



Stockholms
universitet

Fine-Tuning Language Models with Preferences for Text Summarization Tasks: A Comparative Study of Reinforcement Learning from Human Feedback and Direct Preference Optimization

Fanny Walldén

Masteruppsats 2026:5
Matematisk statistik
Juni 2026

www.math.su.se

Matematisk statistik
Matematiska institutionen
Stockholms universitet
106 91 Stockholm

Fine-Tuning Language Models with Preferences for Text Summarization Tasks: A Comparative Study of Reinforcement Learning from Human Feedback and Direct Preference Optimization

Fanny Walldén*

June 2026

Abstract

The work investigates reinforcement learning from human feedback (RLHF) and direct preference optimization (DPO), two approaches used to fine-tune language models using human preference. The approaches are trained and evaluated for the text summarization task, using the ROUGE and BERTScore metrics. In addition, the language models are fine-tuned using AI preference obtained by an off-the-shelf language model to investigate their robustness to different sources of feedback. The results demonstrate the main disadvantage of RLHF, namely, the approach is prone to suffer from reward hacking. Reward hacking is a common problem in reinforcement learning that occurs because the objective diverges from the actual goal. In addition, the results demonstrate that the main advantage of DPO lies in its robustness and stability. Lastly, the results show that the approaches are robust to different sources of feedback, enabling efficient feedback generation without loss of quality.

*Postal address: Mathematical Statistics, Stockholm University, SE-106 91, Sweden.
E-mail: fawa8135@student.su.se. Supervisor: Chun-Biu Li.

Acknowledgments

First, I would like to express my deepest gratitude to my supervisor, Chun-Biu Li, for his guidance and for so generously sharing his expertise with me throughout this semester. I am also incredibly thankful to Taariq Fahren Nazar for his dedication to our weekly meetings and feedback, which have really helped shape this work. I am truly grateful for the time and energy that both have invested in this report.

Secondly, I am sincerely grateful to my supervisors at the Swedish Defence Research Agency, Agnes Tegen, Iza Smedberg, and Joel Brynielsson. In addition to invaluable feedback and suggestions for implementation, they have provided me with the necessary resources, which has made this work possible.

Lastly, my sincere thanks go to my fellow students and their supervisors at the Swedish Defence Research Agency for their feedback during our monthly meetings. Thanks to the inputs by primarily Alexander Tedestedt, Anton Johansson, Fredrik Lindholm, Jesper Söderström, Madeleine Lindström, and Marcus Zander, the report could be refined to its current form.

Contents

1	Introduction	1
1.1	Purpose and Research Question	2
1.2	Swedish Defence Research Agency	2
1.3	Report Structure	2
2	Theory	4
2.1	Transformer Architecture	4
2.1.1	Tokenization Layer	6
2.1.2	Embedding Layer	7
2.1.3	Positional Encoding Layer	7
2.1.4	Attention Layer	8
2.1.5	Residual Connection	11
2.1.6	Normalization Layer	12
2.1.7	Feed-Forward Layer	12
2.1.8	Outputs	12
2.2	Reinforcement Learning Policy Gradient Methods	13
2.2.1	Fundamentals	13
2.2.2	REINFORCE	16
2.2.3	Natural Policy Gradient	17
2.2.4	Trust Region Policy Optimization	19
2.2.5	Proximal Policy Optimization	21
2.3	Reinforcement Learning from Human Feedback	24
2.3.1	Supervised Fine-Tuning	25
2.3.2	Reward Modeling	26
2.3.3	Reinforcement Learning	28
2.4	Direct Preference Optimization	30
2.5	Low-Rank Adaption	32
2.6	Evaluation Metrics	34
2.6.1	ROUGE	34
2.6.2	BERTScore	35
3	Methodology	39
3.1	Dataset	39
3.1.1	Reinforcement Learning from Human Feedback Datasets	41
3.1.2	Direct Preference Optimization Datasets	43
3.1.3	AI Feedback	43
3.2	Pre-Trained Language Models	45
3.2.1	Qwen3-4B-Instruct-2507	46
3.2.2	Qwen3-30B-A3B-Instruct-2507	46
3.3	Experimental Setup	46

3.3.1	Optimization Algorithm	47
3.3.2	Regularization	48
3.3.3	Hyperparameters	49
4	Results and Analysis	51
4.1	Supervised Fine-Tuning Results	51
4.2	Reward Modeling Results	55
4.3	Reinforcement Learning Results	57
4.4	Direct Preference Optimization Results	60
5	Discussion	64
5.1	Stability	64
5.2	Performance	64
5.3	Feedback Robustness	65
5.4	Objective	65
5.5	Limitations and Improvements	66
6	Conclusions	69
	Bibliography	70
A	Mathematical Derivations	74
A.1	Kullback-Leiber Divergence	74
A.2	Identity	75
A.3	Importance Sampling	75
A.4	Generalized Advantage Estimator	76
A.5	Rewriting Objective	76
A.6	Rewriting Policy	76
B	Figures	77
B.1	Subreddit Distribution	77
B.2	Learning Curves	78
C	Tables	79
C.1	Generations	79
C.2	Hyperparameter Settings	81
C.3	Performance	81

List of Abbreviations

BERT bidirectional encoder representations from transformers.

BPE byte pair encoding.

DPO direct preference optimization.

DPOAI direct preference optimization (AI version).

GAE generalized advantage estimation.

GRPO grouped relative policy optimization.

KL Kullback-Leiber.

LoRA low-rank adaption.

MDP Markov decision process.

NLP natural language processing.

NPG natural policy gradient.

PPO proximal policy optimization.

RL reinforcement learning.

RLAI reinforcement learning (AI version).

RLAIF reinforcement learning from AI feedback.

RLHF reinforcement learning from human feedback.

RM reward modeling.

RMAI reward modeling (AI version).

ROUGE recall oriented understudy for gisting evaluation.

SFT supervised fine-tuning.

SFTAI supervised fine-tuning (AI version).

TD temporal difference.

TRPO trust region policy optimization.

Chapter 1

Introduction

Text summarization is the task of automatically generating a shorter version of a text that captures its most important information [1]. The technique is used extensively in a wide range of fields that require the processing of large amounts of data. As the demand for efficient summarization grows, researchers consistently propose novel text summarization methods. Text summarization methods are commonly divided into extractive and abstractive techniques. The former technique extracts whole fragments from a text to write a corresponding summary, resulting in truthful but potentially incoherent texts. The latter technique is able to interpret and rephrase the text, resulting in more human like summaries at the risk of factual errors. In this work, different approaches are studied to improve abstractive techniques.

Abstractive techniques were revolutionized by the transformer architecture [2]. Transformer-based language models demonstrate impressive capabilities, but have a couple of limitations. First, these models are prone to provide factual errors, especially in specialized domains where the model does not recognize the content. One approach to mitigate this disadvantage is targeted fine-tuning, in which case the model is trained on a domain-specific dataset. Targeted fine-tuning adapts the model’s vocabulary to the domain but does not necessarily produce summaries that align with the user’s intent, which leads to the second limitation. The second limitation is that transformer-based language models often have the risk of a misaligned objective. More specifically, these models are usually trained using a simple next-token prediction loss, which treats all mismatches between the target and the output the same regardless of the severity of the mismatch. To take severity into account and better adapt to human preference, the model can be fine-tuned on human feedback, which is the main idea of reinforcement learning from human feedback (RLHF).

RLHF is commonly divided into three core stages. In the first stage, a pre-trained language model is fine-tuned using supervised learning, which adapts the model to specific domains or queries. In the second stage, a reward model is trained to predict the human preference. In the final stage, the supervised fine-tuned language model is fine-tuned on human preference using the reward model and proximal policy optimization (PPO). RLHF demonstrate impressive capabilities, both in text summarization tasks and in other natural language processing (NLP) tasks [3], [4]. However, it is a complex and often unstable procedure [5]. As an alternative approach, [5] proposed direct preference optimization (DPO).

DPO optimizes the same objective as RLHF, but does not require a reward model or reinforcement learning. The approach is built on the key finding that the objective used in RLHF can be rewritten to not depend on a reward model. According to [5], DPO achieves performance comparable to, or even better than, RLHF in a wide range of NLP tasks.

The drawback with both RLHF and DPO is that they rely on human annotated data. More specifically, both approaches are trained on datasets consisting of inputs, preferred outputs, and rejected outputs. The datasets are obtained by letting human annotators rank outputs in a pairwise manner, which is time-consuming. To overcome this limitation, a language model can be used to rank the outputs. The dataset can then be used to train RLHF and DPO, in which case those language models are said to be fine-tuned using AI feedback rather than human feedback. According to [6], reinforcement learning from AI feedback (RLAIF) achieves a performance comparable to, or even better than, RLHF in a wide range of NLP tasks. However, how RLHF and its corresponding AI version compare with DPO and its corresponding AI version remains an unanswered question that this work aims to answer.

1.1 Purpose and Research Question

The purpose of the work is to compare RLHF and DPO when used for the text summarization task in terms of different aspects. More precisely, the report aims to answer the research question:

How do RLHF and DPO compare in terms of objective, stability, performance, and robustness to AI feedback?

To answer the research question, the respective objectives are derived in Sections 2.3 and 2.4. The stability of the approaches are studied using the learning curves in Chapter 4. The chapter also includes the ROUGE and BERTScore metrics used to compare performance in Chapter 5. Lastly, the robustness to AI feedback is discussed in Chapter 5 using learning curves and evaluation metrics.

1.2 Swedish Defence Research Agency

The work is carried out in collaboration with the Swedish Defence Research Agency, which is a government agency that conducts research within defense and security. The rapid development of machine learning algorithms has made the agency devote more resources to research within this field. One particular part of machine learning the agency is interested in is interactive machine learning, and especially how that can be applied to process domain-specific documents.

1.3 Report Structure

The remainder of the report is organized into five chapters. In Chapter 2, the theoretical foundations needed to fully grasp and appreciate the methods discussed

in this work are explained. The report continues with Chapter 3, where implementing details of the experiments are provided. The following chapter, Chapter 4, presents the learning curves, the evaluation metrics, and an example generation for each model of all approaches. The results are further discussed in Chapter 5 together with the limitations and proposed improvements. Lastly, Chapter 6 presents the main messages of the work, acting as a summary.

Chapter 2

Theory

In this chapter, the theory needed to fully grasp the report is presented and explained. The chapter begins with the transformer architecture in Section 2.1. The chapter continues with reinforcement learning policy gradient methods in Section 2.2. The section covers fundamentals, the REINFORCE algorithm, natural policy gradients, trust region policy optimization, and proximal policy optimization. In this work, only proximal policy optimization is used, but to fully understand and appreciate that algorithm, the predecessor algorithms are also included. The first two sections lay the foundation for understanding the approaches in this report, while the following two sections present the approaches. To this end, the different stages of reinforcement learning from human feedback are explained in Section 2.3, while the alternative approach, direct preference optimization, is explained in Section 2.4. The remaining two sections, Sections 2.5 and 2.6, present low-rank adaption and the evaluation metrics ROUGE and BERTScore.

Throughout this chapter, several different notations for the parameter vector are used to clearly separate the different stages of reinforcement learning from human feedback. The supervised fine-tuning stage uses the notation ω , while θ is used for the reward modeling stage and ϕ is used for all reinforcement learning algorithms.

2.1 Transformer Architecture

Language models are trained to learn the probability distribution of natural language and generate natural language, and are commonly used in NLP tasks. The field of NLP was revolutionized by deep learning architectures, such as recurrent neural networks. Although such networks achieve impressive performance, they process sequences sequentially, which slows down training, and struggle with modeling long-term dependencies. To overcome these limitations, [2] proposed the transformer architecture. The transformer architecture further revolutionized the field of NLP through its use of attention, explained in Section 2.1.4. At a high level, the transformer architecture processes the input through multiple encoder layers to create a representation of the input. Given the representation, the corresponding output is generated through multiple decoder layers. The encoder and decoder layers, with corresponding sublayers, are visualized by the blue boxes in Figure 2.1. In addition, the input processing steps are visualized by the green boxes, while the output processing steps are visualized by the purple boxes in Figure 2.1.

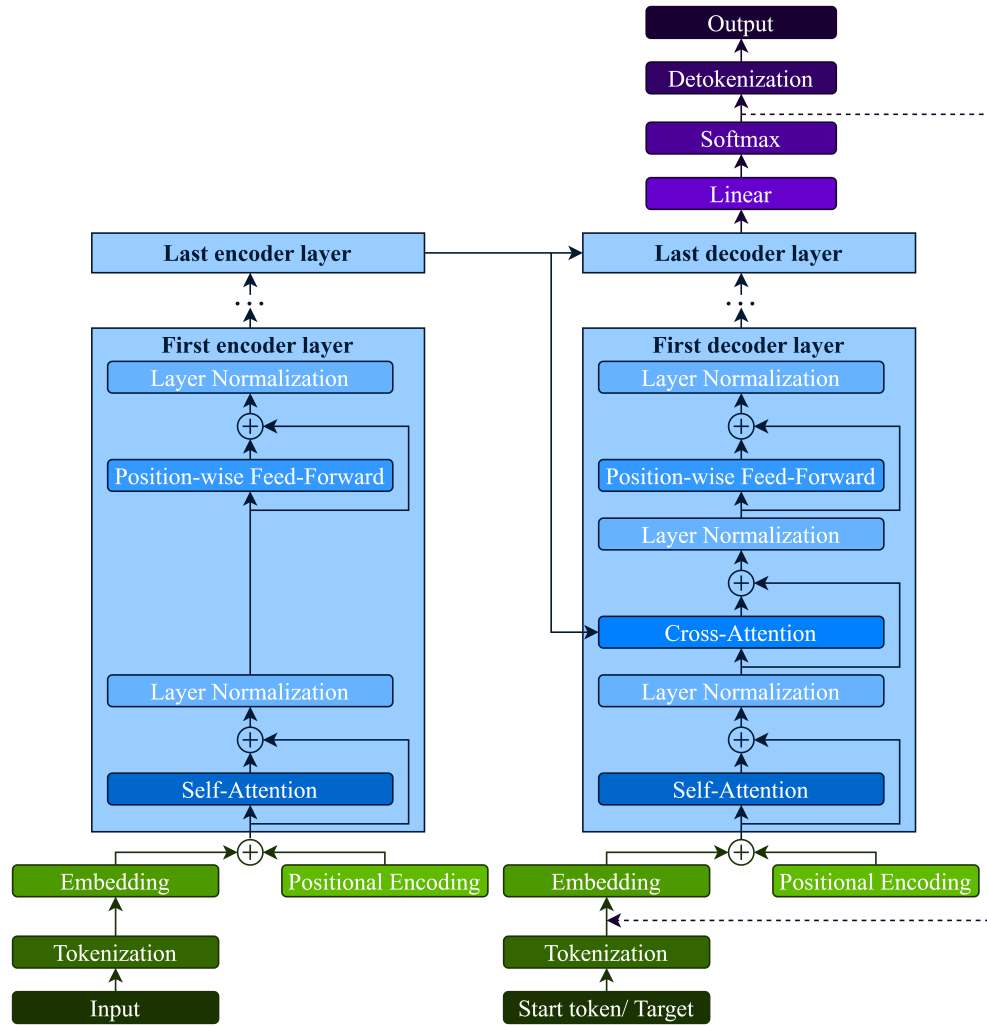


Figure 2.1: Transformer architecture. The input processing of the encoder and decoder is colored green. The first and last layers of the encoder and decoder are colored blue, while the intermediate layers are represented by the three dots. The decoder output processing is colored purple. Arrows that go around the sublayers in the encoder and decoder represent residual connections. Arrows from the last encoder layer go to the cross-attention sublayer of all decoder layers. The dashed arrow is used only during inference. During inference, the start token is used and the remaining elements in the sequence are predicted sequentially. During training, a target that demonstrates the desired behavior is used.

The input processing of the decoder starts with a target text or a start token, as visualized in Figure 2.1, depending on whether the model is training or doing inference. During training, a target text that demonstrates desired behavior is used. The predicted probability distribution for each position in the target text is then compared with the corresponding target element to update the learnable weights. During inference, a start token is used, and the remaining elements of the sequence are predicted sequentially. More specifically, the model predicts the first element, which is fed to the decoder to predict the second element. The procedure is repeated until the model predicts that the sequence has ended.

2.1.1 Tokenization Layer

The transformer architecture cannot actually read natural language, but performs linear algebra on real numbers. Because of this, the input processing of the encoder starts with transforming the input format in the tokenization layer. In the tokenization layer, the input text is split into small, meaningful units called tokens, where the splitting rule is defined by the tokenization approach. There exist multiple tokenization approaches, but the most intuitive is word tokenization which splits the text into words as visualized by the upper box in Figure 2.2. The drawback of word tokenization is that it gives a large vocabulary, defined as the set of unique tokens that the model has trained on and recognizes, which increases time complexity and space complexity [7]. An alternative approach is character tokenization, which splits the text into characters as visualized by the middle box in Figure 2.2. The drawback with character tokenization is that it gives a large sequence length, making it difficult to learn the relationships between tokens [7]. The intermediate approach is subword tokenization. Subword tokenization offers a promising tradeoff between vocabulary size and sequence length, while avoiding the out-of-vocabulary problem that word tokenization suffers from. More specifically, in contrast to word tokenization, which treats all out-of-vocabulary words as unknown, subword tokenization can obtain a representation of an unknown word by combining the known parts of the word. Several subword tokenization approaches exist, including byte pair encoding (BPE) tokenization. BPE tokenization splits the text into characters and iteratively merges the most common pairs of tokens until a decided vocabulary size is obtained, as visualized by the bottom box in Figure 2.2. Alternative subword tokenization approaches use more sophisticated merging rules or start with the words and iteratively split these.

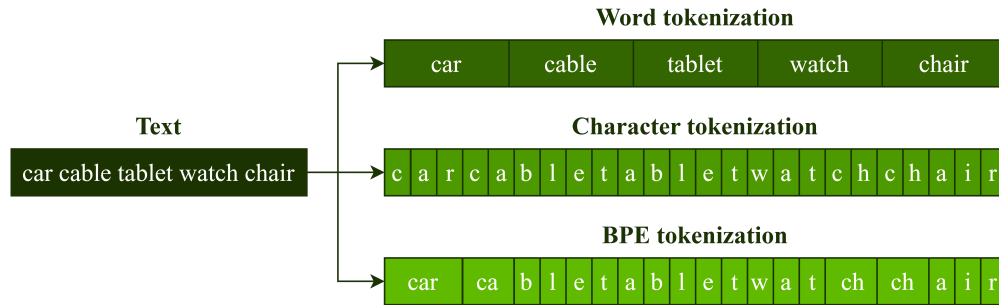


Figure 2.2: Different tokenization approaches. Word tokenization is visualized by the upper green box. Character tokenization is visualized by the middle green box. BPE tokenization is visualized by the bottom green box. The example text and tokenization results are provided by [7].

The first part of the tokenization layer is to transform the input text into a sequence of tokens, as decided by the tokenization approach. The second part of the tokenization layer is to transform the sequence of tokens into a vector of natural numbers. To this end, each token is assigned a fixed numeric ID. That is, the token “car” produced by the word tokenization approach in Figure 2.2 is assigned a natural number such as 24. The procedure, for a single sample, gives a vector $T \in \mathbb{N}^n$ where n is the length of the sequence and each element represents a token in that sequence.

2.1.2 Embedding Layer

The tokenization layer produces a vector containing numeric IDs, where each ID represents a token in the sequence. Although these IDs enable numerical computations, they do not provide any semantic relationship between tokens. In other words, the numerical difference between IDs for the tokens “Dog” and “Cat” might be the same as that between “Dog” and “Computer”, despite their differing semantic similarities. To represent the semantic relationship, each ID is assigned an embedding vector in the embedding layer. That is, the ID 24 corresponding to the token “car” in the previous section is assigned a embedding vector such as $[18, \dots, 171]^T$. The embedding vector is a fixed vector for each ID, whose length is commonly referred to as the embedding size. The procedure, for a single sample, gives a matrix $E \in \mathbb{R}^{n \times d}$ where n is the length of the sequence and d is the length of the embedding vector.

2.1.3 Positional Encoding Layer

The transformer architecture enables parallel computations through its attention sublayers, explained in Section 2.1.4. The ability to process each token of the sequence in parallel is one of the main advantages of the transformer architecture because it speeds up training. The disadvantage of parallel computation is that the positional information about each token is lost, which can be a problem when dealing with NLP tasks, in which case order matters. Fortunately, the order of the tokens can be preserved by adding a positional embedding matrix to the output of the embedding layer. The positional embedding matrix is of the same dimension as the embedding layer output and contains fixed constants for each position in the sequence and embedding. These constants are computed as

$$p_{i,2j} = \sin\left(\frac{i}{10000^{2j/\text{embedding size}}}\right), \quad (2.1)$$

$$p_{i,2j+1} = \cos\left(\frac{i}{10000^{2j/\text{embedding size}}}\right). \quad (2.2)$$

The rows of the positional embedding matrix represent the positions within a sequence, while the columns represent the positions within the embedding vector. Alternative positional encoding schemes exist, but sine and cosine functions are advantageous for several reasons. First, they capture the absolute positional information through the different frequency representations. That is, low embedding dimensions give short wavelengths, which captures local positional information, while large embedding dimensions give long wavelengths, which captures global positional information. Secondly, the sine and cosine functions are bounded, which enables positional embedding of sequences of variable length. Third, they capture relative positional information, which allows one position to attend to another position through a linear transformation rather than storing that relative information. More specifically, any pair $(p_{i+\kappa,2j}, p_{i+\kappa,2j+1})$ can be represented as a linear transformation of $(p_{i,2j}, p_{i,2j+1})$.

Input processing consists of three parts. The first part is tokenization, which transforms the input text into a vector of natural numbers. The second part is embedding, which represents the semantic relationship between tokens. The third part is the positional encoding, which preserves the positional information about each token. In Figure 2.3, the process for 5 samples is visualized.

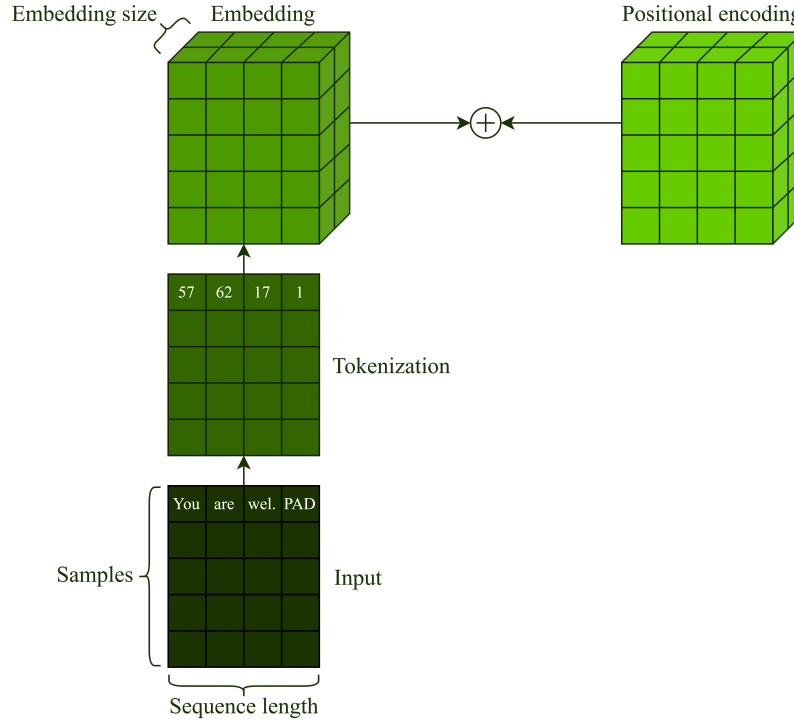


Figure 2.3: Input processing. Input processing consists of three parts. The first part is tokenization, colored dark green. The second part is embedding, colored light green. The third part is positional encoding, colored the lightest green. The number of samples is 5, where the first sample is the example provided by [8]. The length of the sequence is 4 and the embedding size is 2.

Matrix operations are a vital part of the transformer architecture, and to enable these, all samples must be of the same length. In practice, different sequences of tokens often have different lengths and due to this, padding tokens are inserted as visualized in Figure 2.3.

2.1.4 Attention Layer

The input processing output gives the encoder input. The input of the encoder is fed to the first sublayer of the first encoder layer, which is a multi-head self-attention layer, as visualized in Figure 2.1. At a high level, the attention mechanism measures the relationship between different tokens in sequences, allowing the model to focus on the most important parts. More specifically, consider a single sample stored in a matrix whose rows represent the position within the sequence and whose columns represent the position within the embedding vector. The matrix is first fed through a linear layer, where it is multiplied by the weight matrices $W^Q, W^K, W^V \in \mathbb{R}^{d \times d}$, respectively. The weight matrices contain learnable weights where d is the length of the embedding vector. The procedure gives three products, a query Q that determines what to focus on, a key K that contains the available data, and a value V that gives the values. In the regression example provided by [9], the query determines the location of the regression, the key contains the locations of the observations, and the value gives the regression values for those observations. Given the products, attention is defined as

$$\text{Attention}(Q, K, V) = \text{softmax}\left(\frac{QK^T}{\sqrt{\text{embedding size}}}\right)V. \quad (2.3)$$

The dot product QK^T in the attention gives a similarity score between all pairs of tokens in the sequence. The advantage of the dot product is that it is computationally fast to compute compared to other distance functions. The disadvantage of the dot product is that the variance grows with the embedding size. More specifically, it is assumed that the components of the query and the key are random variables with mean 0 and variance 1. Computing the dot product between a query vector and a key vector therefore gives a variance equal to the embedding size. Since the embedding size is commonly large, the variance of the dot product is large, which pushes the softmax operation to extreme cases where the gradient vanishes during backpropagation. Vanishing gradients are a common problem in deep learning that slow down or stop training of earlier layers in the network. To avoid this, the similarity scores obtained from the dot product are scaled by their standard deviation, that is, by $\sqrt{\text{embedding size}}$. The scaled dot product is commonly known as the scaled dot product attention scoring function. Furthermore, the scaled similarity scores are appropriately represented relative to each other, and to this end, the softmax operation is used. The softmax operation ensures that the scaled similarity scores are not negative and sum to 1, and in that way are represented as a probability relative to each other. These probabilities are called scalar attention weights. The scalar attention weights are then used to weight the elements of the value matrix, which is why the procedure is called attention. Particular attention is paid to elements in the value matrix for which the scalar attention weights are large [9].

To increase performance, the transformer architecture uses multi-head attention. Multi-head attention is a procedure that computes multiple attentions on different representations of the input in parallel. More specifically, in multi-head attention, K , V , and Q are all split into h matrices, denoted by K_i , V_i and Q_i for $i = 1, \dots, h$. In each of these matrices, the rows represent the position within the sequence, while the columns represent the position within the query size, which is defined as

$$\text{query size} = \frac{\text{embedding size}}{h} \quad (2.4)$$

[10]. The splitting procedure gives h representations of the input, each in the form (K_i, V_i, Q_i) . To capture the different dependencies in these representations, they are all fed to the attention mechanism in parallel, resulting in h outputs. These outputs are commonly called heads and, more formally, computed as

$$\text{head}_i = \text{Attention}(Q_i, K_i, V_i). \quad (2.5)$$

The last part of the multi-head attention is to transform the different heads into one matrix of the original form. To this end, the heads are first concatenated. Concatenation essentially reverses the splitting operation by joining the head matrices along their query dimension to form a single matrix. The resulting matrix is then multiplied by a matrix $W^o \in \mathbb{R}^{d \times d}$ of learnable weights, where d is the lengths of the embedding vector. More precisely, the resulting multi-head attention output is given by

$$\text{MultiHead}(Q, K, V) = \text{Concat}(\text{head}_1, \dots, \text{head}_h)W^o \quad (2.6)$$

$$= [\text{head}_1, \dots, \text{head}_h]W^o. \quad (2.7)$$

The procedure allows the model to capture different dependencies, such as long-term and short-term dependencies, of the input. The multi-head attention output is therefore a richer representation of the input compared to that of single-head attention, which improves the performance for a appropriate number of heads. The number of heads is a hyperparameter to be decided. Few heads mitigate the advantages of multi-head attention, whereas many heads give a query size too small to effectively learn dependencies.

