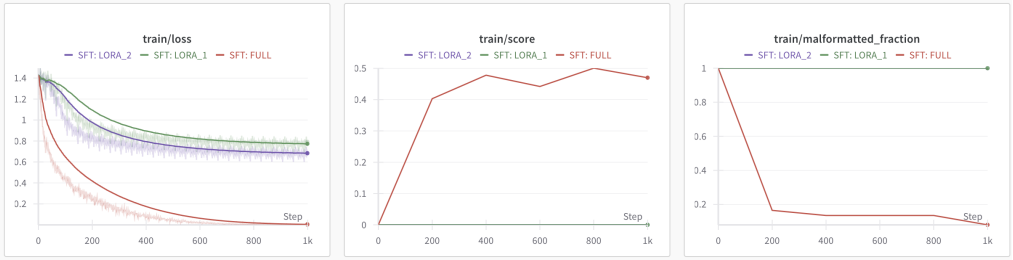


Figure 5: SFT (Math Reasoning): full fine-tuning vs LoRA.

A.1.2 RLOO

Figure 6 shows impact of KL divergence penalty (α) parameter on model's performance.
Figure 7 shows impact of learning rate parameter on model's performance.
Figure 8 shows impact of length penalty parameter on model's performance.
Figure 9 shows impact of number of generations (k) parameter on model's performance.

A.1.3 Test-time compute

Figure 10 shows impact of various generation paramters on model's performance.

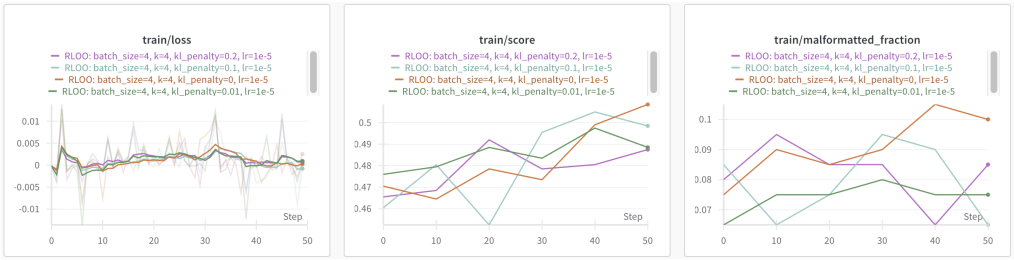


Figure 6: RLOO: impact of KL divergence penalty.

Figure 7: RLOO: impact of learning rate.

A.2 Instruction Following

A.2.1 DPO

Figure 11 shows impact of number of β parameter on model's performance.

B Implementation Details

We have used an M1 Macbook Pro with `EleutherAI/pythia-70m` model (a model optimised for CPUs) for a lot of the local development to save AWS credits.

All of the computations aimed improving the performance of tasks considered have been carried out on AWS G6e instance featuring a 48GB GPU (NVIDIA L40S). The reported runtimes correspond to that hardware.

We have used Hugging Face for downloading datasets and model weights. PyTorch has been used for all the tensor computations, optimisation and dataset handling (we have implemented our own data structures implementing PyTorch's `Dataset` class).

We have also implemented Weights & Biases integration for convenient reporting and monitoring during remote runs, as well as saving optimised models to private repositories on Hugging Face Hub. We have also implemented extensive logging to help with debugging and configuration files serialised to disk and uploaded to W&B after each run to assure reproducibility.

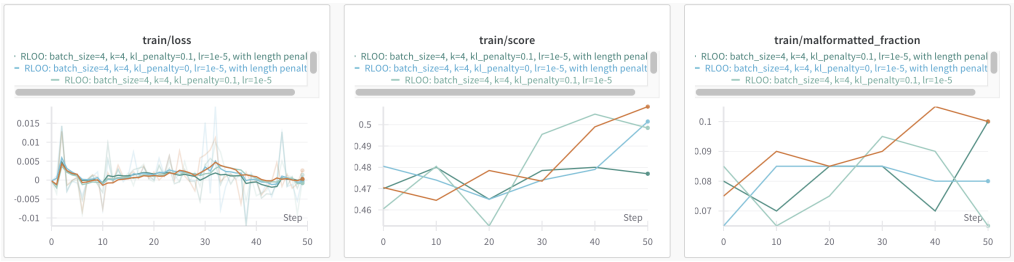


Figure 8: RLOO: impact of length penalty.

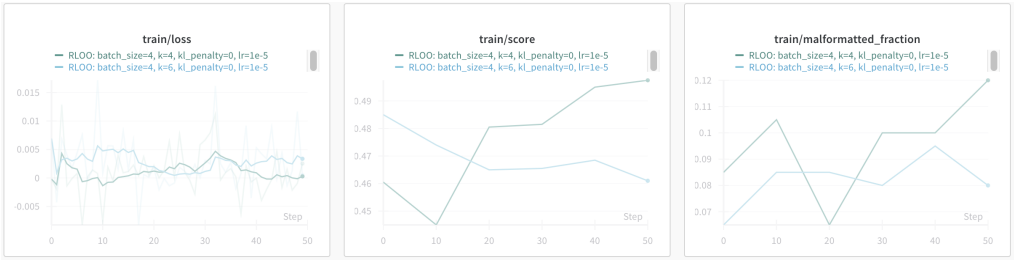


Figure 9: RLOO: impact of number of generations.

(a) Rep Pen (b) Top P (c) Top K

(d) Max Beam (e) Max Temp (f) Max Resp

Figure 10: Verifier external parameter analysis

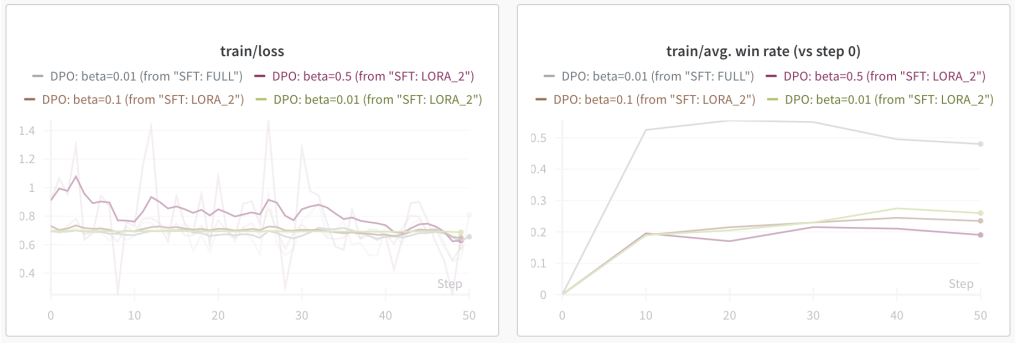


Figure 11: DPO: impact of β.