The key difference between the encoder layers and the decoder layers is that the encoder layers include a single self-attention sublayer, whereas the decoder layers include a self-attention sublayer and a cross-attention sublayer. The self-attention sublayer of the encoder relates all tokens in the input sequence. It is called self-attention because the query, key, and value come from the same place. More specifically, the self-attention sublayer of the first encoder layer takes its query, key, and value from the encoder input. The remaining encoder layers take their query, key, and value from the previous encoder layer output. The self-attention sublayers of the decoder layers work similarly but relates all tokens in the target sequence instead of the input sequence. To this end, the self-attention sublayer of the first decoder layer takes its query, key, and value from the decoder input. The second sublayer of the decoder layer is the cross-attention sublayer. The cross-attention sublayer is commonly called the encoder-decoder attention because it takes its query, key, and value from both the encoder and decoder. More specifically, the cross-attention sublayer takes its key and value from the encoder output and its query from the previous decoder sublayer output. This procedure enables the model to relate the tokens in the input sequence to the tokens in the target sequence.

Another difference between the attention sublayers in the encoder layers and the decoder layers is their use of masking. In Figure 2.3, a padding token is added to the end of the example sequence to ensure that all samples are of the same dimension. The padding token does not provide any information and could therefore be disregarded when computing the attention. To this end, a mask matrix is added to the softmax operation in the attention function, given in Equation (2.3). That is, the attention function is rewritten as

$$\text{Attention}(Q, K, V) = \text{softmax}\left(\frac{QK^T}{\sqrt{\text{embedding size}}} + M\right)V, \quad (2.8)$$

where $M \in \mathbb{R}^{n \times n}$ is the mask matrix and n is the length of the sequence. In the mask matrix, the elements corresponding to the padding tokens are set to negative infinity, as visualized by the right mask matrix in Figure 2.4. Setting the elements to negative infinity ensures that the softmax operation returns these as zero, while the remaining elements of the mask matrix are set to zero to ensure that they remain unchanged. The encoder self-attention sublayer and the decoder cross-attention sublayer use this sort of masking. The decoder self-attention sublayer uses masking to preserve autoregression. That is, to ensure that the predictions

for a certain timestep depend only on the outputs generated before that timestep. During training, the target sequence is known, allowing the model to look at the answer instead of predicting it. Masking forces the model to actually predict the sequence by letting the mask matrix in Equation (2.8) be a lower triangular matrix, as visualized by the left mask matrix in Figure 2.4. Like before, the elements that should be masked are set to negative infinity so that the softmax operation returns these as zero.

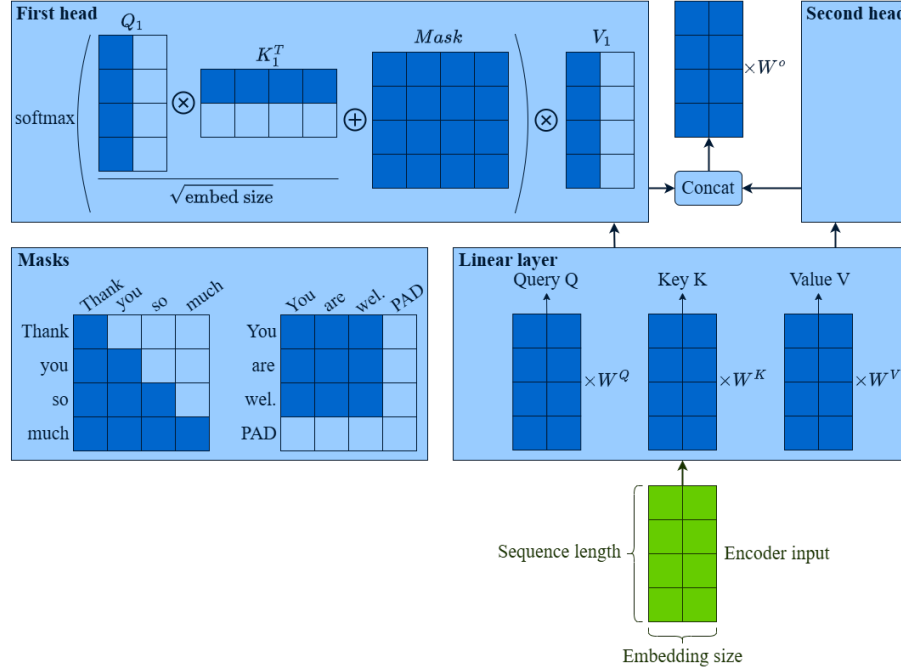


Figure 2.4: Attention layer. The number of samples is 1, the sequence length is 4 and the embedding size is 2. The number of heads is 2, resulting in query size 1. The darker elements of the matrices in the first head represent the elements that belong to that split. The lighter elements represent those that do not belong to that split. The mask matrix in the softmax operation is to be replaced by one of those given in the lower left box. In the encoder self-attention sublayer and the decoder cross-attention sublayer, the matrix on the right is used, while the decoder self-attention sublayer uses the matrix on the left. The darker elements of the mask matrices represent zero and correspond to the elements that should be kept. The lighter elements represent negative infinity and correspond to the elements that should be masked. The components of the second head are omitted from the figure, but are similar to the first head. The second head uses the second column of the query, key and value matrices in contrast to the first head in which the first column is used.

2.1.5 Residual Connection

The transformer architecture contains many layers as visualized in Figure 2.1 and is a so-called deep network. The main limitation with training a deep network is that they are prone to suffer from vanishing gradients. Vanishing gradients is a common deep learning problem that arises because the gradients of a network become small during backpropagation, which stops or slows the training of earlier

layers. To mitigate this drawback, the transformer architecture uses residual connections around the attention sublayers as visualized by the arrows in Figure 2.1. These connections allow the input of a sublayer to pass directly to the sublayer output. More specifically, given an input x to a sublayer, the residual connection outputs $x + \text{Sublayer}(x)$. The procedure results in higher gradients during backpropagation, which prevents vanishing gradients and accelerates learning [11].

2.1.6 Normalization Layer

The residual connections are followed by layer normalization, which further stabilizes and accelerates the training. Layer normalization is similar to batch normalization but is better adjusted to be combined with the attention mechanism. Batch normalization normalizes across the samples using the formula

$$\text{Normalized value} = \frac{\text{Original value} - \mu}{\sigma}, \quad (2.9)$$

where μ is the mean and σ is the standard deviation of the samples [12]. The reason why batch normalization struggle when combined with attention mechanisms is because the sample dimension contains multiple zeros. Recall that padding tokens are added to ensure that all samples have the same length. The padding tokens do not contain any information and are therefore masked during the attention layers, resulting in matrices with multiple zero elements. Layer normalization, on the other hand, normalizes across the embedding dimension, making it sample independent. The normalization layer outputs $\text{LayerNorm}(x + \text{Sublayer}(x))$, where $x + \text{Sublayer}(x)$ is the residual connection output.

2.1.7 Feed-Forward Layer

The output of the previous normalization sublayer is given as input to a fully connected two-layer position-wise feed-forward network. The feed-forward network allows for a richer representation of the embedding vectors by expanding the vectors and introducing non linearity. More specifically, if the input to the network is denoted by x , then the network outputs

$$\text{FFN}(x) = \max(0, xW_1 + b_1)W_2 + b_2. \quad (2.10)$$

In Equation (2.10), W_1 is a weight matrix that contains learnable weights and expands the input dimension. To this end, the weight matrix is usually chosen as $W_1 \in \mathbb{R}^{d \times 4d}$. Similarly, W_2 is a weight matrix of learnable weights that transforms the input matrix back to its original form. To this end, it is usually chosen as $W_2 \in \mathbb{R}^{4d \times d}$. Furthermore, b_1 and b_2 are bias vectors that consist of learnable biases, and the non linearity is introduced through the ReLU activation function, that is, through the max operation in Equation (2.10). The network is called position-wise because the same weight matrices are used independently of the position in the sequence [9].

2.1.8 Outputs

The feed-forward sublayer of the last decoder layer is followed by a residual connection and layer normalization, which gives the decoder output. The decoder output is then used to generate the probability distributions through a linear layer followed by a softmax layer, as visualized in Figure 2.1. In the linear layer, the

embedding vector for a token is transformed into a vector of word scores. The vector is of the same length as the target vocabulary, where higher scores make the model more likely to choose that token in that position. The word scores are then transformed into probabilities in the softmax layer. In the softmax layer, the scores are first divided by a hyperparameter called the temperature, and then the softmax operation is applied. The temperature is commonly chosen as a value between 0 and 2 [13]. Low temperature values increase the difference between probabilities, making the model more deterministic. High temperature values reduce the difference between probabilities, making the model more stochastic. The probabilities returned by the softmax operation are then used to define the probability of choosing each token in the vocabulary for that position. Finally, the resulting output text is obtained by detokenization, which essentially reverses the tokenization by transforming the vector of numerical IDs into a text.

2.2 Reinforcement Learning Policy Gradient Methods

Reinforcement learning is about learning to act through interactions to maximize the sum of rewards received over time. The field consists of multiple different algorithms that solve a problem that is commonly formalized as a Markov decision process (MDP). In the following sections, the problem of MDPs is explained together with some policy-based reinforcement learning solutions.

2.2.1 Fundamentals

MDPs are a classical formalization of sequential decision making, where decisions affect both immediate rewards and subsequent situations, which in turn affect future rewards [14]. More precisely, MDPs involve an agent, which is the decision maker, and an environment the agent interacts with. Given the current situation, the agent selects a decision as visualized by the green box and arrow in Figure 2.5. Following the decision, the environment presents a reward and a new situation to the agent, as visualized by the blue box and the arrows in Figure 2.5. Given the new situation, the agent selects a new decision, and through these interactions, the agent seeks to learn what decisions to select to maximize the cumulative rewards, defined as the sum of rewards received over time.

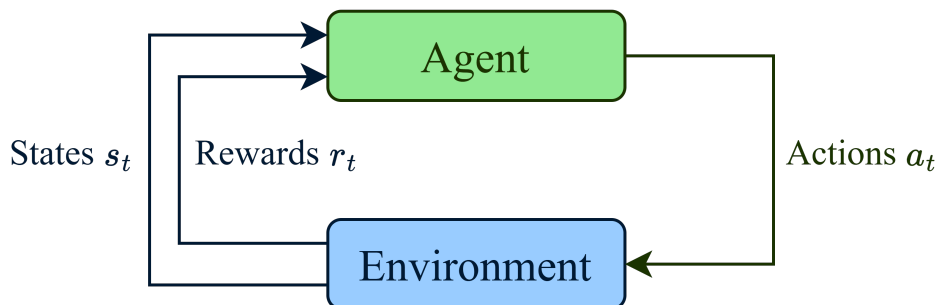


Figure 2.5: Agent-environment interaction in a MDP. The green box and arrow represent the agent and the decision made by the agent, respectively. The blue box represents the environment, while the blue arrows represent the reward and the new situation the environment presents to the agent, respectively.

In MDPs, the situation the agent is in is called the state, while the decision taken in that state is called the action. Consider, for instance, the game Ludo, where the state presented to the agent is the game setup and the action is the chosen piece to move. In each round, the agent moves a piece, after which the environment transitions to a new game setup. The transition is affected by the moves taken by the other players and the randomness of the dice. In addition, the environment presents a reward in each round. The reward should guide the agent towards the goal of reaching home with all its pieces. Therefore, an example reward function gives +1 for moving a piece towards home, +10 for reaching home, and -10 for being knocked out by an opponent. More formally, in the timestep t , the agent is in a state s_t and takes an action a_t . Based on the state and action, the agent receives a reward r_t and moves to the next state s_{t+1} . The procedure gives rise to the trajectory

$$\tau = (s_0, a_0, r_0, s_1, a_1, r_1, \dots). \quad (2.11)$$

The probability that the agent takes action a_t when in state s_t is given by the policy $\pi_\phi(a_t|s_t)$ with the parameter vector ϕ . The policy is called deterministic if it maps every state to a certain action with probability 1, otherwise it is called stochastic. The choice between these and the level of stochasticity is called the exploration-exploitation tradeoff. The agent should exploit the optimal actions found, but occasionally explore new actions to find better ones. Furthermore, the transition between states is not necessarily deterministic, but might be affected by some stochasticity, such as a dice. Therefore, the probability of moving to state s_{t+1} when in state s_t and taking action a_t is defined by the transition probabilities $p(s_{t+1}|s_t, a_t)$. Given the transition probabilities, the policy and the initial state distribution $p(s_0)$, the probability of observing a given trajectory is given by the trajectory probability defined as

$$p_\phi(\tau) = p(s_0) \prod_{t=0}^{\infty} \pi_\phi(a_t|s_t) p(s_{t+1}|s_t, a_t). \quad (2.12)$$

Apart from the trajectory probability, the trajectory return is of interest to write a formalization of the goal. The trajectory return is the cumulative rewards obtained on a trajectory and can be defined with and without a discount factor $0 \leq \gamma \leq 1$. More commonly, the discount factor is used and the corresponding discounted trajectory return is defined as

$$G_0 = r_0 + \gamma r_1 + \gamma^2 r_2 + \dots = \sum_{t=0}^{\infty} \gamma^t r_t. \quad (2.13)$$

The discounted trajectory return allows the agent to prioritize immediate rewards through the discount factor. Low discount factors prioritize immediate rewards, which could lead the agent in the Ludo game example to always move the piece closest to home. High discount factors make the agent value long-term rewards more, which enables the agent in the Ludo game example to balance the immediate reward of moving a piece in front of an opponent with the risk of getting knocked out by said opponent in the following rounds. The goal is to maximize cumulative rewards, and to achieve this, there exist two primary categories of reinforcement learning methods. The first category of methods is value-based methods, which learn a value function and use this to define a policy. One such value function is the state-value function defined as

$$v_{\pi_\phi}(s_t) = \mathbb{E}_{\pi_\phi} [G_t | s_t] = \mathbb{E}_{\pi_\phi} \left[\sum_{k=0}^{\infty} \gamma^k r_{t+k} \middle| s_t \right]. \quad (2.14)$$

The state-value function, given in Equation (2.14), gives the expected return when in state s_t and following policy π_ϕ . Intuitively, it measures how good a given state is. Another value function is the action-value function defined as

$$q_{\pi_\phi}(s_t, a_t) = \mathbb{E}_{\pi_\phi} [G_t | s_t, a_t] = \mathbb{E}_{\pi_\phi} \left[\sum_{k=0}^{\infty} \gamma^k r_{t+k} \middle| s_t, a_t \right]. \quad (2.15)$$

The action-value function, given in Equation (2.15), gives the expected return when taking action a_t in state s_t and then following policy π_ϕ . Intuitively, it measures how good a given action is in a given state. In value-based methods, the value function is updated using the temporal difference (TD) error

$$\delta_t = r_t + \gamma \hat{v}_{\pi_{\phi_{old}}}(s_{t+1}) - \hat{v}_{\pi_{\phi_{old}}}(s_t), \quad (2.16)$$

which gives the estimation error at each timestep. Given a learned value function, the agent chooses the action that maximizes the value function in each state. That is, value-based methods learn a policy indirectly through the value function in contrast to policy-based methods, which directly learn a policy. To this end, policy gradient methods follow the updating rule

$$\phi \leftarrow \phi_{old} + \alpha \nabla_\phi \mathcal{L}(\phi_{old}). \quad (2.17)$$

In Equation (2.17), $0 \leq \alpha \leq 1$ is a hyperparameter called the learning rate that decides how much to update the parameter vector ϕ . Low α values give stable but slow training, whereas high α values give fast but potentially unstable training. Furthermore, in Equation (2.17), $\nabla_\phi \mathcal{L}(\phi_{old})$ defines the direction of the update, which should align with the goal of maximizing the return. Therefore, the objective $\mathcal{L}(\phi_{old})$ can be formulated as finding the parameter vector ϕ for which the policy maximizes the expected discounted trajectory return over any trajectory. More precisely, the objective can be written as

$$\mathcal{L}(\phi) = \mathbb{E}_\tau [G_0] \quad (2.18)$$

$$= \sum_{\tau} p_\phi(\tau) G_0. \quad (2.19)$$

Unfortunately, taking the gradient of the objective requires taking the gradient of the trajectory probability, which includes the transition probabilities. In model-free reinforcement learning, the agent learns without a model of the environment. That is, the Ludo game agent does not know the probability distribution of the dice or how the other players behave, it just learns that certain action-state pairs give high rewards. Without a model of the environment, the effect of policy changes on the transition probabilities is unknown. Fortunately, the policy gradient theorem proves that the gradient of the objective can be written without including the transition probability gradient. More precisely, the gradient of the objective can be written as

$$\nabla_{\phi} \mathcal{L}(\phi) = \sum_{\tau} G_0 \nabla_{\phi} p_{\phi}(\tau) \quad (2.20)$$

$$= \sum_{\tau} G_0 p_{\phi}(\tau) \nabla_{\phi} \log p_{\phi}(\tau) \quad (2.21)$$

$$= \mathbb{E}_{\tau} [G_0 \nabla_{\phi} \log p_{\phi}(\tau)] \quad (2.22)$$

$$= \mathbb{E}_{\tau} \left[\sum_{t=0}^{\infty} G_0 \nabla_{\phi} \log \pi_{\phi}(a_t | s_t) \right] \quad (2.23)$$

[15]. Equation (2.20) is equal to the gradient of the entire expression since the trajectory return is a constant whose derivative is 0. The equality between Equations (2.20) and (2.21) follows from the chain rule $\nabla_{\phi} \log p_{\phi}(\tau) = \frac{\nabla_{\phi} p_{\phi}(\tau)}{p_{\phi}(\tau)}$. Furthermore, the equality between Equations (2.22) and (2.23) follows from the trajectory probability given in Equation (2.12). The trajectory probability is a product of three terms, but taking the logarithm makes it a sum of three terms where all but one vanishes as the gradient is computed. The resulting policy gradient $\nabla_{\phi} \mathcal{L}(\phi_{old})$ in Equation (2.23) increases the probability of selecting actions associated with a high trajectory return and decreases the probability otherwise. From the policy gradient, multiple reinforcement learning policy gradient methods can be derived, including the REINFORCE algorithm.

2.2.2 REINFORCE

The policy gradient given in Equation (2.23) is unbiased but suffers from high variance because the policy is multiplied by the discounted trajectory return at each timestep. In other words, the policy gradient would decrease the probability of good action-state pairs in the Ludo game example if other action-state pairs in the trajectory lead to being knocked out by the opponent. Fortunately, the action at a given timestep does not affect the rewards received before that timestep, and therefore the discounted trajectory return G_0 can be replaced with the discounted return G_t [16]. The discounted return G_t is defined similarly as the discounted trajectory return, but considers the rewards to-go rather than all rewards obtained on the trajectory. Furthermore, rather than taking the expectation over the entire trajectory, the expectation is commonly taken over timesteps. The modifications give the policy gradient used by the REINFORCE algorithm, which is given by

$$\nabla_{\phi} \mathcal{L}(\phi) = \mathbb{E}_t [G_t \nabla_{\phi} \log \pi_{\phi}(a_t | s_t)]. \quad (2.24)$$

The REINFORCE algorithm involves sampling a trajectory from the policy, computing the discounted return G_t at each timestep t , and updating the parameter vector ϕ by substituting the policy gradient given in Equation (2.24) into the update rule given in Equation (2.17). The policy gradient considers how good each set of action-state pairs in the trajectory is through the return, enabling the updates to adjust the corresponding probabilities accordingly. Once the policy is updated, the trajectory is resampled since the trajectory sampled from the old policy no longer represents the current policy. Such algorithms, where the agent learns from the policy it is currently following, are called on-policy. The contrast is called off-policy algorithms, where the agent learns from a policy different from the one it is currently following. Apart from being an on-policy algorithm, which

is computationally expensive, the REINFORCE algorithm still suffers from high variance, which can lead to unstable training. Fortunately, the variance can be further reduced by subtracting a baseline [14]. To keep the expression unbiased, the baseline can be any function that does not depend on the action, but the most natural choice is to use the state-value function defined in Equation (2.14). In addition, it is common to replace the discounted return G_t with the action-value function defined in Equation (2.15). The modifications gives a policy gradient that involves the difference between the action-value function and the state-value function. The difference gives a measure of how a certain action in a state compares to the average action taken in that state and is known as the advantage function. More precisely, the advantage function is defined as

$$A(s_t, a_t) = q_{\pi_\phi}(s_t, a_t) - v_{\pi_\phi}(s_t). \quad (2.25)$$

The advantage function offers exactly what is needed to reduce the variance of the REINFORCE updates, since it measures how good a given action is with respect to the corresponding state and involves a baseline that does not depend on the action. By replacing the discounted return in Equation (2.24) with the advantage function, a policy gradient that can be used in the baseline version of the REINFORCE algorithm is obtained. More precisely, the resulting policy gradient is given by

$$\nabla_\phi \mathcal{L}(\phi) = \mathbb{E}_t [A(s_t, a_t) \nabla_\phi \log \pi_\phi(a_t | s_t)]. \quad (2.26)$$

The remaining algorithm steps of REINFORCE with baseline are similar to those of the REINFORCE algorithm. The steps involve sampling a trajectory from the policy, computing the advantage function at each timestep, and updating the parameter vector using the update rule given in Equation (2.17). The drawback of the algorithm is that it is difficult to decide the learning rate, as a low value can result in undershooting where unnecessarily small updates are made, which slows down convergence [17]. However, a high learning rate can result in overshooting in which the update misses the optimal peak [17]. In reinforcement learning, it is hard to recover from overshooting as the trajectory sampled from the updated policy may not provide much meaningful information. To overcome this limitation natural policy gradients (NPG) were proposed. NPG gives a direction, how much to move in that direction, and uses a dynamic learning rate to make appropriate updates.

2.2.3 Natural Policy Gradient

NPG is based on the idea that updates can be constrained to avoid overshooting. The simplest form is to constrain the parameter updates using the Euclidean distance, which gives the optimization problem

$$\Delta\phi^* = \arg \max_{\|\Delta\phi\| \leq \epsilon} \mathcal{L}(\phi_{old} + \Delta\phi). \quad (2.27)$$

In Equation (2.27), $\Delta\phi = \phi - \phi_{old}$ is the difference between parameter updates, $\|\Delta\phi\|$ is the Euclidean distance of the update, and ϵ is a hyperparameter that controls how large updates to allow. Low ϵ values result in restrictive updates, giving stable but slow training, while high ϵ values give faster but potentially unstable training. Although the Euclidean distance offers a simple constraint,

it does not necessarily constrain the actual policy due to the metric mismatch between parameters and policies. More precisely, a small parameter update can result in a large policy update. To prevent the policy from changing too much, the Kullback-Leiber (KL) divergence can be used. The KL divergence is defined as

$$D_{KL}(\pi_{\phi_{old}}||\pi_{\phi}) = \sum \pi_{\phi_{old}} \log \frac{\pi_{\phi_{old}}}{\pi_{\phi}}, \quad (2.28)$$

and measures the statistical distance between the old policy $\pi_{\phi_{old}}$ and the new policy π_{ϕ} . The KL divergence can be used to formulate a constrained optimization problem similar to that in Equation (2.27). However, instead Lagrangian relaxation is performed to formulate a penalized optimization problem that is easier to solve [17]. The penalized optimization problem is given by

$$\Delta\phi^* = \arg \max_{\Delta\phi} \mathcal{L}(\phi) - \lambda(D_{KL}(\pi_{\phi_{old}}||\pi_{\phi}) - \epsilon). \quad (2.29)$$

In Equation (2.29), λ is a hyperparameter that controls the impact that the penalty has on the optimization problem. Low λ values mitigate the impact, while high λ values can make the penalty dominate the optimization problem. Although the penalized optimization problem offers improvements over the constrained optimization problem based on Euclidean distance, it is not feasible to compute the KL divergence, since it requires evaluating all state-action pairs, which are typically many. Therefore, the KL divergence is approximated using a Taylor expansion as

$$D_{KL}(\pi_{\phi_{old}}||\pi_{\phi}) \approx D_{KL}(\pi_{\phi_{old}}||\pi_{\phi_{old}}) \quad (2.30)$$

$$+ \Delta\phi^T \nabla_{\phi} D_{KL}(\pi_{\phi_{old}}||\pi_{\phi})|_{\phi=\phi_{old}} \quad (2.31)$$

$$+ \frac{1}{2} \Delta\phi^T \nabla_{\phi}^2 D_{KL}(\pi_{\phi_{old}}||\pi_{\phi})|_{\phi=\phi_{old}} \Delta\phi. \quad (2.32)$$

Each term of the approximated KL divergence can be further rewritten, as shown by [18]. The derivation steps are shown in Appendix A.1. In short, the first two terms are approximated as 0, while the third term is approximated as the Fisher information matrix when approximated locally. More precisely, the third term can be written as

$$\nabla_{\phi}^2 D_{KL}(\pi_{\phi_{old}}||\pi_{\phi})|_{\phi=\phi_{old}} = \mathbb{E}_{\pi_{\phi_{old}}} [\nabla_{\phi} \log \pi_{\phi} \nabla_{\phi} \log \pi_{\phi}^T |_{\phi=\phi_{old}}] \quad (2.33)$$

$$= F(\phi_{old}). \quad (2.34)$$

The local equivalence between the KL divergence and the Fisher information matrix is the key finding, on which the NPG is based. It is an important property because the Fisher information matrix considers how sensitive a probability distribution is to changes in the parameter, which is exactly what is needed for stable and fast training [17]. Given the equivalence, the penalized optimization problem given in Equation (2.29) can be approximated as

$$\Delta\phi^* \approx \arg \max_{\Delta\phi} \mathcal{L}(\phi_{old}) + \nabla_{\phi} \mathcal{L}(\phi)|_{\phi=\phi_{old}} \Delta\phi - \frac{1}{2} \lambda \Delta\phi^T F(\phi_{old}) \Delta\phi + \lambda\epsilon \quad (2.35)$$

$$= \arg \min_{\Delta\phi} -\mathcal{L}(\phi_{old}) - \nabla_{\phi} \mathcal{L}(\phi)|_{\phi=\phi_{old}} \Delta\phi + \frac{1}{2} \lambda \Delta\phi^T F(\phi_{old}) \Delta\phi - \lambda\epsilon. \quad (2.36)$$

The approximate penalized optimization problem is solved by updating the parameter vector ϕ in the direction of the gradient. Setting the gradient of Equation (2.36) with respect to $\Delta\phi$ to 0 and solving for $\Delta\phi$ gives

$$\Delta\phi = \frac{2}{\lambda} F^{-1}(\phi_{old}) \nabla_{\phi} \mathcal{L}(\phi)|_{\phi=\phi_{old}}. \quad (2.37)$$

In Equation (2.37), $F^{-1}(\phi_{old}) \nabla_{\phi} \mathcal{L}(\phi)$ is the NPG. The NPG considers how sensitive the probability distribution is to parameter changes through the inverse of the Fisher information matrix. However, to actually adjust the updates to this sensitivity, a dynamic learning rate is used. The dynamic learning rate is obtained by substituting the solution for $\Delta\phi$ into the KL divergence approximation $\frac{1}{2} \Delta\phi^T F(\phi_{old}) \Delta\phi$. Setting the expression to the chosen hyperparameter ϵ and solving for $\alpha = \frac{1}{\lambda}$ gives a dynamic learning rate α . Given the dynamic learning rate, the update rule used in the NPG algorithm is given by

$$\phi = \phi_{old} + \underbrace{\sqrt{\frac{2\epsilon}{g^T H^{-1} g}}}_{\text{Learning rate } \alpha} H^{-1} g. \quad (2.38)$$

In Equation (2.38), $g = \nabla \mathcal{L}(\phi)|_{\phi=\phi_{old}}$ and H is the Hessian of the KL divergence which is equivalent to the Fisher information matrix. Although the NPG update scheme offers improvements compared to that of REINFORCE, it still suffers from some limitations. The limitations are that it is an on-policy method, the constraint may be violated because of all approximations, and it is computationally expensive to compute the inverse of the Hessian matrix. To overcome these limitations, [19] proposed trust region policy optimization (TRPO).

2.2.4 Trust Region Policy Optimization

The first improvement of TRPO is based on an identity that expresses the expected return of one policy in terms of the advantage function of another policy. The identity is derived in Appendix A.2 as shown by [19]. The final expression is given by

$$\mathcal{L}(\phi) - \mathcal{L}(\phi_{old}) = \mathbb{E}_{s_t \sim p_{\pi_{\phi}}, a_t \sim \pi_{\phi}} \left[A_{\pi_{\phi_{old}}}(s_t, a_t) \right]. \quad (2.39)$$

The drawback with the identity expression is that the states are sampled from the discounted state probability distribution $p_{\pi_{\phi}} = \sum_{t=0}^{\infty} \gamma^t \mathbb{P}(s_t = s | \pi_{\phi})$, while the actions are sampled from the policy π_{ϕ} . In other words, actions and states depend on the policy the agent is currently following, just as in the REINFORCE algorithm. In the REINFORCE algorithm, trajectories are resampled, but that is very inefficient. An alternative approach is to perform multiple updates on the same trajectory, but that could lead to destructively large policy updates. Instead, TRPO uses importance sampling and approximation. Importance sampling is a technique that rewrites an expectation over one probability distribution in terms of the expectation over another probability distribution. In other words, importance sampling allows actions to be sampled from another policy than the one currently followed. The derivation steps for importance sampling are given in Appendix A.3. Furthermore, [19] show that locally the expectation over states sampled from $p_{\pi_{\phi}}$ can be approximated as the expectation over states sampled from $p_{\pi_{\phi_{old}}}$.

Modifying the identity given in Equation (2.39) according to these findings gives the surrogate objective, also known as the conservative policy iteration objective

$$\mathcal{L}(\phi) - \mathcal{L}(\phi_{old}) \approx \mathbb{E}_{s_t \sim p_{\pi_{\phi_{old}}}, a_t \sim \pi_{\phi_{old}}} \left[\frac{\pi_{\phi}(a_t|s_t)}{\pi_{\phi_{old}}(a_t|s_t)} A_{\pi_{\phi_{old}}}(s_t, a_t) \right] \quad (2.40)$$

$$= L^{CPI}(\phi). \quad (2.41)$$

The surrogate objective enables multiple updates using the same trajectory. As a consequence, algorithms that use the surrogate objective are considered off-policy algorithms locally. Before the first update, the importance sampling ratio is equal to 1, while each update will increase or decrease the ratio depending on the sign of the advantage function. More specifically, a positive advantage function indicates an action better than the average, in which case the probability of taking that action is increased, whereas a negative advantage function indicates an action worse than the average, in which case the probability is decreased. Furthermore, from the identity and the surrogate objective, the monotonic improvement theorem can be expressed as

$$\mathcal{L}(\phi) - \mathcal{L}(\phi_{old}) \geq \mathcal{L}^{CPI}(\phi) - CD_{KL}^{\max}(\pi_{\phi_{old}} || \pi_{\phi}), \quad (2.42)$$

where

$$C = \frac{4\epsilon\gamma}{(1-\gamma)^2}. \quad (2.43)$$

In Equation (2.42), $D_{KL}^{\max}(\pi_{\phi_{old}} || \pi_{\phi})$ is the maximum KL divergence over the states, and in Equation (2.43), $\epsilon = \max_{s,a} |A_{\pi_{\phi_{old}}}(s_t, a_t)|$. The monotonic improvement theorem is a minimization-maximization algorithm as improving the right hand side guaranties an improvement in the left hand side. That property is the foundation on which TRPO is based. Rather than optimizing the objective directly, TRPO approximates a lower bound on the improvement using the surrogate objective and defines a trust region in which the approximation is accurate. In practice, the suggested C gives very small updates, and therefore TRPO uses a constrained objective rather than a penalized one. Furthermore, rather than taking the maximum KL divergence as in the monotonic improvement theorem, the expected KL divergence is used in TRPO. The modifications give the TRPO objective

$$\max_{\phi} \mathbb{E}_{s_t \sim p_{\pi_{\phi_{old}}}, a_t \sim \pi_{\phi_{old}}} \left[\frac{\pi_{\phi}(a_t|s_t)}{\pi_{\phi_{old}}(a_t|s_t)} A_{\pi_{\phi_{old}}}(s_t, a_t) \right] \quad (2.44)$$

$$\text{subject to } \mathbb{E}_{s_t \sim p_{\pi_{\phi_{old}}}} [D_{KL}(\pi_{\phi_{old}} || \pi_{\phi})] \leq \delta. \quad (2.45)$$

In Equation (2.45), δ is a hyperparameter that defines the trust region. Small δ values give a small trust region, giving slow but stable training, while high δ values give a large trust region, resulting in fast but potentially unstable training. To optimize the objective, Taylor expansions are used to obtain a first-order approximation of the objective and a second-order approximation of the constraint in the same way as in Equations (2.30)-(2.36). In addition, a dynamic learning rate is derived as in the NPG update rule, derived in Section 2.2.3. The final TRPO update rule is given by

$$\phi = \phi_{old} + \sqrt{\frac{2\delta}{g^T H^{-1} g}} H^{-1} g, \quad (2.46)$$

where $g = \nabla_{\phi} \mathcal{L}^{CPI}(\phi)|_{\phi=\phi_{old}}$ and H is the Hessian of the expected KL divergence. The TRPO update rule is similar to that in NPG, but in practice TRPO modifies it to address some practical considerations. The first limitation of the NPG update rule is that the constraint might be violated because of all approximations made. To overcome this, TRPO uses a line search to verify that the update satisfies the constraint before accepting the update. The second limitation is that it is computationally expensive to compute the inverse of the Hessian matrix. To overcome this, TRPO uses the conjugate gradient algorithm. The conjugate gradient algorithm aims to find x such that $Hx = g$ rather than computing $x = H^{-1}g$. Although TRPO demonstrates impressive performance, it is still a computationally expensive optimization algorithm since it relies on a second-order approximation of the constraint. As an alternative, PPO was proposed by [20].

2.2.5 Proximal Policy Optimization

PPO is a first-order optimization algorithm that achieves the stability of TRPO. Rather than defining a trust region in which the surrogate objective is a valid approximation using the KL divergence, PPO defines a clip function. The clip function limits the importance sampling ratio from moving outside of the interval $[1 - \epsilon, 1 + \epsilon]$ and is given by

$$\text{clip} \left(\frac{\pi_{\phi}^{RL}(a_t|s_t)}{\pi_{\phi_{old}}^{RL}(a_t|s_t)}, 1 - \epsilon, 1 + \epsilon \right). \quad (2.47)$$

In Equation (2.47), ϵ is a hyperparameter that controls how large updates to allow. Small ϵ gives small updates, while larger ϵ allows larger updates. The main objective proposed by [20] is a combination of the surrogate objective and the clip function. The resulting objective is referred to as the clipped objective and is given by

$$\mathcal{L}^{clip}(\phi) = \mathbb{E}_t \left[\min \left(\frac{\pi_{\phi}^{RL}(a_t|s_t)}{\pi_{\phi_{old}}^{RL}(a_t|s_t)} A(s_t, a_t), \text{clip} \left(\frac{\pi_{\phi}^{RL}(a_t|s_t)}{\pi_{\phi_{old}}^{RL}(a_t|s_t)}, 1 - \epsilon, 1 + \epsilon \right) A(s_t, a_t) \right) \right]. \quad (2.48)$$

The clipped objective gives a lower bound on the surrogate objective through the minimization term and the clip function, constraining the updates without requiring a second-order approximation. More specifically, suppose that the advantage function is positive, then the importance sampling ratio will increase since updates to the policy will favor that action. The minimization term allows the importance sampling ratio to increase to $1 + \epsilon$, as visualized by the blue curve in Figure 2.6. Now suppose that the advantage function is negative, then the importance sampling ratio will decrease. The minimization term then allows the importance sampling ratio to decrease to $1 - \epsilon$. In Figure 2.6, the behavior of the clipped objective when the advantage function is negative is visualized by the purple curve. Note that increasing the importance sampling ratio when the advantage function is negative decreases the clipped objective.

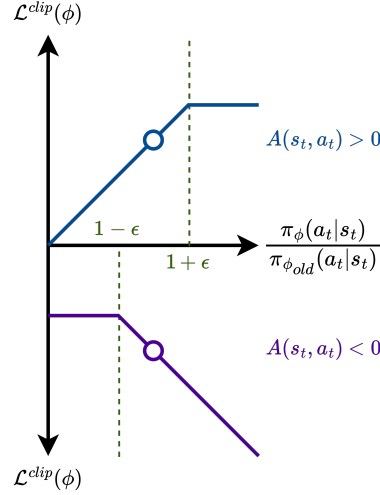


Figure 2.6: Behavior of the clipped objective. The clipped objective is given on the y-axis, while the importance sampling ratio is given on the x-axis. The positive y-axis corresponds to the positive advantage function, visualized by the blue curve. The negative y-axis corresponds to the negative advantage function, visualized by the purple curve. The green dashed lines represent the clipped interval, and the circles represent the starting point when the importance sampling ratio equals 1.

The advantage function is defined in Equation (2.25) as the difference between the action-value function and the state-value function. However, in practice the true functions are unknown, and therefore the advantage function must be estimated. The advantage function can be estimated as the TD error, defined in Equation (2.16). The TD error is an unbiased estimator of the advantage function when the estimated state-value function is equal to the true state-value function. To decrease bias when that is not the case, the advantage function can be estimated as the discounted sum of TD errors. The corresponding estimated advantage functions are given by

$$\hat{A}^{(1)}(s_t, a_t) = \delta_t = -\hat{v}_{\pi_{\phi_{old}}}(s_t) + r_t + \gamma \hat{v}_{\pi_{\phi_{old}}}(s_{t+1}), \quad (2.49)$$

$$\hat{A}^{(2)}(s_t, a_t) = \delta_t + \gamma \delta_{t+1} = -\hat{v}_{\pi_{\phi_{old}}}(s_t) + r_t + \gamma r_{t+1} + \gamma^2 \hat{v}_{\pi_{\phi_{old}}}(s_{t+2}), \quad (2.50)$$

$$\vdots$$

$$\hat{A}^{(k)}(s_t, a_t) = \sum_{i=0}^{k-1} \gamma^i \delta_{t+i}. \quad (2.51)$$

Estimating the advantage function as the discounted sum of TD errors decreases the bias when $k \rightarrow \infty$ since the term $\hat{v}_{\pi_{\phi_{old}}}(s_t)$ does not affect the bias, while the term $\hat{v}_{\pi_{\phi_{old}}}(s_{t+k})$ becomes more heavily discounted. However, increasing the number of terms also increases the variance of the estimator. The number of sums to consider in the estimator is therefore a tradeoff between bias and variance. To balance this tradeoff [21] proposed the generalized advantage estimator (GAE). The GAE is the exponentially weighted average of the estimators given in Equations (2.49) to (2.51).

tions (2.49)-(2.51). The derivation steps are given in Appendix A.4, but in short, GAE is defined as

$$\hat{A}(s_t, a_t) = (1 - \lambda)(\hat{A}^{(1)}(s_t, a_t) + \lambda\hat{A}^{(2)}(s_t, a_t) + \lambda^2\hat{A}^{(3)}(s_t, a_t) + \dots) \quad (2.52)$$

$$= \sum_{i=0}^{\infty} (\gamma\lambda)^i \delta_{t+i}. \quad (2.53)$$

In Equation (2.53), $0 \leq \lambda \leq 1$ is a hyperparameter called the exponential weight discount, which controls the bias and variance tradeoff. Setting $\lambda = 1$ gives the discounted sum of TD errors, which results in a high variance. Setting $\lambda = 0$ gives the TD error, which introduces bias whenever the estimated state-value function is not equal to the true state-value function. Furthermore, the advantage function evaluates the actions chosen by the policy, making PPO an actor-critic method. More precisely, in PPO the policy is the actor that chooses actions and the advantage function is the critic that evaluates them. To take appropriate actions and accurately evaluate them, it is therefore not sufficient to update the policy only. To update the estimated advantage function as well, the estimated state-value function is updated using a squared error loss. In practice, the policy and the state-value function often share parameters, and therefore the objective in PPO combines the clipped objective and the squared error loss given by

$$\mathcal{L}_t^{VF}(\phi) = (\hat{v}_{\pi_{\phi_{old}}}(s_t) - v_{\pi_{\phi_{old}}}(s_t))^2. \quad (2.54)$$

The squared error loss increases with the difference between the estimated state-value function and the true state-value function. Combining the loss with the clipped objective enables the actor and the critic to be updated simultaneously. Furthermore, to encourage exploration so that all possible actions are assigned a non-zero probability of being selected, the clipped objective and the squared error loss are often combined with a third term. The third term is the entropy bonus defined by

$$S[\pi_{\phi}](s_t) = - \sum_a \pi_{\phi}(a|s_t) \log \pi_{\phi}(a|s_t). \quad (2.55)$$

The entropy bonus is 0 for a deterministic policy and positive for a stochastic policy. Combining the three terms gives the PPO objective

$$\mathcal{L}_t^{clip+VF+S}(\phi) = \mathbb{E}_t \left[\mathcal{L}_t^{clip}(\phi) - c_1 \mathcal{L}_t^{VF}(\phi) + c_2 S[\pi_{\phi}](s_t) \right]. \quad (2.56)$$

In the PPO objective, the clipped objective is used to update the actor, the squared error loss is used to update the critic, and the entropy bonus is used to encourage stochastic policies. Furthermore, c_1 and c_2 are hyperparameters that determine the impact that squared error loss and entropy bonus, respectively, have on the objective. Low c_i values give the clipped objective, while high c_i values could make the terms dominate the PPO objective. Finally, the direction in which to update the parameter vector is given by the gradient of the PPO objective. The parameter vector is then updated in that direction using an optimization algorithm such as minibatch stochastic gradient descent or adaptive moment estimation (Adam).

2.3 Reinforcement Learning from Human Feedback

Language models demonstrate impressive capabilities in NLP tasks, but often have the risk of a misaligned objective. More specifically, existing language models are usually trained with a next-token prediction loss, which treats all mismatches between the target and the output the same regardless of the severity of the mismatch [22]. In other words, the objective treats factual errors and synonyms of the target the same despite the differing severity. The objective of language models can become better aligned with the user’s intent by fine-tuning on human feedback, which is the main idea of RLHF. RLHF is commonly divided into three core stages. In the first stage, a pre-trained language model is fine-tuned using supervised learning as visualized by the green box in Figure 2.7. The stage aims to adapt the pre-trained language model to the specific domain or task. In the second stage, a reward model is trained to predict human preference as visualized by the blue box in Figure 2.7. The stage aims to learn a proxy objective that captures the nuanced and complex nature of human preferences. In the third stage, a copy of the supervised fine-tuned language model is fine-tuned on human feedback using the reward model and PPO as visualized by the purple box in Figure 2.7. The stage aims to align the supervised fine-tuned model with the user’s intent. The three stages are visualized in Figure 2.7 and are described in more detail in the following sections.

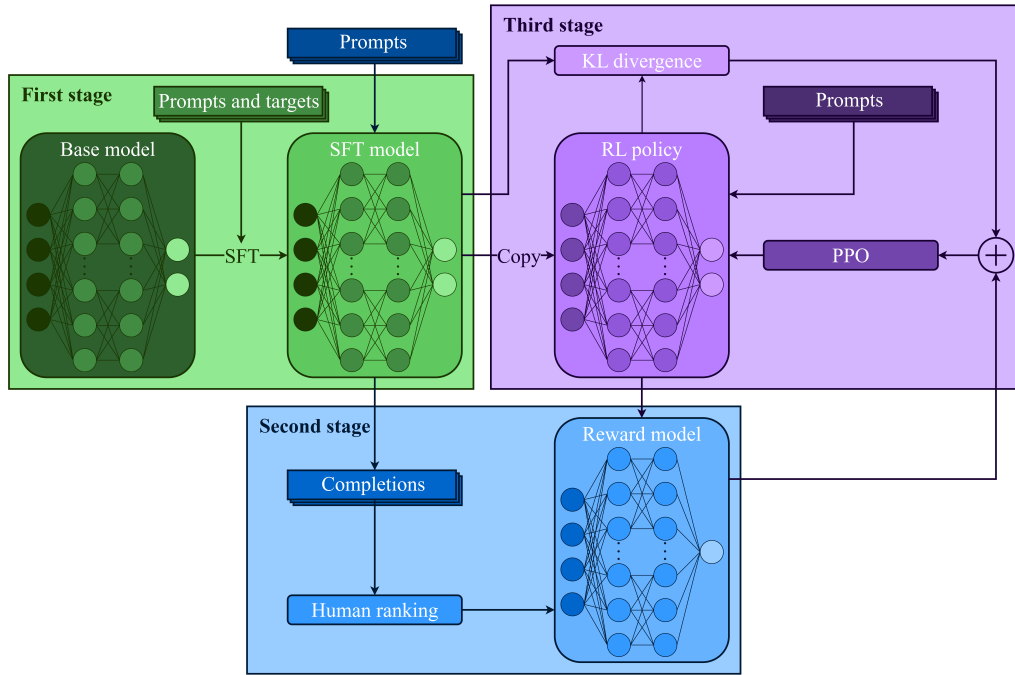


Figure 2.7: The RLHF procedure. In the first stage, a pre-trained language model is fine-tuned using supervised learning, as visualized by the green box. In the second stage, a dataset containing human preferences is constructed to train the reward model, as visualized by the blue box. In the third stage, the reward model acts as a proxy objective to align the supervised fine-tuned model with the user’s intent. The third stage optimizes the proxy objective and a KL divergence penalty term using PPO, as visualized by the purple box. The stages are visualized in more detail in Figures 2.8, 2.9 and 2.10

2.3.1 Supervised Fine-Tuning

In the first stage of the RLHF procedure, a pre-trained language model is fine-tuned using supervised learning. The pre-trained language model is hereafter referred to as the base model and possesses the ability to generate natural language. To better adapt the base model to specific domains or queries, it is fine-tuned using supervised learning. To this end, the base model is trained on a dataset consisting of inputs and targets as visualized in Figure 2.8. The dataset is hereafter referred to as the SFT dataset where the inputs are commonly called prompts, and the targets demonstrate the desired behavior of the model. The model is then trained to minimize the cross-entropy loss given by

$$\mathcal{L}(\omega) = -\mathbb{E}_{x,y \sim \mathcal{D}} [\log p_{\omega}(y_t | y_{<t})]. \quad (2.57)$$

In Equation (2.57), $\mathcal{D}$ is the SFT dataset, ω is the parameter vector that contains the learnable weights, y_t is the target token in timestep t and $y_{<t}$ are the target tokens before timestep t . Conditioning on target tokens ensures that training is not interrupted by a bad prediction early in the sequence and allows the model to predict all tokens in parallel. The loss function is minimized by the parameter vector ω , for which the probability of choosing the target token in the timestep t approaches 1. The direction of the optimal parameter vector ω is given by the gradient of the loss function, which is obtained through backpropagation in deep learning models such as transformers. Given the gradient, the parameter vector is updated using an optimization algorithm such as minibatch stochastic gradient descent or Adam. The training procedure gives the SFT model, which is hereafter denoted π^{SFT} .

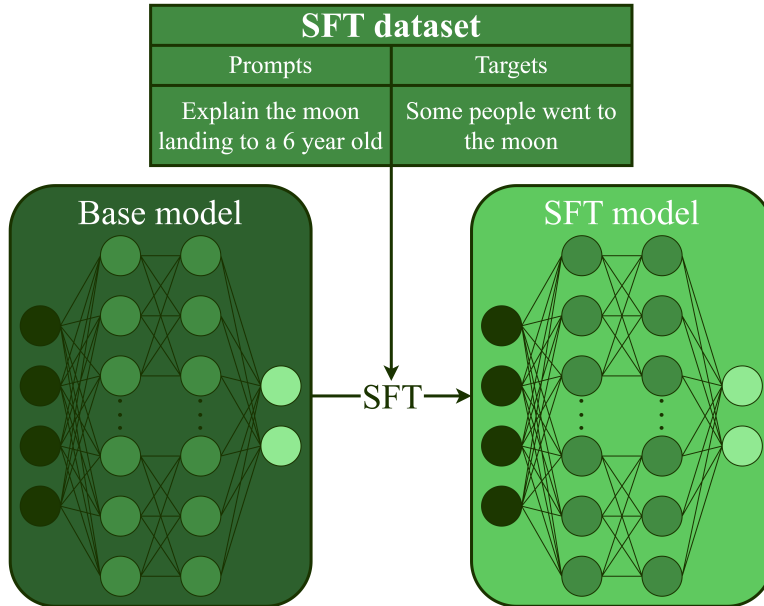


Figure 2.8: Supervised fine-tuning. The first stage of RLHF is to adapt a pre-trained language model to a specific domain or query using supervised fine-tuning. The base model represents the pre-trained language model. The SFT dataset allows for supervised learning through its targets that demonstrate desired behavior. Training the base model using the SFT dataset and cross-entropy loss gives the SFT model. The example prompt and target are provided by [4].

2.3.2 Reward Modeling

The second stage of the RLHF procedure is to train a reward model to predict human preference. The stage begins with the construction of a dataset, hereafter referred to as the RM dataset, which initially contains prompts x . Given a prompt from the RM dataset, the SFT model generates a few outputs y , commonly called completions. To incorporate human feedback into the dataset, all of these completions are ranked in a pairwise manner by human annotators. Inserting the rankings in the dataset gives the final RM dataset, which contains pairs of chosen completions y_1 and rejected completions y_2 for each prompt x as visualized in Figure 2.9. The RM dataset is then used to train the reward model with the goal of obtaining a proxy objective that captures the nuanced and complex nature of human preferences. The architecture of the reward model is similar to the SFT model, but instead of producing a probability distribution over tokens, the reward model produces a scalar reward $r_\theta(y_i, x)$. In transformers, this is typically achieved by replacing the output processing layers with a linear head. The linear head commonly takes the embedding vector of the last token returned by the decoder as input. The embedding vector of the last token is commonly used since it is the only token that has access to all previous tokens because of the autoregressive property of the decoder. As a result, it acts as a natural summary [23]. Given the embedding vector x , the linear head computes the scalar reward

$$r_\theta(y_i, x) = W^T x + b. \quad (2.58)$$

In Equation (2.58), W is a weight vector whose length is equal to the embedding size and b is the bias. The scalar reward is supposed to be high for a good completion and low for a bad completion. To obtain this desired behavior, it is trained on pairs of chosen and rejected completions using a loss function derived from the Bradley-Terry model [15]. The Bradley-Terry model assumes that the preferences for two completions y_1 and y_2 are a noisy reflection of their respective strengths p_1 and p_2 . Given these strengths, the Bradley-Terry model defines the probability that completion y_1 is predicted as preferred by

$$\mathbb{P}(y_1 \succ y_2) = \frac{p_1}{p_1 + p_2}. \quad (2.59)$$

The reason why the Bradley-Terry model is appropriately used to derive the reward model loss function is because it can be rewritten as the sigmoid function. The first part of rewriting the Bradley-Terry model is to parameterize the strengths, which is commonly done by setting $p_1 = e^{r_\theta(y_1, x)}$ and $p_2 = e^{r_\theta(y_2, x)}$. The parameterization allows the scalar rewards to be unbounded without violating the property of positive strengths, which is needed to model probabilities. Given the parameterization, the Bradley-Terry model can be rewritten as

$$\mathbb{P}(y_1 \succ y_2) = \frac{\exp(r_\theta(y_1, x))}{\exp(r_\theta(y_1, x)) + \exp(r_\theta(y_2, x))} \quad (2.60)$$

$$= \frac{\exp(r_\theta(y_1, x))}{\exp(r_\theta(y_1, x))(1 + \frac{\exp(r_\theta(y_2, x))}{\exp(r_\theta(y_1, x))})} \quad (2.61)$$

$$= \frac{1}{1 + \exp(-(r_\theta(y_1, x) - r_\theta(y_2, x)))} \quad (2.62)$$

$$= \sigma(r_\theta(y_1, x) - r_\theta(y_2, x)). \quad (2.63)$$

In Equation (2.63), the sigmoid function depends only on the difference between the scalar rewards, making it an appropriate approximation of human preference [15]. The objective is to maximize the probability that the actual preferred completion y_1 , is also predicted as preferred. Equivalently, the objective can be formulated as minimizing the negative logarithmic likelihood of Equation (2.63). The resulting loss function is given by

$$\mathcal{L}(\theta) = -\mathbb{E}_{x, y_1, y_2 \sim \mathcal{D}} [\log(\sigma(r_\theta(y_1, x) - r_\theta(y_2, x)))], \quad (2.64)$$

where $\mathcal{D}$ is the RM dataset. The optimal parameter vector θ is the one for which the difference $r_\theta(y_1, x) - r_\theta(y_2, x)$ is as big as possible. The direction of the optimal parameter vector is given by the gradient of the loss function, which is obtained through backpropagation in deep learning models, such as transformers. The parameter vector is then updated in that direction using an optimization algorithm such as minibatch stochastic gradient descent or Adam.

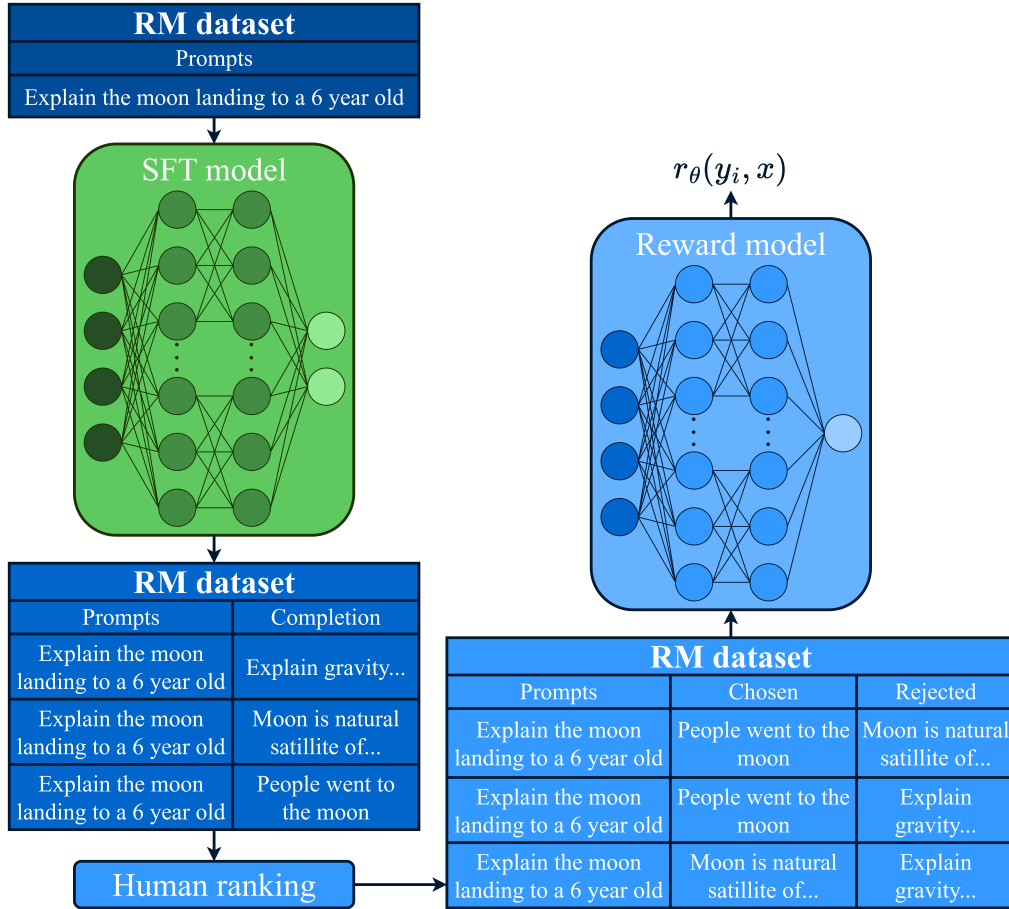


Figure 2.9: Reward modeling. The second stage of RLHF is to train a reward model to predict human preferences with the goal of obtaining a proxy objective. The RM dataset is constructed by feeding the prompts to the SFT model and letting human annotators rank the corresponding completions in a pairwise manner. Given the RM dataset, the reward model is trained to generate scalar rewards $r_\theta(y_i, x)$ that minimize the loss function derived from the Bradley-Terry model. The example prompt and completions are provided by [4].

2.3.3 Reinforcement Learning

In the third and final stage of the RLHF procedure, the reward model acts as a proxy objective that is optimized using PPO, with the goal of aligning a copy of the SFT model with the user’s intent. The trained model in this stage will hereafter be referred to as the RL policy and will be denoted π_ϕ^{RL} . The stage begins with a dataset, which is hereafter referred to as the RL dataset, consisting of prompts. The prompts from the RL dataset are fed to the RL policy, which returns completions. These completions are passed on to the reward model, which returns scalar rewards. Given the scalar rewards, the reward function is defined and maximized using PPO as visualized in Figure 2.10. The reward function, $R(x, y)$, combines the scalar reward $r_\theta(y, x)$ returned by the reward model and a constraint. The constraint is usually chosen as the KL divergence between the per-token probability distributions of the RL policy π_ϕ^{RL} and the SFT model π^{SFT} . The KL divergence gives a statistical distance between the respective probability distributions and is defined in Equation (2.28) as

$$D_{KL}(\pi_\phi^{RL} || \pi^{SFT}) = \sum \pi_\phi^{RL} \log \frac{\pi_\phi^{RL}}{\pi^{SFT}}.$$

In practice, the KL divergence is often approximated because there might not be a closed form expression of the sum or it is too computationally expensive to calculate it. In RLHF, three estimators of the KL divergence are considered, each of which offers different properties in terms of bias and variance [24]. The first estimator is the naive estimator, which is unbiased but suffers from high variance. The naive estimator is given by

$$\hat{D}_{KL}(\pi_\phi^{RL} || \pi^{SFT}) = \log \frac{\pi_\phi^{RL}}{\pi^{SFT}} = -\log r, \quad (2.65)$$

where $r = \frac{\pi^{SFT}}{\pi_\phi^{RL}}$. The naive estimator has a high variance because it is negative for half of the samples, whereas the KL divergence gives a statistical distance which is always positive. To decrease the variance, the estimator could be chosen to give a statistical distance between π^{SFT} and π_ϕ^{RL} . The second estimator gives exactly this and is given by

$$\hat{D}_{KL}(\pi_\phi^{RL} || \pi^{SFT}) = \frac{1}{2} \left(\log \frac{\pi^{SFT}}{\pi_\phi^{RL}} \right)^2 = \frac{1}{2} (\log r)^2. \quad (2.66)$$

The drawback of the second estimator is that it is biased. Another way to reduce the variance is to add an expression to the naive estimator that has expectation zero and is negatively correlated with the estimator. An expression that satisfies the criteria is $r - 1$. Adding the expression to the naive estimator given in Equation (2.65) gives the third estimator, which is given by

$$\hat{D}_{KL}(\pi_\phi^{RL} || \pi^{SFT}) = r - 1 - \log r. \quad (2.67)$$

The third estimator is unbiased and always positive, offering a promising combination of the first and second estimators. The chosen estimator is then used to define the reward function. The reward function combines the scalar reward returned by the reward model and the KL divergence estimator. More precisely, the reward function is given by

$$R(x, y) = r_\theta(y, x) - \beta \hat{D}_{KL}(\pi_\phi^{RL} || \pi^{SFT}). \quad (2.68)$$

The goal is to find the parameter vector ϕ that maximizes the reward function. The optimal parameters are those that maximize the scalar reward while minimizing the KL divergence. The KL divergence acts as a regularization since it increases as the per-token probability distribution of the RL policy moves away from that of the SFT model. Regularization is crucial since without it the RL policy might over-optimize the reward model. Over-optimization is a common problem in reinforcement learning that occurs because an imperfect objective is heavily optimized. In the context of RLHF, this means that the RL policy finds and exploits the weakness of the proxy objective given by the reward model. As a result, reward hacking occurs where the proxy objective diverges from the actual goal, causing high scalar rewards to bad completions [22]. The level of regularization is controlled by the hyperparameter β . Low β eliminates regularization, which can lead to reward hacking, while high β would dominate the reward function, which can lead to zero parameter updates. To achieve the goal of maximizing the reward function, the objective is given by

$$\mathcal{L}(\phi) = \mathbb{E}_{x \sim \mathcal{D}, y \sim \pi_\phi^{RL}} [r_\theta(y, x) - \beta D_{KL}(\pi_\phi^{RL} || \pi^{SFT})], \quad (2.69)$$

where $\mathcal{D}$ is the RL dataset. The objective is then maximized using PPO. As explained in Section 2.2.5, PPO optimizes a lower bound of the surrogate objective using a minimization term and a clip function rather than optimizing the objective directly. More precisely, PPO updates the parameter vector using the clipped objective, defined in Equation (2.48) as

$$\mathcal{L}^{clip}(\phi) = \mathbb{E}_t \left[\min \left(\frac{\pi_\phi^{RL}(a_t | s_t)}{\pi_{\phi_{old}}^{RL}(a_t | s_t)} A(s_t, a_t), \text{clip} \left(\frac{\pi_\phi^{RL}(a_t | s_t)}{\pi_{\phi_{old}}^{RL}(a_t | s_t)}, 1 - \epsilon, 1 + \epsilon \right) A(s_t, a_t) \right) \right].$$

In the context of RLHF, the actor is the RL policy, the actions are the tokens in the vocabulary, the state is the prompt and generated tokens, the return is the reward function, and the policy is the per-token probability distribution. The per-token probability distribution is then updated using the clipped objective by increasing the probability of tokens for which the reward function is expected to be high. Similarly, the clipped objective decreases the probability of tokens for which the reward function is expected to be low. In other words, a negative advantage function decreases the probability, while a positive one increases the probability. To accurately predict the reward function, a critic is trained simultaneously with the actor through a squared error loss term on the state-value function. To increase time efficiency and space efficiency, it is common to let the state-value function and the RL policy share parameters by adding an additional head to the RL policy. Modeling the state-value function in this way motivates combining the clipped objective and the squared error loss term into one objective, as is often done in PPO. In addition, the PPO objective often includes an entropy bonus that encourages stochastic per-token probability distributions. The final PPO objective is defined in Equation (2.56) as

$$\mathcal{L}_t^{clip+VF+S}(\phi) = \mathbb{E}_t \left[\mathcal{L}_t^{clip}(\phi) - c_1 \mathcal{L}_t^{VF}(\phi) + c_2 S[\pi_\phi^{RL}](s_t) \right].$$

The PPO objective seeks to find the parameter vector for which the reward function is maximized while minimizing the reward function estimation error, without ending up in a deterministic per-token probability distribution. To this end, the gradient of the PPO objective is computed to update the parameter vector. The parameter vector is then updated in that direction using an optimization algorithm such as minibatch stochastic gradient descent or Adam.

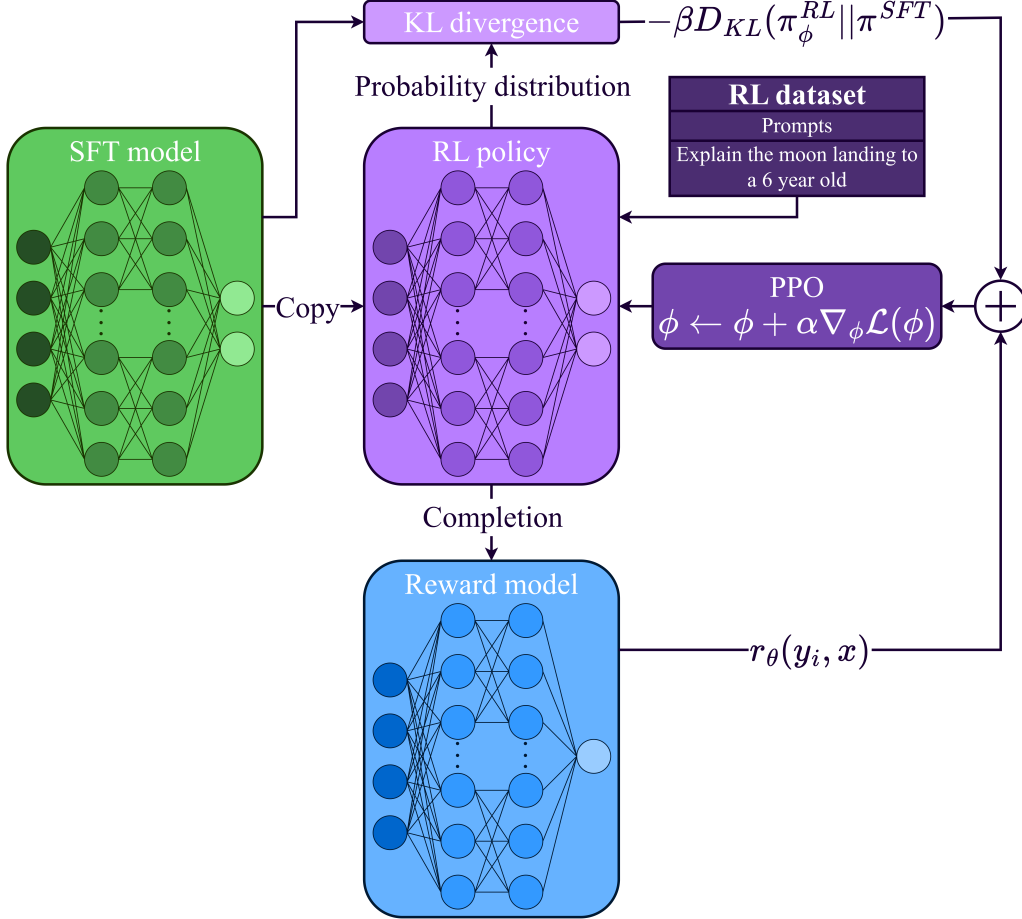


Figure 2.10: Reinforcement learning. The third stage of RLHF is to maximize the scalar reward and a KL divergence constraint using PPO with the goal of aligning a copy of the SFT model with the user’s intent. The stage involves feeding the prompts in the RL dataset, to the RL policy to obtain completions. The completions are then scored by the reward model, while the per-token probability distributions of the SFT model and the RL policy are compared through the KL divergence. Combining the KL divergence and the scalar rewards obtained from the reward model gives the reward function, which is maximized using PPO.

2.4 Direct Preference Optimization

RLHF demonstrate impressive capabilities, both in text summarization tasks and in general tasks [3], [4]. The drawback of RLHF is that it is a complex procedure that involves three separate training stages. In addition, RLHF uses a static reward model as a proxy objective, making the approach sensitive to reward hacking. To overcome these limitations, [5] proposed DPO. DPO is based

on the key finding that the RLHF objective can be rewritten to not depend on a reward model. This allows for fewer training stages and mitigates the reward hacking problem. More specifically, the RLHF objective is given by the reward function, which is a function of the reward model output, where the reward model is trained on a human annotated dataset. In contrast, the DPO objective includes the human annotated dataset directly, giving an approach that does not require a reward model or reinforcement learning.

The DPO approach is divided into two core stages. The first stage is to fine-tune a pre-trained language model using supervised learning just as in the first stage of RLHF. The second stage of DPO is a combination of the reward modeling and reinforcement learning stages of RLHF. In the second stage of DPO, the SFT model is fine-tuned on human preference using the DPO objective, which is obtained by moving backward through the RLHF objectives. The derivation steps are given in Appendix A.5, but in short, the RLHF objective given in Equation (2.69) can be written as

$$\max_{\pi_{\phi}^{RL}} \mathcal{L}(\phi) = \max_{\pi_{\phi}^{RL}} \mathbb{E}_{x \sim \mathcal{D}, y \sim \pi_{\phi}^{RL}} [r_{\theta}(y, x) - \beta D_{KL}(\pi_{\phi}^{RL} || \pi^{SFT})] \quad (2.70)$$

$$= \min_{\pi_{\phi}^{RL}} \mathbb{E}_{x \sim \mathcal{D}, y \sim \pi_{\phi}^{RL}} \left[\log \frac{\pi_{\phi}^{RL}}{\frac{1}{Z(x)} \pi^{SFT} \exp(\frac{1}{\beta} r_{\theta}(y, x))} - \log Z(x) \right]. \quad (2.71)$$

In Equation (2.71), $Z(x) = \sum_y \pi^{SFT} \exp(\frac{1}{\beta} r_{\theta}(y, x))$ is the partition function. The first part of the derivations is to find the optimal RL policy that minimizes Equation (2.71). The partition function does not depend on the RL policy π_{ϕ}^{RL} and, therefore, Equation (2.71) is minimized by minimizing the fraction term. The fraction term is equal to the naive KL estimator given in Equation (2.65), which is minimized when the two distributions are identical. In other words, the optimal RL policy is the one that is equal to the denominator

$$\pi_{\phi}^{RL*} = \frac{1}{Z(x)} \pi^{SFT} \exp\left(\frac{1}{\beta} r_{\theta}(y, x)\right). \quad (2.72)$$

The optimal RL policy, given in Equation (2.72), can be rewritten to give an expression for the scalar reward. The derivation steps are given in Appendix A.6, but in short, the optimal RL policy can be rewritten as

$$r_{\theta}(y, x) = \beta \log \frac{\pi_{\phi}^{RL*}}{\pi^{SFT}} + \beta \log Z(x). \quad (2.73)$$

The scalar reward, given in Equation (2.73), is now expressed in terms of the optimal RL policy π_{ϕ}^{RL*} and the SFT model π^{SFT} , eliminating the need for a reward model. Continuing in the same fashion backward through the RLHF objectives, the scalar reward expression can be used to rewrite the reward model objective used in RLHF. Substituting the scalar reward expression into the reward model objective given in Equation (2.64) gives the DPO loss function

$$\mathcal{L}(\phi) = -\mathbb{E}_{x, y_1, y_2 \sim \mathcal{D}} \left[\log \left(\sigma \left(\beta \log \frac{\pi_{\phi}^{DPO}(y_1|x)}{\pi^{SFT}(y_1|x)} - \beta \log \frac{\pi_{\phi}^{DPO}(y_2|x)}{\pi^{SFT}(y_2|x)} \right) \right) \right]. \quad (2.74)$$

In Equation (2.74), $\mathcal{D}$ is the DPO dataset, which consists of prompts x , preferred completions y_1 and rejected completions y_2 . Furthermore, π_ϕ^{DPO} is the DPO policy, which initially is the SFT model, but then is trained to approach the optimal RL policy. The model is trained by maximizing the difference between implicit rewards. That is, by favoring preferred completions and avoiding rejected completions. To update the parameter vector ϕ in the right direction, the gradient of the loss function is calculated. The parameter vector ϕ is then updated in that direction using an optimization algorithm such as minibatch stochastic gradient descent or Adam. The gradient also provides an intuitive understanding of the loss function. More specifically, the gradient of the DPO loss function given in Equation (2.74) is given by

$$\nabla_\phi \mathcal{L}(\phi) = -\beta \mathbb{E} \left[\sigma(\Delta r_\theta(y_i, x)) \left[\nabla_\phi \log \pi_\phi^{DPO}(y_1|x) - \nabla_\phi \log \pi_\phi^{DPO}(y_2|x) \right] \right], \quad (2.75)$$

where $\Delta r_\theta(y_i, x) = \beta \log \frac{\pi_\phi^{DPO}(y_2|x)}{\pi_\phi^{SFT}(y_2|x)} - \beta \log \frac{\pi_\phi^{DPO}(y_1|x)}{\pi_\phi^{SFT}(y_1|x)}$. The third term in the gradient decreases the likelihood of rejected completions, whereas the second term increases the likelihood of chosen completions. The difference between these is weighted by how incorrectly the model predicts the implicit rewards. More specifically, if the model assigns a higher implicit reward to the rejected completion, the sigmoid function approaches 1 and the update is large. Similarly, if the model assigns a higher implicit reward to the chosen completion, the sigmoid function approaches 0 and the update is small. Furthermore, the weights are scaled by β , whose purpose is the same as before when used to control the impact of the KL divergence penalty.

2.5 Low-Rank Adaption

Although DPO is a simpler preference fine-tuning approach compared to RLHF, both of these approaches require high space complexity and time complexity. Space complexity refers to the memory required to store model parameters and sublayer output during an update. In DPO, each update requires both a frozen reference model and an active policy. In addition, each RLHF update requires a frozen reward model and an active value model. That is, these approaches require approximately two and four times the parameter memory of a supervised fine-tuning update, respectively. Time complexity refers to the computational cost per update. In RLHF, the time complexity is particularly high, as it is an on-policy method that requires sequential token generation in contrast to DPO, which is an off-policy method that relies on a static dataset. Furthermore, multiple updates are required to train a model, and multiple models are required in both the RLHF and DPO approaches. Fortunately, previous work shows that pre-trained language models have a low “intrinsic dimension” when adapted to a specific task [25]. In other words, these models can be trained efficiently despite projecting the weights to a smaller subspace. Inspired by this finding, [25] proposed a parameter-efficient fine-tuning approach called low-rank adaption (LoRA) that projects weight updates to a smaller subspace. The key idea of LoRA is to keep the pre-trained weights frozen and learn a small additive update, rather than updating the full weight matrices. To ensure that the model is initialized as the pre-trained model, the update is initialized to output 0. During training, this

update is iteratively refined, allowing the model to adapt without modifying the pre-trained weights.

Consider the weight matrix W^Q that is used to obtain the query matrix Q in the attention sublayer of the transformer architecture. The number of rows and columns in the matrix is equal to the embedding size, making the number of trainable parameters equal to the embedding size raised to the power of two. In full fine-tuning, the parameters in the weight matrix are updated iteratively using the gradient of the weight matrix. Accumulating these updates gives the forward pass

$$h = W^Q x + \Delta W^Q x, \quad (2.76)$$

where x is the input to the attention sublayer and h is the output, that is, the query matrix Q . The drawback with full fine-tuning is that it requires storing and updating the entire weight matrix, which is computationally expensive as the embedding size grows. Instead, [25] propose to represent the accumulated updates ΔW^Q using two low-rank matrices A and B . Let d denote the embedding size and r denote the rank of the low-rank matrices, then the proposed modified forward pass is given by

$$h = W^Q x + B A x, \quad (2.77)$$

for $B \in \mathbb{R}^{d \times r}$ and $A \in \mathbb{R}^{r \times d}$. The modified forward pass involves updating the low-rank matrices A and B , respectively, while keeping the weight matrix W^Q frozen. The approach reduces the number of trainable parameters for a suitable chosen rank r , which decreases the space complexity and the time complexity of the training procedure. More specifically, fewer parameters need to be stored in memory and fewer gradients need to be computed during backpropagation. The rank r is a hyperparameter that controls compression. Low r compress the weight update to a small representation, in which learning might be limited, while large r barely compress, in which case the advantages of LoRA are mitigated. In addition, increasing r increases the magnitude of the dot product BA , and to prevent this the product is rescaled by $\frac{1}{r}$. However, rescaling the dot product in this way makes the impact of each parameter decrease as the rank increases. To keep the impact more consistent as the rank is tuned, the dot product is rescaled by $\frac{\eta}{r}$ where η is the scaling factor.

LoRA can be applied to all weight matrices in the transformer architecture and to a subset, in which case it can be combined with full fine-tuning. Furthermore, usually A is initialized as random Gaussian and B as zero since that ensures that the training begins with the pre-trained weights. During training, A and B are updated separately using an optimization algorithm such as minibatch stochastic gradient descent or Adam. The resulting low-rank matrices are then used to compute the forward pass as given in Equation (2.77) and visualized in Figure 2.11. Figure 2.11 visualizes both the full fine-tuning update in which the pre-trained weight is added together with the adapted weight, and the parameter-efficient fine-tuning update used in LoRA. In LoRA, the pre-trained weight is added together with the product of the low-rank matrices.

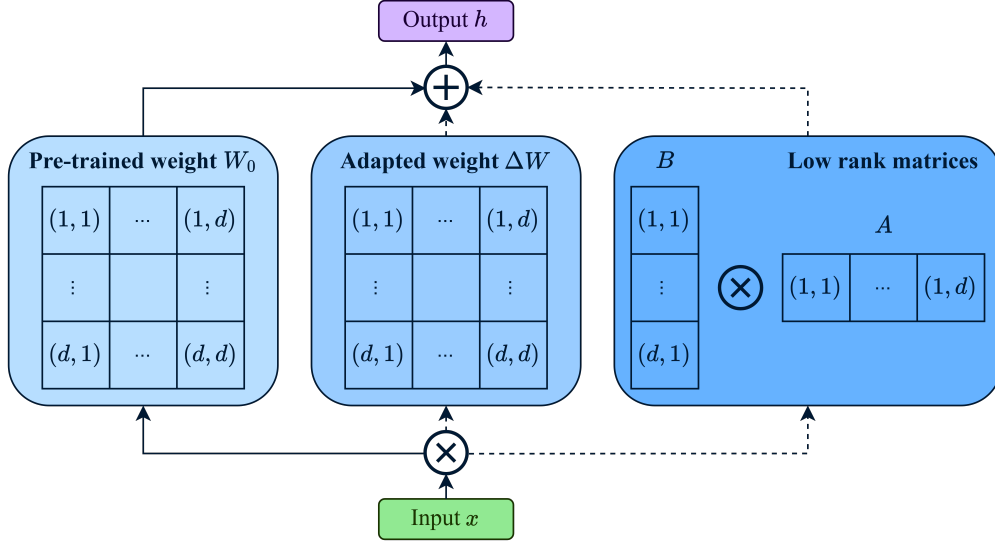


Figure 2.11: Low-rank adaption. The green box visualizes the input, the blue boxes visualize the weight matrices and the purple box visualizes the output. The plain arrow goes to the pre-trained weight, which is common for full fine-tuning and low-rank adaption. The dashed arrows show the distinction between both updates. Combining the plain arrow with the middle dashed arrow gives full fine-tuning. Combining the plain arrow with the right dashed arrow gives low-rank adaption. In the example, $r = 1$ and $W_0 \in \mathbb{R}^{d \times d}$.

2.6 Evaluation Metrics

Despite sophisticated fine-tuning techniques such as RLHF and DPO, the risks of factual errors and incomplete completions are still present. Therefore, it is important to validate the texts generated by the language models. Validating texts is challenging, as there is not necessarily a right or wrong answer. Instead, a good validation approach for text generations should be robust to different orderings of words and be able to capture semantic meaning. Most validation approaches are built on recall and precision. Intuitively, recall measures how much of a reference is preserved by the generation, while precision measures how relevant the generation is. In other words, the former gives a measure of quantity, while the latter gives a measure of quality. The evaluation metrics proposed for text summarization include ROUGE and BERTScore.

2.6.1 ROUGE

ROUGE is a set of metrics designed to evaluate generated texts. The metrics compare a reference text with the completion in terms of different aspects decided by the ROUGE variant [26]. The ROUGE variants include ROUGE-N and ROUGE-L. ROUGE-N measures the number of N consecutive words that overlap between the reference text and the completion. Consider, for instance, the example reference and completion

R : The cat is on the mat,
C : The cat and the dog,

provided by [26]. In the example, the ROUGE-1 overlaps are “the”, “cat”, and “the”, while the ROUGE-2 overlap is “the cat”. These overlaps are usually referred to as matching N-grams so, more formally, the example contains 3 matching 1-grams and 1 matching 2-grams. In both the recall and precision, these matching N-grams are compared to the total number of N-grams. For instance, the reference contains a total of 5 2-grams while the completion contains a total of 4 2-grams. Given these N-grams, the ROUGE-N recall is defined as

$$R_{\text{ROUGE-N}} = \frac{\text{Number of matching N-grams}}{\text{Number of N-grams in the reference}}. \quad (2.78)$$

The ROUGE-N recall calculates the fraction of N-grams in the reference text that the completion preserved. The drawback with the recall is that it can be fooled into giving a high score to a completion that contains every single combination of words in the vocabulary. Similarly, the ROUGE-N precision is defined as

$$P_{\text{ROUGE-N}} = \frac{\text{Number of matching N-grams}}{\text{Number of N-grams in the completion}}. \quad (2.79)$$

The ROUGE-N precision calculates the fraction of N-grams in the completion that are present in the reference text. The drawback with the precision is that it can be fooled into giving a high score to a completion that contains a single matching N-gram. Combining recall and precision gives the ROUGE-N F1 score, which is defined as

$$F1_{\text{ROUGE-N}} = \frac{2 \cdot P \cdot R}{P + R}. \quad (2.80)$$

The ROUGE-N F1 score evaluates both recall and precision, making it a more robust score that is not easily fooled in contrast to evaluating recall and precision separately. In addition, the F1 score penalizes the imbalance between the recall and the precision, making it a more robust score than computing the average between these. More specifically, if the completion contains a single word that overlaps with the reference, then the ROUGE-1 precision would be 1 while the ROUGE-1 recall would be near 0. As a result, the average between these metrics would be around 0.5, which is misleadingly high, while the ROUGE-1 F1-score would be nearer 0, giving a more accurate representation of completion quality.

The corresponding recall, precision, and F1 score of ROUGE-L is defined similarly, but they consider the number of words in the longest common sequence between the reference and the completion rather than N-grams. The sequence does not need to be consecutive, but must be in order. In the example, the longest common sequence is “the cat the”, giving a ROUGE-L recall equal to $\frac{3}{6} = 0.5$ and a ROUGE-L precision equal to $\frac{3}{5} = 0.6$. The remaining ROUGE variants are variations of ROUGE-N and ROUGE-L like ROUGE-W, which is a weighted ROUGE-L that favors consecutive longest common sequences. However, ROUGE-N and ROUGE-L are the most common variants.

2.6.2 BERTScore

The ROUGE metrics are simple and robust but have limited capabilities because they only care about perfect overlaps. In other words, the ROUGE metrics treat a synonym of the reference word the same as a completely different word. For instance, the completion “The cat is on the carpet” would obtain the same scores as

the completion “The cat is on the run” when compared to the reference example in the previous section. To overcome this drawback, [27] proposed an alternative evaluation metric called BERTScore. Computing the BERTScore involves three steps, each of which uses a reference sequence of tokens, and a completion. The first part of computing the BERTScore is to represent the meaning of each token in the reference sequence and the completion sequence with a contextual embedding vector as visualized by the green box in Figure 2.12. The contextual embedding vector is similar to the embedding vector used in the transformer architecture, but it is able to capture the contextual representation of a token. Recall that the embedding vector used in the transformer architecture is obtained during the input processing, before the sequence is known. Therefore, the same tokens have the same embedding vector that is independent of the context. In contrast, the contextual embedding vector is obtained after the attention mechanism, when the sequence is known. Therefore, the contextual embedding vector can capture the context and represent the same token with different vectors. The contextual embedding vector is computed with models such as the BERT model and enables BERTScore to capture semantics. The similarities between the reference tokens and the generated tokens are then computed using the cosine similarity as visualized by the blue box in Figure 2.12. Cosine similarity allows for a soft measure of similarity, which treats synonyms more appropriately compared to the hard measure used by ROUGE. Given the contextual embedding vector x_i for a reference token and the contextual embedding vector $\hat{x}_j$ for a generated token, the cosine similarity is given by

$$\text{Similarity}(x_i, \hat{x}_j) = \frac{x_i^T \hat{x}_j}{\|x_i\| \|\hat{x}_j\|}. \quad (2.81)$$

In practice, cosine similarity is given by the dot product $x_i^T \hat{x}_j$ since BERTScore normalizes the vectors x_i and $\hat{x}_j$. Lastly, the recall, precision, and F1 score are calculated. Recall is calculated by matching each token in the reference with the most similar token in the completion, and is defined as

$$R_{\text{BERT}} = \frac{1}{|x|} \sum_{x_i \in x} \max_{\hat{x}_j \in \hat{x}} x_i^T \hat{x}_j. \quad (2.82)$$

The drawback with the recall is that it can be fooled into giving a high score to a sequence consisting of every token in the vocabulary. The precision is calculated by matching each token in the completion with the most similar token in the reference, and is defined as

$$P_{\text{BERT}} = \frac{1}{|\hat{x}|} \sum_{\hat{x}_j \in \hat{x}} \max_{x_i \in x} x_i^T \hat{x}_j. \quad (2.83)$$

The drawback of precision is that it can be fooled into giving a high score to a sequence consisting of a few but common tokens. The F1 score combines recall and precision and is given by

$$F1_{\text{BERT}} = \frac{2 \cdot P \cdot R}{P + R}. \quad (2.84)$$

The F1 score evaluates both recall and precision, making it a more robust score compared to evaluating recall and precision separately. In addition, the formula penalizes the imbalance between the recall and the precision, making it a more

robust score compared to taking the average between these. Furthermore, recall, precision, and the F1 score can be combined with importance weighting, as common tokens give little information about completion quality compared to uncommon tokens. Importance weighting is obtained by computing the inverse document frequency (IDF) scores. Given M references $\{x^{(i)}\}_{i=1}^M$, the IDF of the token w is calculated as

$$\text{IDF}(w) = -\log \frac{1}{M} \sum_{i=1}^M \mathbb{I}(w \in x^{(i)}), \quad (2.85)$$

where $\mathbb{I}(\cdot)$ is an indicator function. The IDF score is low for a common token that is present in all M references, and high for an uncommon token. Given the IDF scores for all tokens, importance weighting is incorporated in the recall as

$$R_{\text{BERT}} = \frac{1}{\sum_{x_i \in x} \text{IDF}(x_i)} \sum_{x_i \in x} \text{IDF}(x_i) \max_{\hat{x}_j \in \hat{x}} x_i^T \hat{x}_j. \quad (2.86)$$

The importance weighted recall ensure that a completion does not receive a high recall simply because it contains many common words, making it a more robust score. The importance weighted precision is defined similarly, and combining these gives the importance weighted F1 score.

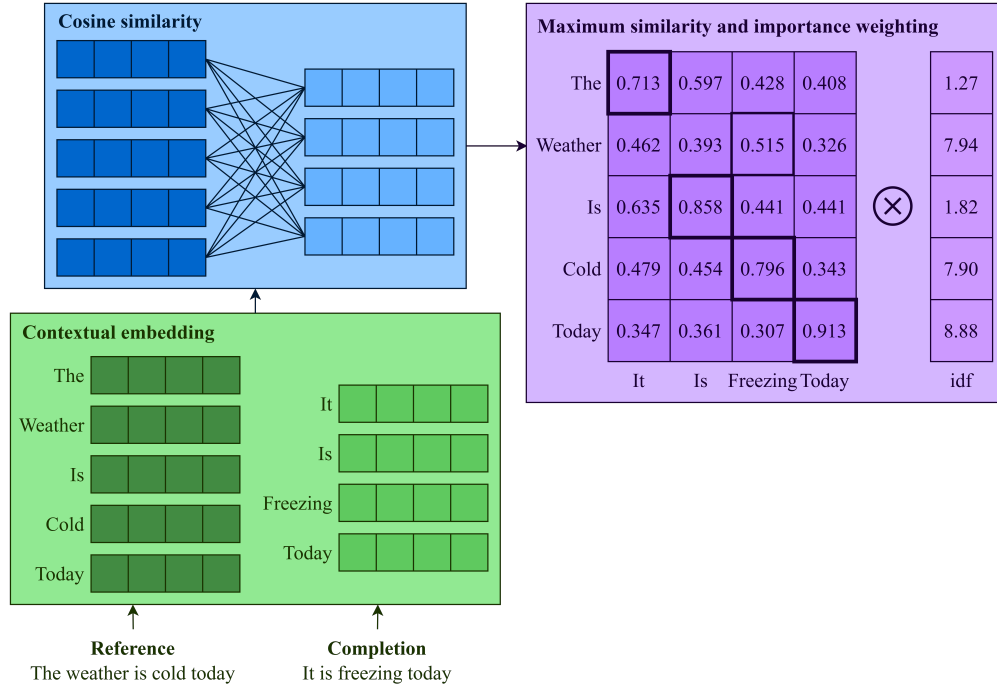


Figure 2.12: Calculation of BERTScore. The procedure starts with a reference sequence of tokens and a completion sequence of tokens. Each token is represented by a contextual embedding vector, computed with a transformer-based model like BERT. The pairwise similarities between tokens in the reference and completion are calculated using cosine similarity. The recall, precision and F1 score are calculated by matching the most similar tokens and multiplying this by inverse document frequency scores.

The BERTScore metrics given in Equations (2.82) - (2.84) have the same numerical range as the cosine similarity, that is, $[-1, 1]$, in theory. However, in practice, the range is more limited because embedding vectors rarely have a negative cosine similarity, making the metrics difficult to interpret. To overcome this limitation, the BERTScore metrics can be rescaled. The rescaled BERTScore recall is defined as

$$\hat{R}_{\text{BERT}} = \frac{R_{\text{BERT}} - b}{1 - b}, \quad (2.87)$$

where R_{BERT} is the BERTScore recall defined in Equation (2.82) and b is the recall baseline. The baseline is computed by [27] by pairing multiple random sentence pairs and averaging the obtained BERTScore recall for these. The rescaled precision is obtained similarly, and combining these gives the rescaled F1 score. Rescaling the metrics in this way make the values typically lie in the range $[0, 1]$. This range is more interpretable as a value of 0 corresponds to a completion comparable to a random sentence, while 1 corresponds to a perfect completion. Although the values typically lie in the range $[0, 1]$, it is possible to obtain a negative value if the completion is worse than a random sentence.

Chapter 3

Methodology

In this chapter, the specific implementation choices made for this work are provided. The chapter begins with the dataset in Section 3.1. The chapter continues with the pre-trained language models in Section 3.2. Lastly, the experimental setup with respect to the optimization algorithm and the hyperparameter settings is given in Section 3.3.

3.1 Dataset

In Section 2.3, the RM dataset construction is explained as feeding prompts to the SFT model, which generates completions that are ranked by human annotators. In practice, the approach is not feasible for a degree project, as it is both time consuming and would require a team of human labelers to rank the completions. Therefore, a dataset consisting of human feedback is used to train the models in this work. The dataset originates from the *TL;DR* dataset, short for “too long; did not read”. The dataset is explained in more detail by [28]. In short, it consists of Reddit posts and the corresponding summaries written by the author of the post. Reddit is, among other things, a discussion forum where members can upload posts, which are segregated into subreddits such as *Pets*, *Cooking*, and *Jobs*. Human feedback was then incorporated into the *TL;DR* dataset by [3], who fine-tuned their transformer based GPT-3 models on the *TL;DR* dataset using supervised learning, and then used the resulting models to generate several summaries per post. The generated summaries were then compared in terms of essence, clarity, accuracy, purpose, conciseness, and style by human labelers. The process is explained in more detail by [3], but the resulting dataset is called *summarize from feedback* and is used to train the models in this work.

The *summarize from feedback* dataset consists of posts, chosen summaries, and rejected summaries [29]. In addition, the dataset was obtained in a preprocessed format, where it had already been divided into two subsets. One intended to be used for training and the other intended to be used for evaluation. The properties of these subsets are summarized in Table 3.1.

	Training subset	Evaluation subset
Number of samples	92858	86086
Number of posts	14770	6233
<i>Samples per post</i>		
Average	6.287	13.447
Standard deviation	9.076	13.661
<i>Summaries per post</i>		
Average	4.108	7.528
Standard deviation	0.630	3.745
<i>Words per post</i>		
Average	243.562	245.167
Standard deviation	78.057	76.864
<i>Words per preferred summary</i>		
Average	27.580	29.211
Standard deviation	10.548	7.243
<i>Words per rejected summary</i>		
Average	26.174	27.960
Standard deviation	10.121	7.544

Table 3.1: Properties of dataset subsets. The number of samples is the number of triple sets consisting of Reddit posts, rejected summaries, and preferred summaries. The number of posts is the number of unique Reddit posts.

Recall that several summaries are generated for each post and then compared in a pairwise manner, which is why the number of samples is different from the number of posts in Table 3.1. More precisely, the number of posts is the number of unique Reddit posts, while the number of samples is the number of triple sets consisting of posts, rejected summaries, and preferred summaries. The difference between these is visualized in Figure 2.9, where three completions are generated for one prompt, resulting in three samples. Table 3.1 shows that the training subset consists of more samples and posts than the evaluation subset. However, the difference is smaller between the number of samples, as the evaluation subset also includes more summaries per post. The remaining properties given in Table 3.1 show that the evaluation subset has somewhat longer posts and summaries than the training subset. Furthermore, the difference between lengths in the preferred and rejected summaries in the training subset is small, which is advantageous, as the contrast would potentially train the models to generate long summaries rather than good ones.

In this work, the evaluation subset is used for two purposes. First, for monitoring the model performance on unseen data during training. Secondly, for evaluating the model performance on unseen data after training. Therefore, the evaluation subset of the *summarize from feedback* dataset is divided into two equally large datasets as visualized by the blue and purple box in Figure 3.1. The blue box represents the validation dataset used during training, and the purple box represents the test dataset used after training. Furthermore, the RLHF procedure is divided into three stages, which require three unique training datasets. The

different training datasets are obtained by dividing the training subset of the *summarize from feedback* dataset into three datasets of approximately the same size, as visualized by the green boxes in Figure 3.1.

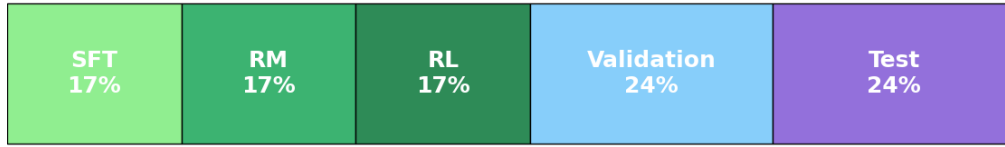


Figure 3.1: Proportion of samples in the different datasets. The *summarize from feedback* training subset is divided into three datasets, which are colored green. The first dataset is used to supervise fine-tune (SFT) a pre-trained language model, the second dataset is used to train the reward model (RM), and the third dataset is used to fine-tune the SFT model on human preference using reinforcement learning (RL). The *summarize from feedback* evaluation subset is divided into two datasets, which are colored blue and purple. The first dataset is the validation dataset used to monitor the model performance on unseen data during training. The second dataset is the test dataset used to evaluate the model performance on unseen data after training.

Similarly as RLHF, DPO is divided into two stages, which requires two unique training datasets. The first stage is to fine-tune a pre-trained language model using supervised learning, and for consistency reasons, the same SFT model is used in both RLHF and DPO. The second stage is to fine-tune the SFT model using human preference, and for this stage, the RM dataset is used for reasons that become apparent in the following sections. In the following two sections, the different properties of the different datasets used for RLHF and DPO are explained.

3.1.1 Reinforcement Learning from Human Feedback Datasets

The first training dataset, the SFT dataset, is used in the first stage of RLHF. The first stage is to fine-tune a pre-trained language model using supervised learning. To this end, the training dataset should consist of posts and preferred summaries that demonstrate the desired behavior, as visualized in Figure 2.8. In other words, the rejected summaries in the SFT dataset are not needed in the first stage, and therefore, they are removed. Removing rejected summaries results in duplicate samples, as visualized in Figure 2.9. The figure shows three samples of pairwise comparisons between three completions. The best completion is preferred in two samples, whereas the second best is preferred in a single sample. Due to this pairwise comparison, the figure contains only two unique pairs of prompts and preferred completions. Removing duplicates gives 15018 training samples. The subreddits and their corresponding distributions in the modified SFT training dataset are visualized in Figure 3.2a. Furthermore, during training, performance on unseen data is monitored. To this end, the validation dataset is used, which must satisfy the same properties as the SFT training dataset. Removing rejected summaries and duplicates gives 17213 validation samples. The subreddits and their corresponding distributions in the modified validation dataset are visualized in Figure 3.2b.

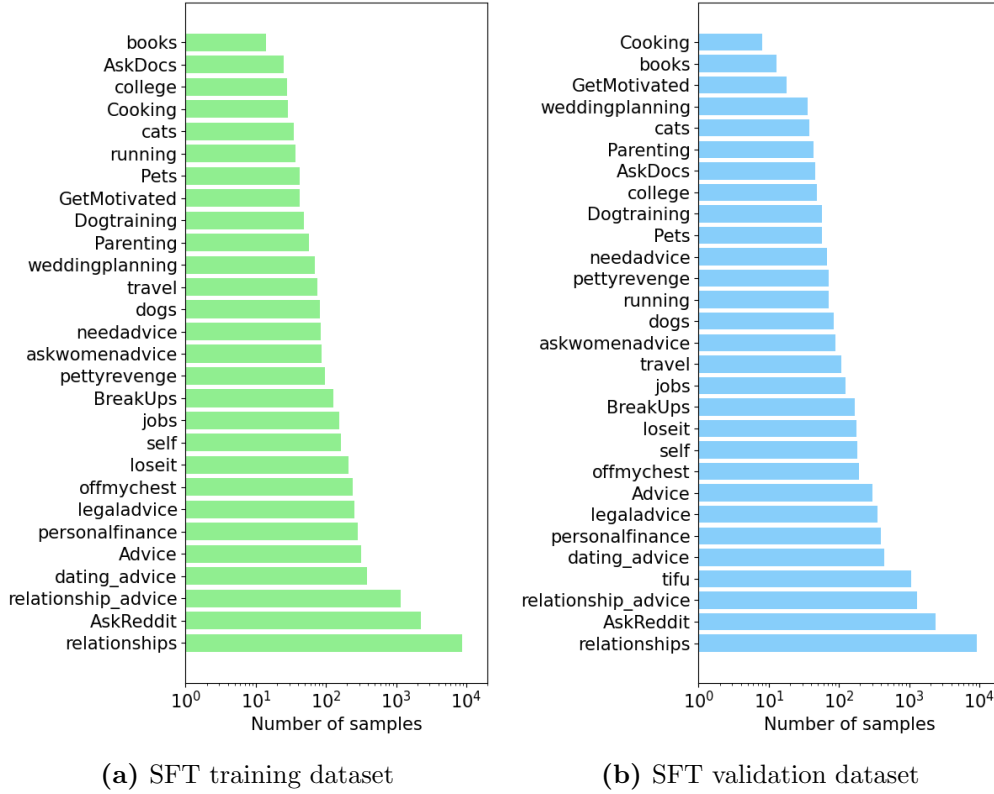


Figure 3.2: Subreddits and their corresponding distribution. The SFT datasets involve a training dataset and a validation dataset, both of which contain Reddit posts from several different subreddits. The subreddits and their distribution in the SFT training dataset are visualized on a logarithmic scale in Figure 3.2a. The subreddits and their distribution in the SFT validation dataset are visualized on a logarithmic scale in Figure 3.2b.

Figure 3.2 shows that the SFT training dataset consists of a wide range of subreddits, allowing the pre-trained language model to adapt to several different domains such as *Legal advice* and *Personal finance*. Furthermore, the distribution between the train dataset and the validation dataset are similar, which allows for consistency between training and validation.

The second training dataset, the RM dataset, is used in the second stage of the RLHF procedure. The second stage is to train a reward model to predict human preference. To this end, the training dataset requires posts, preferred summaries, and rejected summaries, as visualized in Figure 2.9. In other words, no further modifications to the RM dataset are required, giving 30952 training samples. Furthermore, to monitor the model performance on unseen data during training, the validation dataset is used. The validation dataset does not require further modifications, which results in 43043 validation samples. The subreddits and their corresponding distributions in the RM datasets are visualized in Appendix B.1 and show results similar to Figure 3.2.

The final third training dataset, the RL dataset, is used in the third stage of the RLHF procedure. The third stage is to fine-tune a copy of the SFT model on human preference using the reward model. To this end, the training dataset

should consist of posts as visualized in Figure 2.10. In other words, the preferred and rejected summaries in the RL dataset are not needed in the third stage, and therefore, they are removed. Removing the summaries and duplicate samples that appear gives 4640 training samples. Furthermore, in contrast to previous stages that are considered supervised learning methods, the third stage does not contain targets in the dataset. Because of this, the performance of the model is not properly monitored using the validation dataset during training.

The number of samples in the different datasets is summarized in Table 3.2. Furthermore, after training the SFT model and the RL policy, the ROUGE and BERTScore metrics are used for evaluation. To this end, posts and corresponding preferred summaries that demonstrate desired behavior are required from the test dataset. Keeping a single reference summary for each post gives 4793 test samples.

3.1.2 Direct Preference Optimization Datasets

The DPO procedure is divided into two stages, where the first stage is shared with RLHF and the second stage is to fine-tune the SFT model using human preference. The second stage is in many ways similar to the RLHF reward modeling stage, in part because it requires a dataset consisting of posts, preferred summaries, and rejected summaries. Therefore, to keep training as consistent as possible, the RM dataset is used for the second stage of the DPO approach. The alternative would be to use the RL dataset for the second stage of DPO, as that dataset is used to train the RL policy that DPO is compared to. However, the RL dataset consists only of 4640 samples. Modifying the RL dataset back to its original format, to match the desired DPO format, would therefore introduce 26313 samples that RLHF has not seen, leading to inconsistency. Clearly, inconsistency is introduced by using the RM dataset as well since DPO never sees the RL dataset. However, a discrepancy of the 4640 samples between RLHF and DPO introduces less inconsistency compared to a discrepancy of the 26313 samples. In addition, DPO involves one stage less than RLHF, and therefore it is more reasonable to show slightly fewer samples to DPO.

In contrast to the final stage of RLHF, the final stage of DPO is considered a supervised learning method. Therefore, the validation part of the dataset is used to monitor model performance during training. Furthermore, after training, the DPO policy is evaluated using ROUGE and BERTScore. To this end, posts and corresponding preferred summaries are required from the test dataset. Keeping a single reference summary for each post gives 4793 test samples.

3.1.3 AI Feedback

The *summarize from feedback* dataset offers advantages as it enables training on human preference without human labelers. The disadvantage of using a dataset that includes human preferences is that few exist because it is resource-expensive to create such a dataset. It can therefore be challenging to find an appropriate dataset if the task is to fine-tune the models on domain-specific content. In addition, the completions in the dataset are generated using a language model which may not perform at the same level as new models, potentially resulting in poor target completions. To overcome these limitations, RLHF and DPO can be trained on AI feedback rather than human feedback, in which case RLHF is called

RLAIF. The AI feedback is obtained by letting a pre-trained but not fine-tuned, so called off-the-shelf, language model rank the completions. In other words, the difference between human feedback and AI feedback lies in who ranks the completions. According to [6], RLAIF achieves a performance comparable to, or even better than, RLHF in several NLP tasks, making it an interesting alternative approach. The impact of AI feedback is further investigated in this work by comparing RLHF and DPO, trained on both the *summarize from feedback* dataset and a dataset that includes AI feedback.

Using an off-the-shelf language model to rank the completions allows the summaries to be generated by the SFT model without the need for human labelers. This aligns more with the standard RLHF procedure and is advantageous as it could produce target summaries of better quality than those in the *summarize from feedback* dataset. However, in this work, the main purpose is to compare approaches, and therefore consistency is prioritized. Because of this, the off-the-shelf language model is used to rank the summaries in the *summarize from feedback* dataset to ensure that all models have seen approximately the same summaries. Doing this gives a dataset where the preferences align with those in the *summarize from feedback* dataset in approximately 69.92% of the cases. Apart from the preferences, the dataset that incorporates AI feedback is very similar to that with human feedback. They are both of the same format and have approximately the same number of samples in each dataset, as shown in Table 3.2.

	SFT	SFTAI	RM	RMAI	RL	RLAI
Train dataset	15018	15062	30952	30952	4640	4640
Validation dataset	17213	17518	43043	43043	-	-
Test dataset	4793	4793	—	—	4793	4793

Table 3.2: Number of samples in the different datasets. The SFT dataset is used for supervised fine-tuning in the first stage of both RLHF and DPO. The RM dataset is used for reward modeling in RLHF and preference fine-tuning in DPO. The RL dataset is used for the final stage of RLHF. The corresponding AI datasets are used for the same purposes, but are obtained by letting an off-the-shelf language model rank summaries rather than human labelers. The train dataset is used for training, the validation dataset is used to evaluate performance during training, and the test dataset is used to evaluate performance after training using ROUGE and BERTScore. The test dataset is not used for the reward modeling stage of RLHF, while the validation dataset is not used for the reinforcement learning stage of RLHF.

To simplify notation and improve clarity, the SFT dataset obtained by allowing an off-the-shelf language model rank summaries is hereafter referred to as the SFTAI dataset. In addition, the SFT model trained on the SFTAI dataset is hereafter referred to as the SFTAI model. Similarly, the AI versions of the remaining datasets are hereafter referred to as the RMAI and RLAI datasets. These datasets are used to train the AI versions of the corresponding models, which are hereafter referred to as the RMAI, the RLAI policy, and the DPOAI policy.

3.2 Pre-Trained Language Models

RLHF and DPO are both approaches that are used to further train a pre-trained language model. Furthermore, to generate the AI feedback, a pre-trained language model is needed to compare the generated summaries in the *summarize from feedback* dataset. To this end, two pre-trained language models are required, both of which are chosen from the Qwen3 model family, since the family includes multiple different models that achieve state-of-the-art results on a wide range of tasks [30].

Qwen3 is a collection of transformer-based models that have been through sophisticated pre-training and post-training techniques. The architecture of these models includes grouped query attention, SwiGLU activation, rotary positional embeddings, and RMSNorm. In short, grouped query attention uses fewer key and value heads than query heads, which increases memory efficiency. SwiGLU activation combines swish and gated linear units, allowing the model to capture more complex relationships compared to ReLU activation. The rotary positional embedding encodes positional information by rotating vectors rather than adding embeddings, which is more efficient. Lastly, RMSNorm is a more efficient and stable alternative to layer normalization [31]. In addition, the Qwen3 models use byte level BPE, a variant of the tokenization approach described in Section 2.1.1.

The pre-training process of the Qwen3 models is explained in more detail by [30], but in short, it consists of three stages. Each of these stages focus on improving a certain aspect of the models by training them on a high quality dataset. In the first stage, the models are trained on a large dataset to obtain general knowledge. In the second stage, the models are trained on a domain-specific dataset to increase their reasoning abilities in areas such as STEM (science, technology, engineering and mathematics) and coding. In the third and final stage, the models are trained on long-context data to enhance their ability to capture long contexts.

The post-training process is also explained in more detail by [30], but in short, it includes either four stages or strong-to-weak distillation depending on the model. More specifically, larger so-called teacher models are post-trained through four techniques. The resulting teacher models are then used to transfer knowledge to the smaller so-called student models through strong-to-weak distillation. The post-training process is visualized in Figure 3.3.

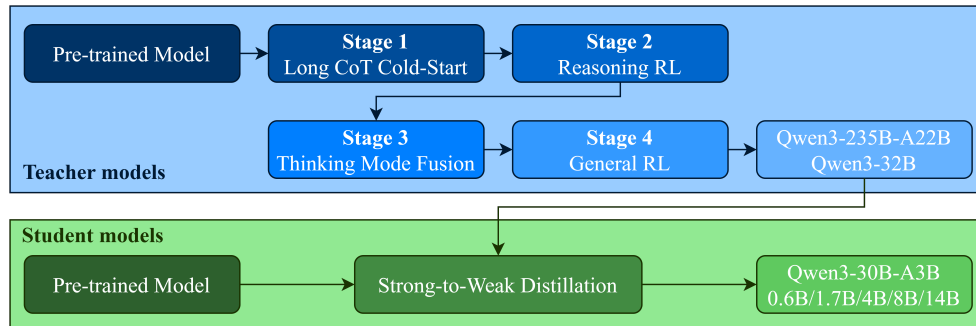


Figure 3.3: The post-training process of Qwen3 models. Teacher models are post-trained using a four stage process. The resulting teacher models are used to transfer knowledge to the student models through strong-to-weak distillation.

The first stage of post-training the teacher models is long chain-of-thought (CoT) cold-start fine-tuning. The objective of the stage is to create reasoning capabilities in the models by training them on complex problems that require multi-step reasoning. The second stage is reasoning RL, where the grouped relative policy optimization (GRPO) is used to update the models parameters. GRPO is an alternative approach to PPO that enables fine-tuning on human feedback. The first two stages focus on enhancing the models reasoning abilities to create a thinking mode, while the remaining two stages focus on the non-thinking mode to generate rapid responses. To this end, the third stage is thinking mode fusion, where supervised fine-tuning is performed to integrate non-thinking capabilities in the models. The final and fourth stage is general RL, which aims to further improve the capabilities of the models through sophisticated reward systems and reinforcement learning. The post-training process of the student models involves strong-to-weak distillation. The process first allows the student models to learn basic reasoning capabilities and switch between non-thinking and thinking mode. Secondly, they are fine-tuned by aligning their per-token probability distribution with that of the teacher models.

3.2.1 Qwen3-4B-Instruct-2507

The pre-trained language model on which RLHF and DPO is applied is chosen as the *Qwen3-4B-Instruct-2507* model. The model has 4 billion parameters, making it one of the student models that has been post-trained using strong-to-weak distillation. Furthermore, it has 36 layers, where multi-head attention consists of 32 heads, enabling capturing a diverse set of dependencies. In addition, to increase memory efficiency, only 8 key and value heads are used. In contrast to the base model, whose post-training is visualized in Figure 3.3, this model has been further post-trained to follow instructions and generate rapid responses. Therefore, it is particularly suitable when speed is prioritized over multi-step logical reasoning, making it an interesting model for summarization tasks [32].

3.2.2 Qwen3-30B-A3B-Instruct-2507

The pre-trained language model used to generate the AI feedback is chosen as the *Qwen3-30B-A3B-Instruct-2507* model. The model has 30.5 billion parameters, where 3.3 billion are activated, which makes it one of the student models that has been post-trained using strong-to-weak distillation. Furthermore, it has 48 layers, where multi-head attention consists of 32 heads, enabling capturing a diverse set of dependencies. In addition, to increase memory efficiency, only 4 key and value heads are used [33]. In contrast to the base model, whose post-training is visualized in Figure 3.3, this model has been further post-trained to follow instructions and generate rapid responses. The combination of instruction following, the number of parameters and the efficiency obtained by activating a subset of these makes the model suitable for ranking summaries.

3.3 Experimental Setup

In this work, the SFT model is initialized as the *Qwen3-4B-Instruct-2507* model. The trained SFT model is in turn, used to initialize the reward model, which uses a randomly initialized reward head. Furthermore, the SFT model and the reward

model are in turn used to initialize the models used in the third stage of the RLHF procedure. More precisely, the RL policy is initialized as the SFT model, while the value model is initialized as the reward model. The initialization procedure aligns with that chosen by [3]. Lastly, the DPO policy is initialized as the SFT model. In addition, all models are trained using LoRA adapters applied to W^Q and W^V , while all other parameters remain frozen. The exceptions are the reward model and the value model, both of which train the heads without LoRA.

Regardless of how LoRA is applied, all models require an optimization algorithm to update the model parameters. The chosen optimization algorithm is explained in Section 3.3.1. In addition, the chosen regularization technique is presented in Section 3.3.2. Furthermore, all training stages involve choosing multiple hyperparameters, which requires a setting selection approach, as it is not feasible to test every single combination. The setting selection approach is detailed in Section 3.3.3 together with the chosen settings.

3.3.1 Optimization Algorithm

RLHF and DPO involve a total of four distinct stages with corresponding objectives. All of these objectives require an optimization algorithm to find their minimum or maximum. In this project, the chosen optimization algorithm is Adam, as it offers a promising combination of momentum and root mean square propagation [34]. More precisely, Adam involves the first and second momentum estimates and a bias correction. Suppose that Adam is applied to update the parameter vector θ in the objective $\mathcal{L}(\theta)$, then the first momentum estimate is updated as

$$m_t = \beta_1 m_{t-1} + (1 - \beta_1) \nabla_{\theta} \mathcal{L}_t(\theta). \quad (3.1)$$

Similarly, the second momentum estimate is updated by

$$v_t = \beta_2 v_{t-1} + (1 - \beta_2) (\nabla_{\theta} \mathcal{L}_t(\theta))^2. \quad (3.2)$$

In Equations (3.1) and (3.2), $0 \leq \beta_1, \beta_2 \leq 1$ are hyperparameters that control how much the momentum estimates are updated. High β_i values will make small updates, producing stable but slow training, while low β_i values will make big updates, resulting in fast but potentially unstable training. The bias corrected estimate of Equation (3.1) is denoted $\hat{m}_t$ and is obtained by dividing m_t by $1 - \beta_1^t$. Similarly, the bias corrected estimate of Equation (3.2) is denoted $\hat{v}_t$ and obtained by dividing v_t by $1 - \beta_2^t$. Given the bias corrected estimates, Adam updates the parameter vector θ according to the update rule

$$\theta_t = \theta_{t-1} - \alpha \frac{\hat{m}_t}{\sqrt{\hat{v}_t} + \rho}. \quad (3.3)$$

In Equation (3.3), ρ is a hyperparameter that prevents division by zero, and $0 \leq \alpha \leq 1$ is a hyperparameter called the learning rate. Low learning rates lead to small updates, giving stable but slow training, while high learning rates lead to big updates, resulting in fast but potentially unstable learning. Moreover, the learning rate is often described as a constant throughout the entire training, but in practice it is often set to decrease to take smaller steps as the updates approach the optimal value. The decay can take many forms, but the default is

linear decay. An alternative form is cosine decay, which was used by [4]. The most straightforward way in these cases is to set an initial learning rate value and let that decay linearly or as a cosine curve to a small value for the duration of a fixed number of steps. However, that approach might point training to suboptimal regions, especially for cosine decay that rapidly decays the learning rate early in training. Therefore, decay can be combined with a warmup ratio, in which case the learning rate starts small, increases up to the initial learning rate, and then decreases down to a small value again. The warmup ratio defines the proportion of steps the learning rate has to increase from the small value to the initial value.

3.3.2 Regularization

RLHF and DPO are two complex approaches that are prone to suffer from over-fitting and over-optimization. More specifically, fine-tuning a language model using supervised learning could overfit the model, in which case it learns not only the underlying patterns but also the noise in the dataset [35]. As a result, the model performs well on the training dataset, but might fail to generalize to new data. Similarly, the reward model can be overfitted, making it sensitive to reward hacking. Reward hacking refers to the problem in which a reinforcement learning algorithm maximizes the formulated objective without improving the underlying objective [15]. More precisely, the reinforcement learning stage of RLHF can find a way to maximize the reward function, without actually improving the generation quality. Furthermore, DPO combines reward modeling and reinforcement learning into an objective over pairwise preferences, making that objective prone to both over-fitting and over-optimization. As a result, DPO can learn noise in the dataset rather than the underlying patterns, causing the objective to diverge from the actual goal [15]. Due to these reasons, training RLHF and DPO without proper regularization can result in models that generate gibberish completions.

In this work, early stopping is used as regularization to prevent over-fitting. Early stopping is obtained by monitoring the model performance on an unseen validation dataset during training. As the validation loss starts to increase, the training is stopped because it is an indication of over-fitting. The number of steps required without improvements before the training is stopped is decided by a hyperparameter called patience.

The reinforcement learning stage of RLHF is a bit special because it is an unsupervised learning approach that uses the current generated completion and the reward function to train the model. In other words, the reward model output is treated as the ground truth in contrast to the other stages that use a labeled dataset. Instinctively, one might suggest using the target completions and compare those to the generated completions to obtain a cross-entropy loss similar to that in supervised fine-tuning. However, that would conflict with the advantages of using PPO for RLHF. More precisely, the goal in RLHF is to train a reward model that gives a higher representation of quality compared to using a simple cross-entropy loss. As a result, stopping the reinforcement learning training of RLHF early based on cross-entropy loss might limit the potential of the model. Instead, over-optimization is regularized by the hyperparameter β . The hyperparameter is explained in Section 2.3.3 as the coefficient that controls how much the reward function is affected by the KL divergence penalty. Low β values miti-

gate the impact of the penalty, and make the objective equivalent to maximizing the reward model output. An objective of this form is very sensitive to reward hacking since nothing is preventing it from exploiting every imperfection of the proxy objective. High β values make the penalty dominate the reward function, resulting in an objective equivalent to minimizing the per-token probability distribution changes. An objective of this form prevents improvements to the SFT model since every change is penalized. The goal is to find a suitable intermediate β that allows the RL policy to improve from the SFT model by using the reward model as a judge of quality. In contrast to early stopping based on cross-entropy loss, which constrains changes to the SFT model, this approach penalizes changes while allowing the RL policy to improve.

3.3.3 Hyperparameters

In the previous sections, a total of 8 choices are presented with regard to LoRA, learning rate, and patience. Most of these choices are common for all stages of the RLHF and DPO approaches. However, the reinforcement learning stage of RLHF involves 3 additional hyperparameters that are distinct for reinforcement learning. The first distinct hyperparameter is the discount factor γ , which controls how the agent prioritizes between the immediate rewards and the future rewards. The second hyperparameter is the clip range ϵ , which controls the clip interval in the clipped objective. The third hyperparameter is the exponential weight discount λ , which controls the bias and variance tradeoff. In addition to these hyperparameters that are distinct for reinforcement learning, the objective used to train the RL policy involves 3 additional hyperparameters and choices. The hyperparameters are the value term coefficient c_1 and the KL divergence coefficient β , which is also used by the DPO objective. The last choice is the KL divergence estimator.

In total, the reinforcement learning stage of RLHF involves 13 hyperparameters or choices, while the remaining RLHF stages involve 8 and the final stage of DPO involve 9. In practice, it is not feasible to consider all combinations of these hyperparameters and choices when training the models. Therefore, a greedy setting selection approach is combined with small experiments to find the settings for each stage. The greedy setting selection approach refers to starting from the default settings and changing a single hyperparameter or choice at a time. By retaining the best setting and continuing by changing another hyperparameter or choice, the greedy settings are achieved. Combining the greedy setting selection approach with small experiments, where the models are trained using a small subset of the dataset, enables testing multiple settings somewhat efficiently. The greedy setting selection approach has its limitations, as the optimal setting for one hyperparameter might change with the setting of another hyperparameter. However, it gives better results than the default settings for the small experiments while being a feasible approach for a degree project.

The settings for the RLHF and DPO stages are tuned using the greedy setting selection approach. The final hyperparameters and choices made for these stages are summarized in Table 3.3. In addition, the settings presented in Table 3.3 are also used for the corresponding AI version of the stages because no further tuning of the hyperparameters is performed for those stages.

Parameter	SFT	RM	RL	DPO
Rank r	16	16	16	16
Scaling factor η	32	32	32	32
Coefficient β_1	0.9	0.95	0.9	0.9
Coefficient β_2	0.999	0.999	0.999	0.999
Learning rate α	$1 \cdot 10^{-4}$	$2 \cdot 10^{-5}$	$5 \cdot 10^{-5}$	$2 \cdot 10^{-5}$
Patience	5	10	-	10
Learning rate type	Cosine	Cosine	Linear	Linear
Warmup ratio	0.1	0.1	0.1	0.1
Discount factor γ	-	-	1.0	-
Clip range ϵ	-	-	0.2	-
Exponential weight discount λ	-	-	0.95	-
Coefficient c_1	-	-	0.1	-
KL divergence estimator	-	-	Third	-
Coefficient β	-	-	0.05	0.1

Table 3.3: Final hyperparameters and choices. The hyperparameters and choices used to train the models in RLHF and DPO. The corresponding AI versions of the models are trained using the same settings.

Chapter 4

Results and Analysis

In this chapter, learning curves are visualized, evaluation metrics are presented, and generated summaries are given. The chapter begins with the results obtained by the SFT and SFTAI models in Section 4.1. The chapter continues with the RM and RMAI results in Section 4.2. In Section 4.3, the results of RL and RLAI are presented. Lastly, the DPO and DPOAI results are provided in Section 4.4.

4.1 Supervised Fine-Tuning Results

The first stage of RLHF and DPO is to fine-tune the *Qwen3-4B-Instruct-2507* model using supervised learning. During training, the loss function given in Equation (2.57) is monitored using the training and validation datasets, as visualized in Figure 4.1a. In addition, the fraction of target tokens that is assigned the highest probability for that position is monitored using the training and validation datasets, as visualized in Figure 4.1b. In the learning curves, these metrics are plotted against the number of optimization steps taken. Each step corresponds to 64 samples, as the batch size is 16 and 4 GPUs are used during training [36].

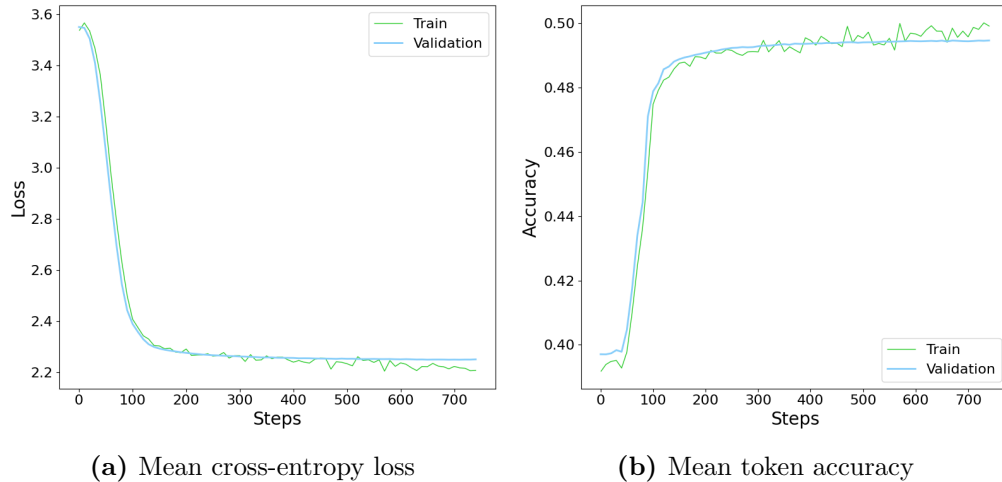


Figure 4.1: Learning curves for SFT model. The plotted metrics are the cross-entropy loss and the token accuracy, both averaged over the current logging interval, which is set to 10 steps. The metrics are monitored on the training and validation datasets, as visualized by the green and blue curves, respectively. The curves show steady and fast improvements with the number of optimization steps.

The learning curves in Figure 4.1 show steady and fast improvements in both cross-entropy loss and token accuracy. Training is stopped after 3.149 epochs, after the minimum validation loss of 2.248 is obtained. The minimum validation loss corresponds to the geometric mean probability of $\exp(-2.248) \approx 0.106$ to choose the target token in each position. Furthermore, the maximum validation token accuracy obtained is 0.494, which means that approximately half of the target tokens are assigned the highest probability for that position. Initially, one might suspect the probability and the accuracy to be closer to each other. However, since the model assigns a probability to all tokens in the vocabulary, it is reasonable that a probability of approximately 0.106 is the highest probability for approximately half of the tokens. The performance of the SFT model is further investigated by ROUGE and BERTScore presented in Table 4.1.

In addition, the *Qwen3-4B-Instruct-2507* model is fine-tuned using supervised learning and the SFTAI dataset. During training, the cross-entropy loss and the token accuracy are monitored using the training and validation datasets, as visualized in Figures 4.2a and 4.2b, respectively. In the figures, these metrics are plotted against the number of optimization steps taken, where each step considers 64 samples.

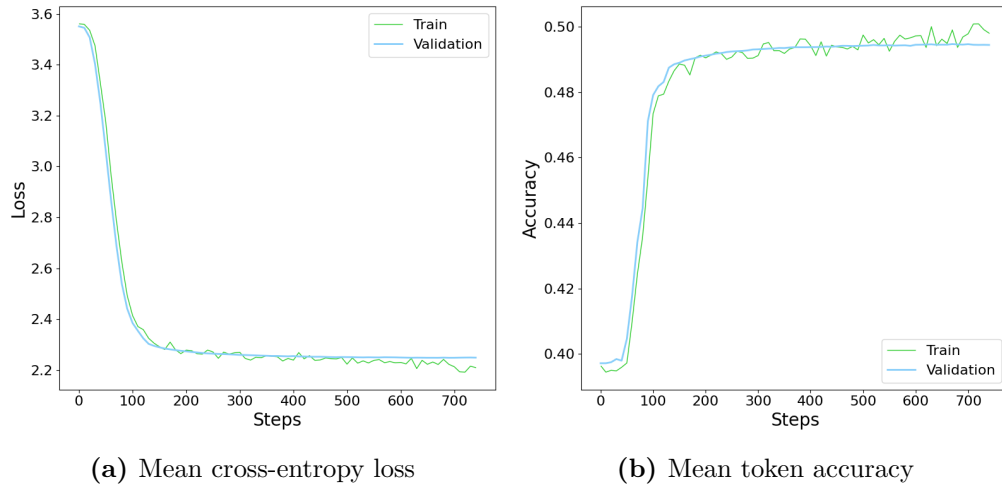


Figure 4.2: Learning curves for SFTAI model. The plotted metrics are the cross-entropy loss and the token accuracy, both averaged over the current logging interval, which is set to 10 optimization steps. The performance is monitored using the metrics on both the training dataset, as visualized by the green curves, and on the validation dataset, as visualized by the blue curves. The learning curves show steady and fast improvements with the number of optimization steps.

The learning curves in Figure 4.2 show similar results to those in Figure 4.1. The cross-entropy loss and the token accuracy improve steadily and fast with the number of optimization steps. Training is stopped after 3.136 epochs, after the minimum validation loss of 2.248 is obtained, which is the same as that obtained by the SFT model. In addition, the maximum validation token accuracy obtained is 0.495, which is slightly higher than that obtained by the SFT model. The performance of the SFTAI model is further investigated by ROUGE and BERTScore presented in Table 4.1.

Table 4.1 presents the ROUGE-1, ROUGE-2, and ROUGE-L F1 scores obtained by the summaries generated by the SFT and SFTAI models, respectively. These summaries are generated for the Reddit posts in the test dataset and compared to the reference summaries in the said dataset. The summaries are also compared using the precision, recall, and F1 score of the BERTScore. In addition, the table includes the metrics obtained by the supervised fine-tuned model trained by [3], which is presented in Table 15 of their article.

Model	F1 _{ROUGE-1}	F1 _{ROUGE-2}	F1 _{ROUGE-L}	P _{BERT}	R _{BERT}	F1 _{BERT}
SFT	0.367	0.166	0.298	0.426	0.371	0.397
SFTAI	0.369	0.165	0.300	0.437	0.363	0.399
OpenAI	0.425	0.198	0.395	-	-	-

Table 4.1: Performance of SFT and SFTAI models. Performance is assessed using the ROUGE and BERTScore. The metrics are computed to compare the summaries generated by the SFT and SFTAI models with the reference summaries in the test dataset. The performance of the supervised fine-tuned model trained by [3] is included for comparison.

Table 4.1 shows that the models trained in this work achieve somewhat lower results than the SFT model trained by [3]. However, [3] used a model with 6.7B parameters and trained it using a dataset called *CNN/DM*. In addition, [3] found that models trained using the *CNN/DM* dataset copy more from the input than models trained using the *TL;DR* dataset. Considering that the SFT and SFTAI models trained in this work have fewer parameters, even fewer trainable parameters, and are trained using a dataset more similar to the *TL;DR* dataset, their performances are comparable to those achieved by [3]. Furthermore, since the reference summaries are somewhat short, even a small difference in overlaps results in a high ROUGE metrics difference. More precisely, the reference summaries contain on average approximately 29 words, resulting in a ROUGE-1 recall equal to 0.379 for an overlap of 11 words and a ROUGE-1 recall equal to 0.414 for an overlap of 12 words.

Table 4.1 also shows that the BERTScore precision is higher than the BERTScore recall, indicating that the generated summaries are more similar to the references than the references are to the generated summaries. In other words, the reference summaries contain some tokens that are not well preserved by the generated summaries. In addition, Table 4.1 shows that the BERTScore F1 scores are higher than the ROUGE-1 F1 scores, indicating that tokens that do not perfectly overlap between references and generations still have a high similarity. Lastly, the table shows that the SFTAI model achieves a performance comparable to that of the SFT model. The difference between performance is further investigated by the example given in Table 4.2. The table gives an example Reddit post, the reference summary, and the summaries generated by the SFT and SFTAI models.

Reddit post	Hello Reddit, First time poster in this subreddit. It’s been a about 14 months since I’ve graduated undergraduate, and I’ve been having trouble finding an entry level, full time position in either marketing or sales. When I first graduated, I had a fair share of interviews and recruiters contacting me, but I had received no formal offer or I ended up through the process realizing the company might not be a good fit. I’m underemployed at the moment working a job paying \$13/hr that doesn’t require a Bachelor’s degree, so I’m keeping my head above water in paying my student loans and other bills, but just barely. I live at home and I am saving money, but I can’t help but feel discouraged that I “SHOULD” have a salaried position at this point, but I don’t. I tweak my resume to the job description with keywords, I actively use LinkedIn, reach out for networking, and nothing seems to materialize other than the automated rejection email sent by computers. I apply to at the very least, 2 jobs per week, and I feel like my resume doesn’t even get glanced at upon by human eyes at this point, even when I include a personalized cover letter with my application. I’m wondering if there are any other redditors on here with similar experiences struggling to find their “career” jobs as well within their recent years of post undergraduate life, because sometimes I feel like I might be the only one. If anyone can share their experiences, I would LOVE to hear them.
Reference summary	I’m underemployed and I’m wondering if anyone else had similar experiences and how they overcame them.
SFT generation	I’m a recent graduate, and I’m having trouble finding a full time, entry level position in marketing or sales. I’m underemployed at the moment, and I’m wondering if there are any other redditors on here with similar experiences.
SFTAI generation	I’m a recent graduate, and I’m having trouble finding a full time, entry level position in either marketing or sales.

Table 4.2: Example post, reference summary and generated summaries. The Reddit post and reference summary are taken from the test dataset. The generated summaries are generated by the SFT and SFTAI models.

Table 4.2 shows that the summary generated by the SFT model is longer than that generated by the SFTAI model. This aligns with the average as the SFT and SFTAI generations contain on average approximately 60 and 45 tokens, respectively. The table also shows that generations do not preserve content on how to overcome the situation, exemplifying the cause of lower recall.

4.2 Reward Modeling Results

The second stage of RLHF is to train a reward model to predict human preference. The model is initialized from the SFT model, but with a randomly initialized reward head. During training, the loss function given in Equation (2.64) is monitored using the training and validation datasets as visualized in Figure 4.3a. In addition, the fraction of preferred summaries for which predicted rewards are higher than that of rejected summaries is monitored using training and validation datasets as visualized in Figure 4.3b. In the learning curves, these metrics are plotted against the number of optimization steps taken, where each optimization step considers a batch size of 8 samples.

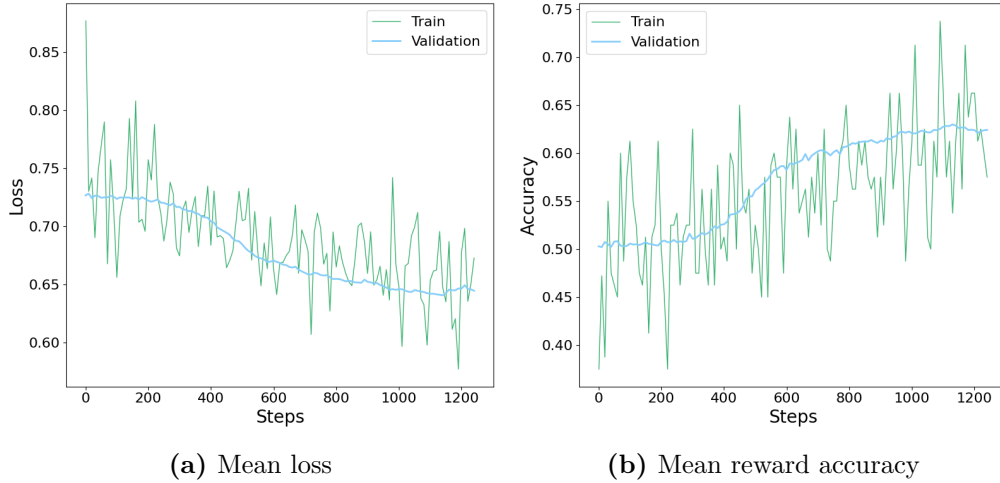


Figure 4.3: Learning curves for RM. The plotted metrics are the loss and reward accuracy, both averaged over the current logging interval, which is set to 10 optimization steps. The performance is monitored using the metrics on both the training dataset, as visualized by the green curves, and on the validation dataset, as visualized by the blue curves. The learning curves show oscillations, but consistent improvements with the number of optimization steps.

The learning curves in Figure 4.3 show more oscillations and less steady improvements in both loss and accuracy compared to the curves in Figure 4.1. This is expected because fewer samples are considered in each optimization step during RM training compared to SFT training, causing greater variance between steps. In addition, RM training is known to be unstable due to the noisy preferences of human labelers. However, despite the oscillations, the learning curves show consistent improvements for the duration of the training, which lasts for 0.321 epochs. In this case, a higher patience would be appropriate as the loss starts to decrease again before the training is stopped. This suggests that the training momentarily ended up in a poor region rather than overfitting. Fortunately, both curves show signs of convergence, suggesting that further training would not improve the model much. Moreover, the obtained validation reward accuracy lies in the lower range of what is achieved by [3], who plotted their validation reward accuracy against different model sizes with respect to the size of the training dataset, as visualized in Figure 6 of their article. The plot ranges from a reward accuracy of approximately 0.61 for a model of size 160M trained on a dataset of size 8000 to a reward accuracy of approximately 0.75 for a model of size 13B trained on

a dataset of size 64000. Considering the size of the model and the size of the training dataset in combination with the limitations discussed in Chapter 5, the achieved validation reward accuracy of 0.629 is within a reasonable range compared to what [3] achieved. Lastly, the minimum validation loss obtained is 0.641.

In addition, the AI version of the reward model is trained using the RMAI dataset. Similarly as in the RM case, the RMAI model is initialized as the SFTAI model with a randomly initialized reward head. During training, the loss and reward accuracy are monitored using the training and validation datasets, as visualized in Figures 4.4a and 4.4b, respectively. In the figures, these metrics are plotted against the number of optimization steps taken, where each step considers 8 samples.

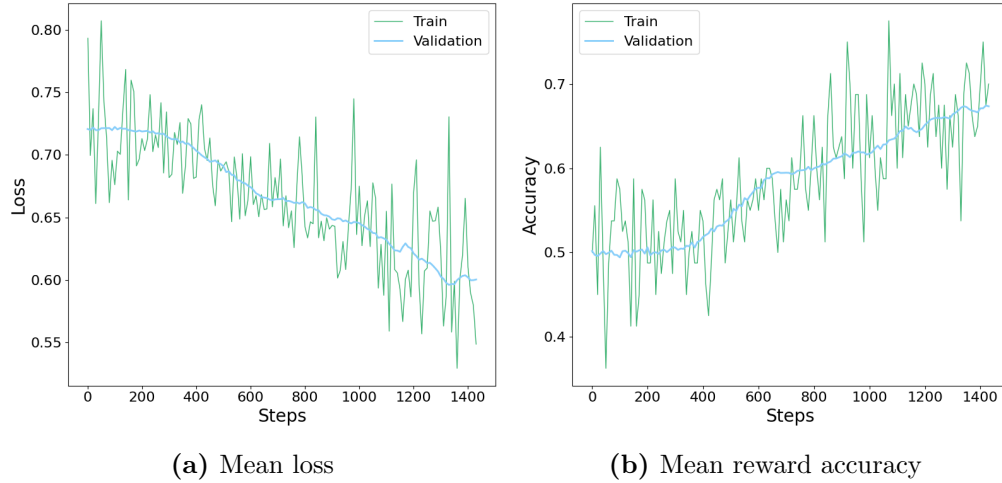


Figure 4.4: Learning curves for RMAI. The plotted metrics are the loss and reward accuracy, both averaged over the current logging interval, which is set to 10 optimization steps. The performance is monitored using the metrics on both the training dataset, as visualized by the green curves, and on the validation dataset, as visualized by the blue curves. The learning curves show oscillations, but consistent improvements with the number of optimization steps. In addition, the curves suggest that further improvements are expected with higher patience.

The learning curves in Figure 4.4 show some similarities and some dissimilarities with those in Figure 4.3. The similarities are that the learning curves show oscillations while consistently improving with the number of optimization steps. The dissimilarities are that they obtain a lower validation loss, a higher validation reward accuracy, and show little to no sign of convergence. This suggests that the reward model benefits from being initialized as the SFTAI model and training it using the RMAI dataset. In addition, further improvements are expected with a higher patience. However, the obtained validation reward accuracy of 0.672 is still reasonable given the validation reward accuracy obtained by [3] and the limitations discussed in Chapter 5. Furthermore, the minimum validation loss is obtained at 0.596, making the training stop at 0.370 epochs.

4.3 Reinforcement Learning Results

The third stage of RLHF is to align the SFT model with human preferences using the reward model as a proxy objective and the reinforcement learning algorithm PPO. In addition, the stage involves training a value model to predict the reward function. In this work, the value model is initialized as the reward model. During training, the clipped objective given in Equation (2.48) is monitored as a loss, as visualized in Figure 4.5a. In addition, the squared error loss given in Equation (2.54) is monitored as visualized in Figure 4.5b. Furthermore, the estimated KL divergence between the per-token probability distributions of the SFT model and the RL policy given in Equation (2.67) is monitored as visualized in Figure 4.5c. Lastly, the scalar reward obtained from the reward model is monitored as visualized in Figure 4.5d. In the learning curves, these metrics are plotted against the number of optimization steps taken, where each step considers 4 samples.

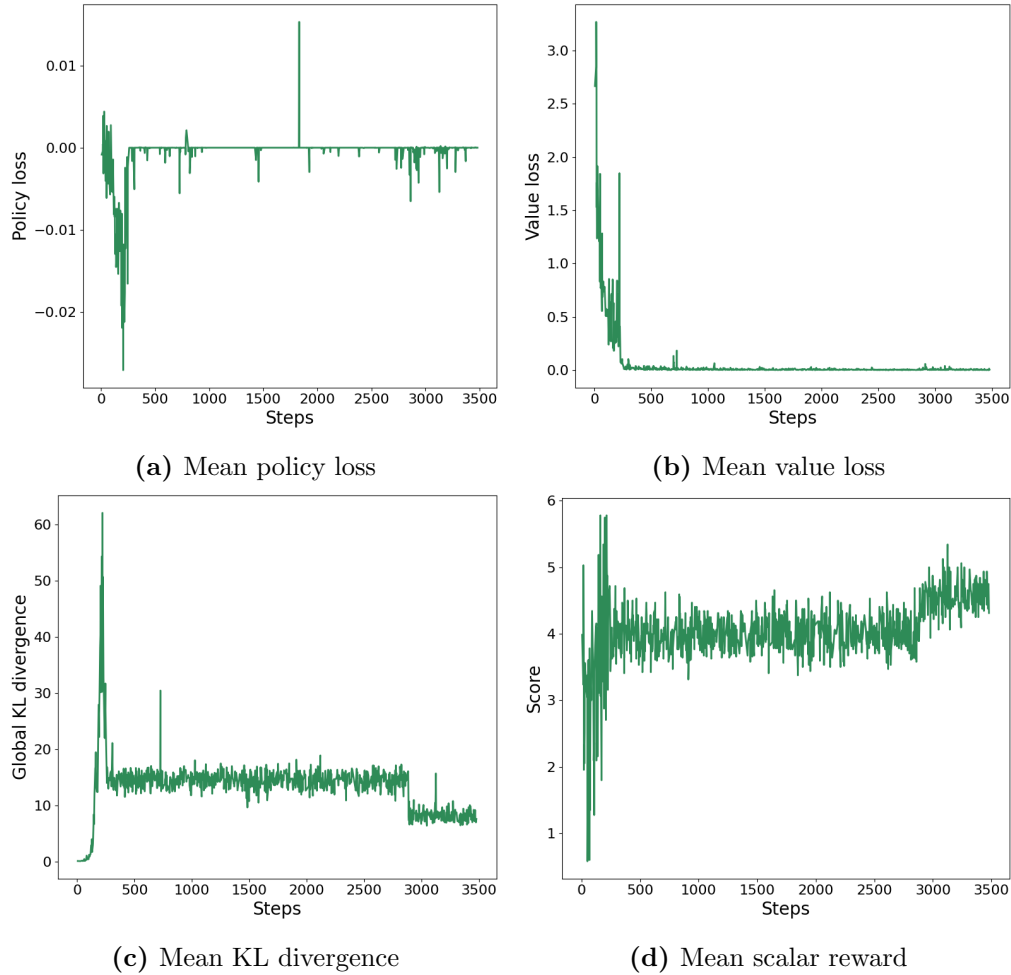


Figure 4.5: Learning curves for RL policy. The plotted metrics are the policy loss, the value loss, the KL divergence between the SFT per-token probability distribution and that of the RL policy, and the scalar rewards returned by the reward model. The metrics are plotted against the number of optimization steps and averaged over the samples in the corresponding step. The learning curves demonstrate a deterministic policy and a near-perfect value model. Combining these curves with the high KL divergence suggests reward hacking.

The policy loss monitored in Figure 4.5a shows a fast decrease followed by an increase and a plateau around 0. The curve indicates that the RL policy has become more or less deterministic, where the same trajectories are sampled. This mitigates updates to the RL policy and results in an advantage function approximately equal to 0. The hypothesis is confirmed by the monitored policy entropy, whose learning curve is plotted in Appendix B.2. Furthermore, the deterministic policy makes the return function predictable, which enables training the near-perfect value model, whose training is visualized in Figure 4.5b. The sudden peak and drop in value loss is explained by the KL divergence, whose learning curve is visualized in Figure 4.5c. Initially, the agent is exploring, causing the KL divergence to increase. However, as the KL divergence increases too much, it begins to dominate the reward function, resulting in a return drop, as visualized by the reward function learning curve in Appendix B.2. The agent realizes the mistake, and moves the per-token probability distribution of the RL policy back to that of the SFT model. However, the value model is not aware of this shift and continues to predict a return based on the high KL divergence before adjusting to the new situation. Combining these learning curves suggests that the monitored scalar reward visualized in Figure 4.5d tells a less fortunate story than it looks like. The scalar rewards improve with the number of optimization steps, where oscillations are high during early stages of training when the agent explores and the value model is trained. However, combining the scalar reward learning curve with a deterministic policy, the near-perfect value model, and the high KL divergence suggests reward hacking.

Reward hacking is a common problem in RLHF that occurs because the RL policy finds and exploits a weakness in the reward model. Generating summaries to the posts in the test dataset using the RL policy confirms that reward hacking occurs. The weakness that the RL policy finds and exploits is that the reward model gives high scalar rewards to the word “Please”. This causes the RL policy to generate that word only multiple times for all posts. In turn, the achieved ROUGE F1 scores are equal to 0.000, while the achieved BERTScore metrics are negative. The negative BERTScore metrics mean that the generated summaries are worse than the random baseline computed by [27].

In addition, the RLAI policy is trained from the SFTAI model using the AI version of the reward model as a proxy objective and PPO. The AI version of the reward model is also used to initialize the value model. During training, the clipped objective is monitored as visualized in Figure 4.6a. The squared error loss is monitored as visualized in Figure 4.6b. The estimated KL divergence between the per-token probability distributions is monitored as visualized in Figure 4.6c. Lastly, the scalar reward is monitored as visualized in Figure 4.6d. In the learning curves, these metrics are plotted against the number of optimization steps taken, where each step considers a batch size of 4 samples.

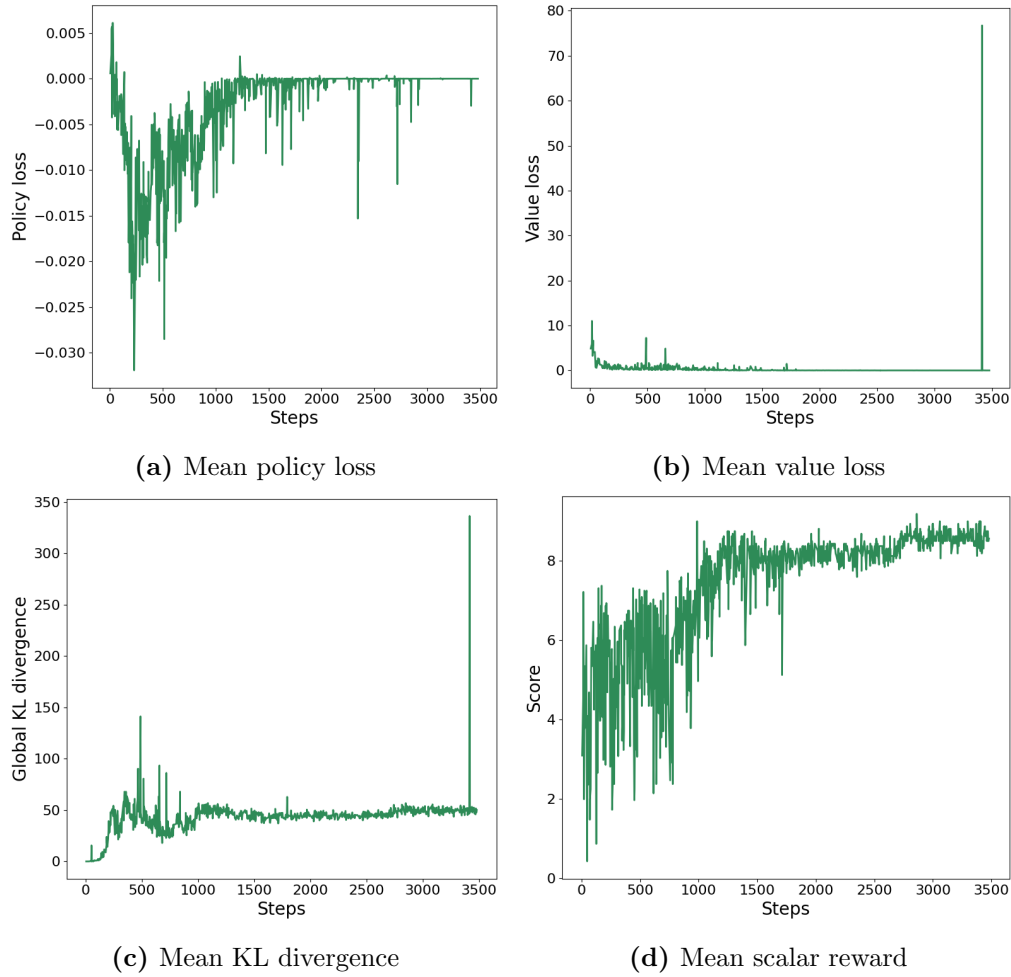


Figure 4.6: Learning curves for RLAI policy. The plotted metrics are the policy loss, value loss, KL divergence between the SFTAI per-token probability distribution and that of the RLAI policy, and the scalar rewards returned by the AI version of the reward model. The metrics are plotted against the number of optimization steps and averaged over the samples in the corresponding step. The learning curves demonstrate a deterministic policy and a near-perfect value model. Combining these curves with the high KL divergence suggests reward hacking.

The learning curves in Figure 4.6 show similar results to those in Figure 4.5. The policy becomes deterministic, which gives an almost perfect value model, as visualized in Figure 4.6b. In addition, the per-token probability distribution of the RLAI policy drifts far away from that of the SFTAI model, as visualized by the KL divergence in Figure 4.6c. Lastly, the scalar rewards obtained by the AI version of the reward model show consistent improvements with the number of optimization steps despite oscillations early in the training. The main dissimilarities are that the RLAI policy is trained longer before becoming deterministic, and it reaches a higher KL divergence as well as scalar reward. Unfortunately, the signs of reward hacking are still present. Reward hacking is confirmed by generating summaries to the posts in the test dataset. The generated summaries are all repetitive versions of “I know that I can’t control what other people do, but I can control how I react”, resulting in ROUGE and BERTScore metrics barely better than those obtained by the RL policy.

4.4 Direct Preference Optimization Results

The DPO approach involves two stages, where the first is to train the SFT model and the second is to preference fine-tune the SFT model using the RM dataset. The results for the SFT model are presented in Section 4.1, while the preference fine-tuning results are presented in this section. During training, the loss function given in Equation (2.74) is monitored using the training and validation datasets, as visualized in Figure 4.7a. In addition, the fraction of preferred summaries for which implicit rewards are higher than that of rejected summaries is monitored using the training and validation datasets as visualized in Figure 4.7b. In the learning curves, these metrics are plotted against the number of optimization steps taken. In each optimization step, 32 samples are considered, as the batch size is 8 and 4 GPUs are used during training [36].

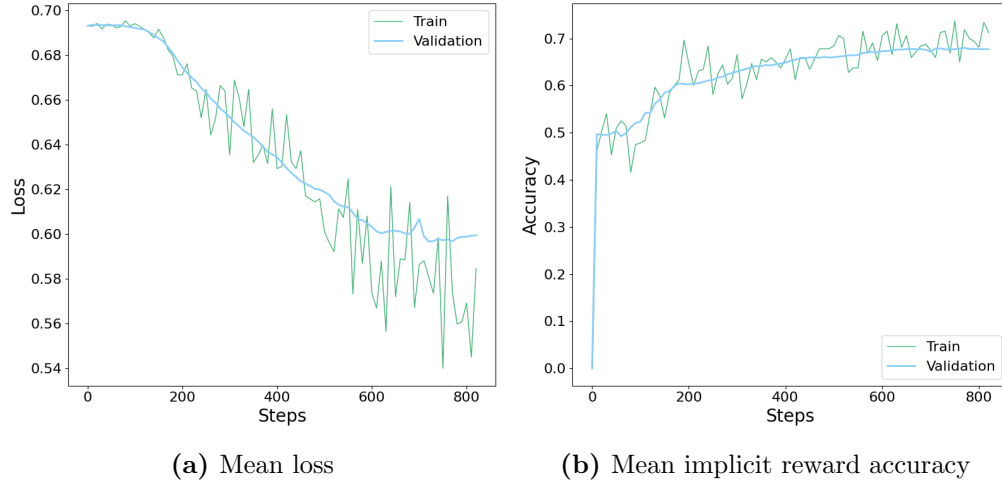


Figure 4.7: Learning curves for DPO policy. The plotted metrics are the loss function and the implicit reward accuracy, both averaged over the current logging interval, which is set to 10 steps. The performance is monitored using the metrics on both the training dataset and on the validation dataset, as visualized by the green and blue curves, respectively. The learning curves show somewhat steady and fast improvements with the number of optimization steps.

The learning curves in Figure 4.7 show somewhat fast and steady improvements in loss and implicit reward accuracy with the number of optimization steps. Training is stopped after 0.847 epochs without clear signs of overfitting, suggesting that the patience of 10 might be too restrictive. However, the curves show clear signs of convergence, indicating that the minimum validation loss of 0.597 is at least close to the global minimum. Furthermore, the maximum validation reward accuracy obtained is 0.679, which is higher than the explicit reward accuracy obtained by the reward models. The performance of the DPO policy is further investigated by ROUGE and BERTScore presented in Table 4.3.

In addition, the DPOAI policy is trained from the SFTAI model. During training, the loss function and implicit reward accuracy are monitored using the training and validation datasets, as visualized in Figures 4.8a and 4.8b, respectively. In the figures, these metrics are plotted against the number of optimization steps, where each step considers a batch size of 32 samples.

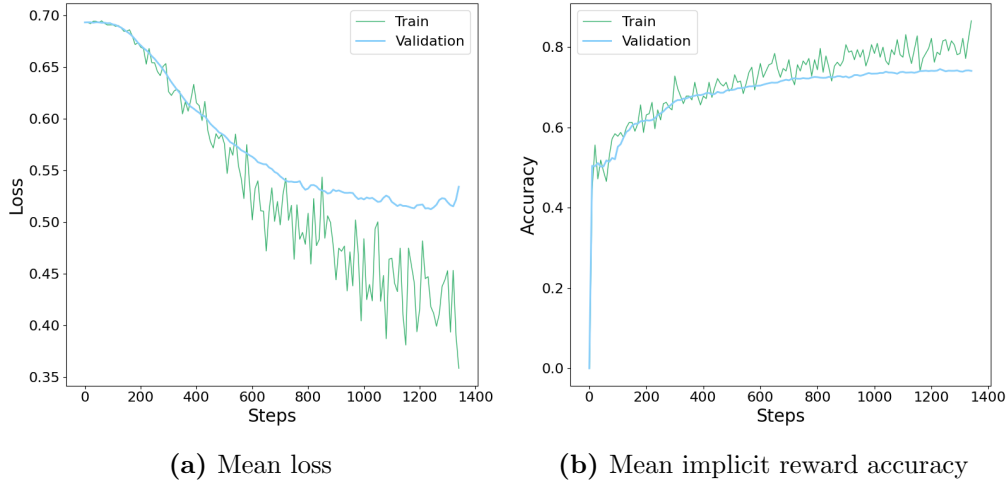


Figure 4.8: Learning curves for DPOAI policy. The plotted metrics are the loss function and the implicit reward accuracy, both averaged over the current logging interval, which is set to 10 steps. The performance is monitored using the metrics on both the training dataset and on the validation dataset, as visualized by the green and blue curves, respectively. The learning curves show somewhat steady and fast improvements with the number of optimization steps.

The learning curves in Figure 4.8 show improvements to those in Figure 4.7. The loss and implicit reward accuracy improve faster and more stable with the number of optimization steps. The duration of training is longer, reaching 1.384 epochs. Furthermore, a lower validation loss is obtained at 0.512 and a higher implicit validation reward accuracy is obtained at 0.743. The performance of the DPO and DPOAI policies is further investigated by the metrics presented in Table 4.3.

Table 4.3 presents the ROUGE-1, ROUGE-2, and ROUGE-L F1 scores obtained by summaries generated by the DPO and DPOAI policies, respectively. These summaries are generated for the Reddit posts in the test dataset and compared to the reference summaries in the said dataset. The summaries are also compared using the precision, recall, and F1 score of the BERTScore.

Model	F1 _{ROUGE-1}	F1 _{ROUGE-2}	F1 _{ROUGE-L}	P _{BERT}	R _{BERT}	F1 _{BERT}
DPO	0.387	0.170	0.297	0.384	0.481	0.431
DPOAI	0.362	0.143	0.269	0.359	0.458	0.407

Table 4.3: Performance of DPO and DPOAI policies. Performance is assessed using the ROUGE and BERTScore metrics. The metrics are computed to compare the summaries generated by the DPO and DPOAI policies with the reference summaries in the test dataset.

Table 4.3 shows a somewhat large difference between BERTScore precision and BERTScore recall, where the recall is higher. This indicates that the references are more similar to the generated summaries than the generated summaries are to the references. In other words, the generated summaries contain some tokens that do not match the tokens in the references too well. In addition, the BERTScore F1 score is higher than the ROUGE metrics F1 scores, indicating that the reference

and generated tokens that do not perfectly overlap, still have a high similarity. Lastly, Table 4.3 shows that the DPOAI policy obtains slightly lower performance compared to the DPO policy. The difference between performance is further investigated by the example given in Table 4.4.

Reddit post	Hello Reddit, First time poster in this subreddit. It’s been a about 14 months since I’ve graduated undergraduate, and I’ve been having trouble finding an entry level, full time position in either marketing or sales. When I first graduated, I had a fair share of interviews and recruiters contacting me, but I had received no formal offer or I ended up through the process realizing the company might not be a good fit. I’m underemployed at the moment working a job paying \$13/hr that doesn’t require a Bachelor’s degree, so I’m keeping my head above water in paying my student loans and other bills, but just barely. I live at home and I am saving money, but I can’t help but feel discouraged that I “SHOULD” have a salaried position at this point, but I don’t. I tweak my resume to the job description with keywords, I actively use LinkedIn, reach out for networking, and nothing seems to materialize other than the automated rejection email sent by computers. I apply to at the very least, 2 jobs per week, and I feel like my resume doesn’t even get glanced at upon by human eyes at this point, even when I include a personalized cover letter with my application. I’m wondering if there are any other redditors on here with similar experiences struggling to find their “career” jobs as well within their recent years of post undergraduate life, because sometimes I feel like I might be the only one. If anyone can share their experiences, I would LOVE to hear them.
Reference summary	I’m underemployed and I’m wondering if anyone else had similar experiences and how they overcame them.
DPO generation	Graduated undergrad 14 months ago, struggling to find entry level full time position in marketing or sales, feel like my resume doesn’t even get glanced upon by human eyes, wondering if anyone else has similar experiences.
DPOAI generation	Struggling to find entry level, full time position in marketing or sales despite consistent applications, networking, and resume tweaks. Feeling discouraged and wondering if others have similar experiences.

Table 4.4: Example post, reference summary and generated summaries. The Reddit post and reference summary are taken from the test dataset. The generated summaries are generated by the DPO and DPOAI policies.

Table 4.4 shows that the generated summaries are approximately of the same lengths. This aligns with the average as the DPO generations contain an average number of approximately 56 tokens, while the DPOAI generations contain an average number of approximately 58 tokens. The table also shows that the generations include more content with regard to the situation, exemplifying the cause of the lower precision.

Chapter 5

Discussion

The purpose of the work is to compare RLHF and DPO in terms of stability, performance, robustness to AI feedback, and objectives. In this chapter, these aspects of the results are discussed. More precisely, the chapter begins with Section 5.1 where the stability is discussed. The following section, Section 5.2, discusses the performances. In the next section, Section 5.3, the robustness to AI feedback is discussed. Lastly, Section 5.4 discusses the objectives. In addition, limitations and potential improvements are presented and discussed in Section 5.5.

5.1 Stability

In Chapter 4 the learning curves associated with each training stage of the RLHF and DPO approaches are visualized. The learning curves show steady and fast improvements for SFT training. The RM and DPO learning curves show consistent improvements with the number of optimization steps, but with more oscillations than the SFT learning curves. The most interesting learning curves are those associated with the RL training stage. These learning curves show that reward hacking occurs in the RLHF approach, demonstrating the main disadvantage of RLHF. The main disadvantage of RLHF is that the approach is unstable and prone to reward hacking the proxy objective, causing gibberish generations. The unstable nature of RLHF is established in related work and motivated the alternative approach, DPO [5]. However, it is interesting to find out just how unstable RLHF is compared to DPO. In this work, they are trained from the same SFT model using approximately the same dataset, but despite this, DPO and RLHF demonstrate completely different behaviors. DPO is able to generate appropriate summaries, while RLHF only generates the word “Please”.

5.2 Performance

The RLHF reward hacking problem causes the approach to generate gibberish summaries, which makes the DPO approach superior in performance. Clearly, generating summaries better than the word “Please” does not provide much information about actual performance, but comparing results between the SFT model and the DPO policy indicates good performance. The one metric for which the SFT model achieves clearly better performance is BERTScore precision. However, these metrics can be misleading if the reference summaries do not demonstrate perfect behavior. More precisely, Table 4.4 shows that the summary generated by

the DPO policy preserves more content of the post than the reference summary. This inevitably causes a lower precision, although the quality can be considered higher. Therefore, to complement the ROUGE and BERTScore metrics, a small subset of generated summaries is also compared by a human judge. The subset includes 10 posts with corresponding summaries, one of these is presented in Tables 4.2 and 4.4, while another is presented in Appendix C.1. The generated summaries are assessed based on how much content from the post they preserve, and for that aspect the DPO generated summaries are better. Combining the obtained ROUGE and BERTScore metrics with the example generations suggests that the DPO approach is superior to the SFT model, as it obtains higher overall metrics and preserves more content of the posts considered by the human judge.

5.3 Feedback Robustness

The learning curves in Chapter 4 suggest that the RLHF and DPO approaches benefit from training on AI feedback. The curves are more stable and achieve better validation metrics during training. This is expected since the AI feedback is less noisy than that obtained by the human labelers and is generated by a model of the same model family as that used to train the approaches. In other words, the *Qwen3-4B-Instruct-2507* model and the *Qwen3-30B-A3B-Instruct-2507* model have been through approximately the same training pipeline, making it easier for the reward model to understand and align with the preferences. Despite the better learning curves and validation metrics obtained during training, the ROUGE and BERTScore metrics are, in general, slightly lower for the AI versions of all stages. However, the ROUGE and BERTScore metrics can be misleading as the reference summaries might not demonstrate perfect behavior. In other words, if the generated summaries preserve more content of the Reddit post than the reference summary, the metrics are penalized despite potentially higher summary quality. Therefore, to complement the ROUGE and BERTScore metrics, a small subset of generated summaries is also compared by a human judge. The generated summaries are assessed based on how much content from the post they preserve, and for that aspect the summaries generated by the SFTAI, RLAI and DPOAI models are comparable to those generated by the SFT, RL, and DPO models, respectively. Combining the learning curves, the validation metrics obtained during training, the human comparison, and the ROUGE and BERTScore metrics suggest that all approaches are robust to AI feedback.

5.4 Objective

The results demonstrate the main advantage of DPO, that it is a stable approach, and the main disadvantage of RLHF, that the approach is sensitive to reward hacking. However, the main disadvantage of DPO is that it is less flexible compared to RLHF, which is the main advantage of RLHF. More precisely, the preference fine-tuning stage of DPO involves a single model trained using an objective that assumes that the naive KL divergence estimator is used. In contrast, the preference fine-tuning stage of RLHF involves two separate models with two separate objectives, both of which involve a number of hyperparameters and choices. The multiple objectives and choices make the RLHF approach more unstable, but also more flexible. In other words, DPO offers a stable approach, while RLHF offers a

flexible approach. Therefore, an untested hypothesis is that RLHF have greater potential than DPO, but also require a more carefully chosen experimental setup.

5.5 Limitations and Improvements

In this work, the *summarize from feedback* dataset is used. The dataset consists of Reddit posts with corresponding preferred and rejected summaries, generated by the GPT-3 models. Although the dataset enables training on human feedback without human labelers, it introduces some limitations. The main limitation of the dataset is that the summaries are generated by a language model, and therefore, they might not demonstrate desired behavior. Training the SFT and SFTAI models to approach poor targets inevitably limits the potential of the models. As a result, the potential of the remaining models in the approaches is limited since they are initialized from the SFT and SFTAI models. An alternative approach is to train the SFT and SFTAI models on the *TL;DR* dataset, which contains posts and summaries written by the author of the post.

In addition, combining poor summaries with the sophisticated pre-trained language model *Qwen3-4B-Instruct-2507* can cause suboptimal behavior. More precisely, the model has been trained to potentially generate summaries of higher quality than those present in the dataset. Because of this, there is a risk that the reward model views all summaries in the dataset as bad. This would cause confusion, making the model look for any discrepancy between preferred and rejected summaries that aligns with what it has learned prior during the GRPO stage. The risk is that the only alignments are low-quality discrepancies such as summary lengths or nice phrases like “Please” and “Thanks in advance”. By exploiting such low-quality discrepancies, each improvement to the loss actually makes the reward model more sensitive to reward hacking. It is hard to decide to what extent there is a quality mismatch between the dataset and model, and how much it contributes to reward hacking, but some signs are present. The sign of quality mismatch is present during the AI feedback generation. During generation, *Qwen3-30B-A3B-Instruct-2507* repeated that none of the presented summaries were good despite being prompted to only generate a ranking. Furthermore, the RL generated summaries suggest that the mismatch also contributes to reward hacking. The RL policy presented in the results tricks the reward model by only generating the word “Please”. In addition, RL policies trained in the setting selection experiments often found that the phrase “Thanks in advance” yielded the highest scalar rewards. This suggests that the reward model finds these low-quality discrepancies rather than nuances correlations between summaries and qualities. To mitigate this reward hacking effect, one approach is to filter out common phrases in all summaries.

Another limitation caused by poor target summaries in the dataset is that the ROUGE and BERTScore metrics could give misleading results. The metrics accurately represent summary quality if the references demonstrate desired behavior. However, if the generated summaries preserve more information from the post compared to the reference summaries, lower metrics are obtained despite better summary quality. This limitation is demonstrated in Tables 4.2 and 4.4. The tables show that the generated summaries preserve more information about the situation the poster is in compared to the limited information present in the refer-

ence, causing misleading ROUGE and BERTScore values. An alternative evaluation approach that is suitable combined with that in this report is to use the posts as references when computing the metrics. The posts contain all information, and therefore mitigate the effect of lower metrics caused by preserving information. Computing the metrics using the posts as references would inevitably give lower recall metrics, as the summaries are supposed to be shorter than the posts. However, precision metrics have the potential to be a more accurate representation of quality when using posts as references.

A second limitation of the dataset is that limited information is provided regarding human preference. More precisely, the dataset includes information about which summary is preferred as judged by human labelers, but not by how much. Having multiple preferred-rejected pairs with low summary quality discrepancy could confuse the reward model, causing it to be unstable. To mitigate this problem, the dataset can be filtered so that only preferred-rejected pairs with large differences are included. One way to achieve this is to use another version of the *summarize from feedback* dataset, where the summaries are scored rather than ranked. In this version, the summaries are scored based on four different aspects which are used to give an overall score to each summary. These scores enables choosing a smaller subset of representative preferred-rejected pairs by filtering out pairs where the differences in scores are low. An alternative way to obtain a smaller RMAI dataset of higher quality is to use the off-the-shelf language model to score the summaries instead of ranking them. Similarly, this approach allows for filtering out pairs for which the differences in scores are low.

The idea of using an off-the-shelf language model to score summaries can be further extended. Rather than generating a preference dataset based on the scores, the scores can be used directly as reward signals in the RLAIIF approach. The resulting approach is called direct-RLAIIF and offers several advantages compared to RLAIIF [6]. First, it completely surpasses the reward modeling stage, and by extent the preference dataset generation. Secondly, it avoids the out-of-distribution problem that an ordinary reward model faces during the reinforcement learning stage of RLHF. More precisely, during the reinforcement learning stage, summaries generated by the RL policy are presented to the reward model. As the RL policy is trained, these summaries become less similar to those used to train the reward model (usually the reward model is trained on summaries generated by the SFT model). The corresponding approach used to avoid the out-of-distribution problem when training RLHF is called iterative RLHF. Iterative RLHF updates the reward model iteratively together with the RL policy. Although iterative RLHF avoids the out-of-distribution problem, it is a time-consuming process.

Additional proposed approaches to improve the reward modeling stage include centering and gathering an ensemble. Centering involves combining the reward model objective with a penalty that encourages the scalar rewards to be centered around 0, which stabilizes training and allows for better tuning of the KL divergence coefficient. Centering is especially important in the ensemble approach. The ensemble approach involves using multiple reward models and an aggregation function, such as mean or median, to obtain the ensemble reward. The approach comes with multiple training choices that yield different results, but in general the ensemble increases the robustness of the scores, which mitigates the risk of

reward hacking [37]. Similarly, an ensemble of off-the-shelf language models can be used to rank the summaries in RLAIIF, making the AI feedback more robust.

Additional limitations in this work come from the setting selection approach. The first limitation is that small experiments, which are used to evaluate performances, might not accurately represent a full experiment. In other words, a model trained on a small subset of a dataset might have other optimal settings than a model trained on the entire dataset. The second limitation is that the greedy setting selection approach does not consider how the hyperparameters affect each other. In other words, the optimal setting for one hyperparameter might change with the setting for another hyperparameter. The most notable limitation caused by this setting selection approach affects the reward model. During the setting selection experiments for the reward model, it seemed like a batch size of 8 was optimal, as those learning curves showed clear signs of over-fitting early in the training. However, these experiments were performed on 4 GPUs without realizing that the actual batch size in each optimization step was the batch size multiplied by the number of GPUs [36]. Therefore, the actual optimal batch size found during the setting selection experiments was 32. Unfortunately, the final experiment was performed on a single GPU without accounting for the effect of number of GPUs. As a result, the final experiment used a batch size of 8 instead of the optimal found value of 32. In addition, the fast over-fitting shown in the setting selection experiments suggested that a patience equal to 10 was more than sufficient. However, in the final experiment, a higher patience would be appropriate.

Another limitation caused by the setting selection approach that is most visible in the reward modeling stages is that no hyperparameter tuning was performed for the AI versions of each stage. In general, the settings seem to be suitable for the AI version of each stage, but in the RMAI stage it is clear that a higher patience should have been used. In addition, the setting selection approach seeks to find the optimal model in each stage, without considering how the other models in the approaches are affected. In other words, the overall approach is considered a greedy model selection approach. Intuitively, it sounds like a reasonable approach, but a better one is to consider how the full RLHF and DPO approaches are affected by the settings in each model. For example, [4] found that overfitting the SFT model improves their InstructGPT model.

Another limitation with the setting selection approach is that reward hacking was not as obvious in the small RL experiments. Because of this, the chosen settings are not as restrictive as they probably would need to be to avoid reward hacking. To force smaller and safer updates, the learning rate and clip range can be decreased. In addition, changes to the policy can be more penalized by increasing the KL divergence coefficient. The impact of choosing such restrictive settings is evaluated by conducting an experiment. The restrictive settings used for the experiment are given in Appendix C.2, while the evaluation metrics obtained for the experiment are given in Appendix C.3. The experiment shows that more restrictive settings are advantageous. However, even with the restrictive settings, the RL policy is not comparable to the other models trained in this work. Despite poor performance, the experiment suggests that better performance is expected from RLHF and RLAIIF when combining restrictive settings with the other improvements proposed in this chapter.

Chapter 6

Conclusions

In conclusion, the work demonstrates the unstable nature of RLHF, which causes the approach to generate gibberish summaries. In contrast, DPO proves to be a stable approach that achieves superior performance compared to RLHF as judged by learning curves, ROUGE, BERTScore and a subset of generated summaries. Despite the demonstrated poor performance of RLHF, the approach is expected to have greater potential than the DPO approach due to the flexibility it offers by having multiple stages with corresponding objectives. In other words, the DPO approach offers stability, while the RLHF approach offers flexibility. In addition, the results suggest that the approaches are robust to different sources of feedback, enabling efficient feedback generation without loss of quality.

Bibliography

- [1] Huseyn Jabbarov, 2025, “Text Summarization Approaches for Domain-Specific Content”. Published as a conference paper for the 10th *International Scientific Conference*. Available: https://www.researchgate.net/profile/Kristine-Omadze-2/publication/392796025_Georgian_Economy_Transition_Financial_Crisis_and_crossroad_of_1999/links/68530d4cdf3fa4286a416703/Georgian-Economy-Transition-Financial-Crisis-and-crossroad-of-1999.pdf.
- [2] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Lukasz Kaiser, Illia Polosukhin, 2017, “Attention Is All You Need”. Published as a conference paper for 31st Conference on *Neural Information Processing Systems*. Available: https://proceedings.neurips.cc/paper_files/paper/2017/file/3f5ee243547dee91fbd053c1c4a845aa-Paper.pdf.
- [3] Nisan Stiennon, Long Ouyang, Jeff Wu, Daniel M. Ziegler, Ryan Lowe, Chelsea Voss, Paul Christiano, et al., 2020, “Learning to summarize from human feedback”. Published as a conference paper for 34th Conference on *Neural Information Processing Systems*. Available: <https://proceedings.neurips.cc/paper/2020/file/1f89885d556929e98d3ef9b86448f951-Paper.pdf>.
- [4] Long Ouyang, Jeff Wu, Xu Jiang, Diogo Almeida, Carroll L. Wainwright, Pamela Mishkin, Paul Christiano, Jan Leike, Ryan Lowe, et al., 2022, “Training language models to follow instructions with human feedback”. Published as a conference paper for 36th Conference on *Neural Information Processing Systems*. Available: https://proceedings.neurips.cc/paper_files/paper/2022/file/b1efde53be364a73914f58805a001731-Paper-Conference.pdf.
- [5] Rafael Rafailov, Archit Sharma, Eric Mitchell, et al., 2023, “Direct Preference Optimization: Your Language Model is Secretly a Reward Model”. Published as a conference paper for 37th Conference on *Neural Information Processing Systems*. Available: https://proceedings.neurips.cc/paper_files/paper/2023/file/a85b405ed65c6477a4fe8302b5e06ce7-Paper-Conference.pdf.
- [6] Harrison Lee, Samrat Phatale, Hassan Mansoor, Thomas Mesnard, Johan Ferret, Kellie Lu, Colton Bishop, Ethan Hall, Victor Carbune, Abhinav Rastogi, Sushant Prakash, 2024, “RLAIF vs. RLHF: Scaling Reinforcement Learning from Human Feedback with AI Feedback”. Published as a confer-

- ence paper for 41st *International Conference on Machine Learning*. Available: <https://proceedings.mlr.press/v235/lee24t.html>.
- [7] Eram Munawwar, 2022, “A comprehensive guide to subword tokenisers”, *Towards data science*. Available: <https://towardsdatascience.com/a-comprehensive-guide-to-subword-tokenisers-4bbd3bad9a7c/>.
 - [8] Ketan Doshi, 2021, “Transformers Explained Visually (Part 2): How it works, step-by-step”, *Towards data science*. Available: <https://towardsdatascience.com/transformers-explained-visually-part-2-how-it-works-step-by-step-b49fa4a64f34/>.
 - [9] Aston Zhang, Zachary C. Lipton, Mu Li, Alexander J. Smola, et al., 2023, “Attention Mechanisms and Transformers” in *Dive into Deep Learning*, Cambridge University Press. Available: <https://d2l.ai/>.
 - [10] Ketan Doshi, 2021, “Transformers Explained Visually (Part 3): Multi-head Attention, deep dive”, *Towards data science*. Available: <https://towardsdatascience.com/transformers-explained-visually-part-3-multi-head-attention-deep-dive-1c1ff1024853/>.
 - [11] Maria Prokofieva, 2024, “Attention in Transformers: residual connection layer, a shortcut that makes it work”, *Medium*. Available: <https://medium.com/@mariaprokofieva/attention-in-transformers-residual-connection-layer-a-shortcut-that-makes-it-work-165b52566167>.
 - [12] Sachin Soni, 2024, “Layer Normalization in Transformer”, *Medium*. Available: <https://medium.com/@sachinsoni600517/layer-normalization-in-transformer-1a2efbff8b85>.
 - [13] Kyrylo Dekanenko, 2025, “Demystifying Temperature in Language Models: The Softmax Secret”, *Medium*. Available: <https://medium.com/@kyrylldekanenko/demystifying-temperature-in-language-models-the-softmax-secret-db4c8b02425d>.
 - [14] Richard S. Sutton, Andrew G. Barto, 2018, *Reinforcement Learning: An Introduction*, MIT Press.
 - [15] Nathan Lambert, 2026, “Reinforcement Learning from Human Feedback”. Available: <https://rlhfbook.com/>.
 - [16] Aayush Garg, 2025, “Deriving the PPO Loss from First Principles”, *Hugging face*. Available: <https://huggingface.co/blog/garg-aayush/ppo-from-first-principle>.
 - [17] Wouter van Heeswijk, 2022, “Natural Policy Gradients In Reinforcement Learning Explained”, *Towards data science*. Available: <https://towardsdatascience.com/natural-policy-gradients-in-reinforcement-learning-explained-2265864cf43c/>.
 - [18] Katerina Fragkiadaki, “Natural Policy Gradients, TRPO, PPO”, *Carnegie Mellon University*. Available: https://www.andrew.cmu.edu/course/10-703/slides/Lecture_NaturalPolicyGradientsTRPOppo.pdf.

- [19] John Schulman, Sergey Levine, Philipp Moritz, Michael Jordan, Pieter Abbeel, 2015, “Trust Region Policy Optimization”. Published as a conference paper for 31st *International Conference on Machine Learning*. Available: <http://proceedings.mlr.press/v37/schulman15.pdf>.
- [20] John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, Oleg Klimov, 2017, “Proximal Policy Optimization Algorithms”, arXiv:1707.06347. Available: <https://arxiv.org/pdf/1707.06347>.
- [21] John Schulman, Philipp Moritz, Sergey Levine, Michael I. Jordan, Pieter Abbeel, 2016, “High-Dimensional Continuous Control Using Generalized Advantage Estimation”. Published as a conference paper for *International Conference on Learning Representations*. Available: <https://arxiv.org/pdf/1506.02438>.
- [22] Nathan Lambert, Louis Castricato, Leandro Von Werra, Alex Havrilla, 2022, “Illustrating Reinforcement Learning from Human Feedback (RLHF)”, *Hugging face*. Available: <https://huggingface.co/blog/rlhf>.
- [23] Michael Brenndoerfer, 2025, “Reward Modeling: Building Preference Predictors for RLHF”, *Micheal Brenndoerfer*. Available: <https://mbrenndoerfer.com/writing/reward-modeling-rlhf-architecture-training>.
- [24] John Schulman, 2020, “Approximating KL Divergence”, *Joschu*. Available: <http://joschu.net/blog/kl-approx.html>.
- [25] Edward Hu, Yelong Shen, et al., 2022, “LoRA: Low-Rank Adaption of Large Language Models”. Published as a conference paper for *International Conference on Learning Representations*. Available: <https://iclr.cc/virtual/2022/poster/6319>.
- [26] Fabio Chiusano, 2022, “Two minutes NLP — Learn the ROUGE metric by examples”, *Medium*. Available: <https://medium.com/nlplanet/two-minutes-nlp-learn-the-rouge-metric-by-examples-f179cc285499>.
- [27] Tianyi Zhang, Varsha Kishore, Felix Wu, et al., 2020, “BERTScore: Evaluating Text Generation With BERT”. Published as a conference paper for *International Conference on Learning Representations*. Available: <https://openreview.net/pdf?id=SkeHuCVFDr>.
- [28] Michael Völske, Martin Potthast, Shahbaz Syed, Benno Stein, 2017, “TL;DR: Mining Reddit to Learn Automatic Summarization”, *Aclanthology*. Available: <https://aclanthology.org/W17-4508.pdf>.
- [29] Nisan Stiennon, Long Ouyang, Jeff Wu, Daniel M. Ziegler, Ryan Lowe, Chelsea Voss, Paul Christiano, et al., 2022, “summarize_from_feedback”, *Hugging face*. Available: https://huggingface.co/datasets/openai/summarize_from_feedback.
- [30] Qwen team, 2025, “Qwen3 Technical Report”, arXiv:2505.09388. Available: <https://arxiv.org/pdf/2505.09388>.
- [31] Al’amin ai, 2025, “Building Qwen3 from Scratch: A Deep Dive into Modern Transformer Architecture”, *Medium*. Available: <https://medium.com/@al>

- [aminibrahim433/building-qwen3-from-scratch-a-deep-dive-into-modern-transformer-architecture-0ff152aa222a](#).
- [32] “Qwen3-4B-Instruct-2507: A Compact Powerhouse for Instruction-Following AI”, *Qwen3lm*. Available: <https://qwen3lm.com/qwen3-4b-instruct-2507/>.
- [33] Qwen team, “Qwen3-30B-A3B-Instruct-2507”, *Hugging face*. Available: <https://huggingface.co/Qwen/Qwen3-30B-A3B-Instruct-2507#qwen3-30b-a3b-instruct-2507>.
- [34] Ian Goodfellow, Yoshua Bengio, Aaron Courville, 2016, *Deep Learning*, MIT Press.
- [35] Cyborg, 2024, “What is Early Stopping in Deep Learning?”, *Medium*. Available: <https://cyborgcodes.medium.com/what-is-early-stopping-in-deep-learning-eeb1e710a3cf>.
- [36] Huggingface team, “Comparing performance across distributed setups”, *Hugging face*. Available: https://huggingface.co/docs/accelerate/en/concept_guides/performance.
- [37] Jacob Eisenstein, Chirag Nagpal, Alekh Agarwal, Jonathan Berant, et al. 2024, “Helping or Herding? Reward Model Ensembles Mitigate but do not Eliminate Reward Hacking”. Published as a conference paper for *Conference on Language Modeling*. Available: <https://openreview.net/pdf?id=5u1GpUkKtG>.

Appendix A

Mathematical Derivations

The first part of the appendices is dedicated to mathematical derivations. The following sections show how to obtain the approximated KL divergence used in NPG, the identity and importance sampling ratio used in TRPO, and the generalized advantage estimator used in PPO. Lastly, how to rewrite the RLHF objective and the optimal RL policy to obtain the DPO objective is given in the last sections.

A.1 Kullback-Leiber Divergence

As shown by [18], the approximated KL divergence given in Equations (2.30) - (2.32) can be further rewritten. More precisely, the second term can be written as

$$\nabla_{\phi} D_{KL}(\pi_{\phi_{old}} || \pi_{\phi})|_{\phi=\phi_{old}} = \nabla_{\phi} \mathbb{E}_{\pi_{\phi_{old}}} \left[\log \frac{\pi_{\phi_{old}}}{\pi_{\phi}} \right] \quad (\text{A.1})$$

$$= -\nabla_{\phi} \mathbb{E}_{\pi_{\phi_{old}}} [\log \pi_{\phi} | \phi=\phi_{old}] \quad (\text{A.2})$$

$$= -\mathbb{E}_{\pi_{\phi_{old}}} [\nabla_{\phi} \log \pi_{\phi} | \phi=\phi_{old}] \quad (\text{A.3})$$

$$= -\mathbb{E}_{\pi_{\phi_{old}}} \left[\frac{\nabla_{\phi} \pi_{\phi}}{\pi_{\phi_{old}}} | \phi=\phi_{old} \right] \quad (\text{A.4})$$

$$= \sum \pi_{\phi_{old}} \frac{\nabla_{\phi} \pi_{\phi}}{\pi_{\phi_{old}}} | \phi=\phi_{old} \quad (\text{A.5})$$

$$= \sum \nabla_{\phi} \pi_{\phi} | \phi=\phi_{old} \quad (\text{A.6})$$

$$= \nabla_{\phi} \sum \pi_{\phi} | \phi=\phi_{old} \quad (\text{A.7})$$

$$= 0. \quad (\text{A.8})$$

Similarly, the third term can be written as

$$\nabla_\phi^2 D_{KL}(\pi_{\phi_{old}} || \pi_\phi) |_{\phi=\phi_{old}} = -\mathbb{E}_{\pi_{\phi_{old}}} [\nabla_\phi^2 \log \pi_\phi |_{\phi=\phi_{old}}] \quad (\text{A.9})$$

$$= -\mathbb{E}_{\pi_{\phi_{old}}} \left[\nabla_\phi \left(\frac{\nabla_\phi \pi_\phi}{\pi_\phi} \right) |_{\phi=\phi_{old}} \right] \quad (\text{A.10})$$

$$= -\mathbb{E}_{\pi_{\phi_{old}}} \left[\frac{\nabla_\phi^2 \pi_\phi \pi_\phi - \nabla_\phi \pi_\phi \nabla_\phi \pi_\phi^T}{\pi_\phi^2} |_{\phi=\phi_{old}} \right] \quad (\text{A.11})$$

$$= -\mathbb{E}_{\pi_{\phi_{old}}} \left[\frac{\nabla_\phi^2 \pi_\phi |_{\phi=\phi_{old}}}{\pi_{\phi_{old}}} \right] \quad (\text{A.12})$$

$$+ \mathbb{E}_{\pi_{\phi_{old}}} [\nabla_\phi \log \pi_\phi \nabla_\phi \log \pi_\phi^T |_{\phi=\phi_{old}}] \quad (\text{A.13})$$

$$= \mathbb{E}_{\pi_{\phi_{old}}} [\nabla_\phi \log \pi_\phi \nabla_\phi \log \pi_\phi^T |_{\phi=\phi_{old}}]. \quad (\text{A.14})$$

A.2 Identity

The identity on which TRPO is based is derived by [19] as follows

$$\mathbb{E}_{\tau \sim \pi_\phi} \left[\sum_{t=0}^{\infty} \gamma^t A(s_t, a_t) \right] = \mathbb{E}_{\tau \sim \pi_\phi} \left[\sum_{t=0}^{\infty} \gamma^t (q_{\pi_{\phi_{old}}}(s_t, a_t) - v_{\pi_{\phi_{old}}}(s_t)) \right] \quad (\text{A.15})$$

$$= \mathbb{E}_{\tau \sim \pi_\phi} \left[\sum_{t=0}^{\infty} \gamma^t (r_t + \gamma v_{\pi_{\phi_{old}}}(s_{t+1}) - v_{\pi_{\phi_{old}}}(s_t)) \right] \quad (\text{A.16})$$

$$= \mathbb{E}_{\tau \sim \pi_\phi} \left[-v_{\pi_{\phi_{old}}}(s_0) + \sum_{t=0}^{\infty} \gamma^t r_t \right] \quad (\text{A.17})$$

$$= \mathbb{E}_{s_0 \sim p(s_0)} [v_{\pi_{\phi_{old}}}(s_0)] + \mathbb{E}_{\tau \sim \pi_\phi} \left[\sum_{t=0}^{\infty} \gamma^t r_t \right] \quad (\text{A.18})$$

$$= \mathcal{L}(\phi) - \mathcal{L}(\phi_{old}). \quad (\text{A.19})$$

A.3 Importance Sampling

Using importance sampling, the policy gradient used in the REINFORCE with baseline algorithm can be written

$$\nabla_\phi \mathcal{L}(\phi) = \mathbb{E}_t [A(s_t, a_t) \nabla_\phi \log \pi_\phi(a_t | s_t)] \quad (\text{A.20})$$

$$= \sum A(s_t, a_t) \pi_\phi(a_t | s_t) \nabla_\phi \log \pi_\phi(a_t | s_t) \quad (\text{A.21})$$

$$= \sum A(s_t, a_t) \pi_{\phi_{old}}(a_t | s_t) \frac{\pi_\phi(a_t | s_t)}{\pi_{\phi_{old}}(a_t | s_t)} \nabla_\phi \log \pi_\phi(a_t | s_t) \quad (\text{A.22})$$

$$= \mathbb{E}_t \left[A(s_t, a_t) \frac{\pi_\phi(a_t | s_t)}{\pi_{\phi_{old}}(a_t | s_t)} \nabla_\phi \log \pi_\phi(a_t | s_t) \right] \quad (\text{A.23})$$

$$= \mathbb{E}_t \left[A(s_t, a_t) \frac{\pi_\phi(a_t | s_t)}{\pi_{\phi_{old}}(a_t | s_t)} \frac{\nabla_\phi \pi_\phi(a_t | s_t)}{\pi_\phi(a_t | s_t)} \right] \quad (\text{A.24})$$

$$= \mathbb{E}_t \left[A(s_t, a_t) \frac{\nabla_\phi \pi_\phi(a_t | s_t)}{\pi_{\phi_{old}}(a_t | s_t)} \right]. \quad (\text{A.25})$$

A.4 Generalized Advantage Estimator

GAE is derived from the discounted sums of TD errors. The derivation steps, as shown by [21], are as follows

$$\hat{A}(s_t, a) = (1 - \lambda)(\hat{A}^{(1)}(s_t, a) + \lambda \hat{A}^{(2)}(s_t, a) + \dots) \quad (\text{A.26})$$

$$= (1 - \lambda)(\delta_t + \lambda(\delta_t + \gamma \delta_{t+1}) + \dots) \quad (\text{A.27})$$

$$= (1 - \lambda)(\delta_t(1 + \lambda + \dots) + \gamma \delta_{t+1}(\lambda + \lambda^2 + \dots) + \dots) \quad (\text{A.28})$$

$$= (1 - \lambda) \left(\delta_t \left(\frac{1}{1 - \lambda} \right) + \gamma \delta_{t+1} \left(\frac{\lambda}{1 - \lambda} \right) + \dots \right) \quad (\text{A.29})$$

$$= \sum_{i=0}^{\infty} (\gamma \lambda)^i \delta_{t+i}. \quad (\text{A.30})$$

A.5 Rewriting Objective

As shown by [5], the RLHF objective can be rewritten as follows

$$\max \mathcal{L}(\phi) = \max_{\pi_{\phi}^{RL}} \mathbb{E}_{x \sim \mathcal{D}, y \sim \pi_{\phi}^{RL}} [r_{\theta}(y, x) - \beta D_{KL}(\pi_{\phi}^{RL} || \pi^{SFT})] \quad (\text{A.31})$$

$$= \max_{\pi_{\phi}^{RL}} \mathbb{E}_{x \sim \mathcal{D}, y \sim \pi_{\phi}^{RL}} \left[r_{\theta}(y, x) - \beta \log \frac{\pi_{\phi}^{RL}}{\pi^{SFT}} \right] \quad (\text{A.32})$$

$$= \min_{\pi_{\phi}^{RL}} \mathbb{E}_{x \sim \mathcal{D}, y \sim \pi_{\phi}^{RL}} \left[\log \frac{\pi_{\phi}^{RL}}{\pi^{SFT}} - \frac{1}{\beta} r_{\theta}(y, x) \right] \quad (\text{A.33})$$

$$= \min_{\pi_{\phi}^{RL}} \mathbb{E}_{x \sim \mathcal{D}, y \sim \pi_{\phi}^{RL}} \left[\log \frac{\pi_{\phi}^{RL}}{\pi^{SFT}} - \log \exp \frac{1}{\beta} r_{\theta}(y, x) \right] \quad (\text{A.34})$$

$$= \min_{\pi_{\phi}^{RL}} \mathbb{E}_{x \sim \mathcal{D}, y \sim \pi_{\phi}^{RL}} \left[\log \frac{\pi_{\phi}^{RL}}{\pi^{SFT} \exp(\frac{1}{\beta} r_{\theta}(y, x))} \right] \quad (\text{A.35})$$

$$= \min_{\pi_{\phi}^{RL}} \mathbb{E}_{x \sim \mathcal{D}, y \sim \pi_{\phi}^{RL}} \left[\log \frac{\pi_{\phi}^{RL}}{\frac{1}{Z(x)} \pi^{SFT} \exp(\frac{1}{\beta} r_{\theta}(y, x))} - \log Z(x) \right]. \quad (\text{A.36})$$

A.6 Rewriting Policy

As shown by [5], the optimal RL policy can be rewritten as follows

$$\pi_{\phi}^{RL*} = \frac{1}{Z(x)} \pi^{SFT} \exp\left(\frac{1}{\beta} r_{\theta}(y, x)\right) \quad (\text{A.37})$$

$$\Leftrightarrow \frac{\pi_{\phi}^{RL*}}{\pi^{SFT}} Z(x) = \exp\left(\frac{1}{\beta} r_{\theta}(y, x)\right) \quad (\text{A.38})$$

$$\Leftrightarrow \log \frac{\pi_{\phi}^{RL*}}{\pi^{SFT}} Z(x) = \frac{1}{\beta} r_{\theta}(y, x) \quad (\text{A.39})$$

$$\Leftrightarrow \beta \log \frac{\pi_{\phi}^{RL*}}{\pi^{SFT}} Z(x) = r_{\theta}(y, x) \quad (\text{A.40})$$

$$\Leftrightarrow r_{\theta}(y, x) = \beta \log \frac{\pi_{\phi}^{RL*}}{\pi^{SFT}} + \beta \log Z(x). \quad (\text{A.41})$$

Appendix B

Figures

The second part of the appendices is dedicated to figures. The following sections show the subreddits and their corresponding distributions of the RM training and validation datasets. In addition, a couple of learning curves are presented for the RL and RLAI training stages in the following section.

B.1 Subreddit Distribution

The subreddits and their corresponding distributions of the RM datasets are visualized in Figure B.1.

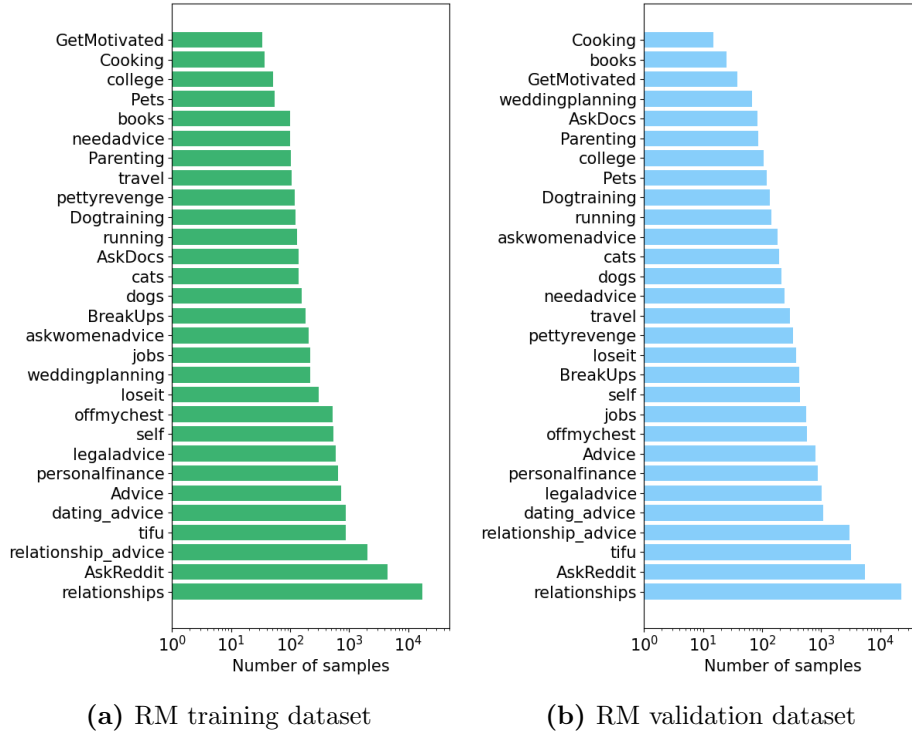


Figure B.1: Subreddits and their corresponding distribution. The RM datasets contain Reddit posts from several different subreddits. The subreddits and their corresponding distributions in the RM training and validation datasets are visualized on a logarithmic scale in Figures B.1a and B.1b, respectively.

B.2 Learning Curves

During RL policy training, the entropy of the policy and the reward function are monitored, as visualized in Figure B.2. The metrics are plotted against the number of optimization steps, where each step considers a batch size of 4.

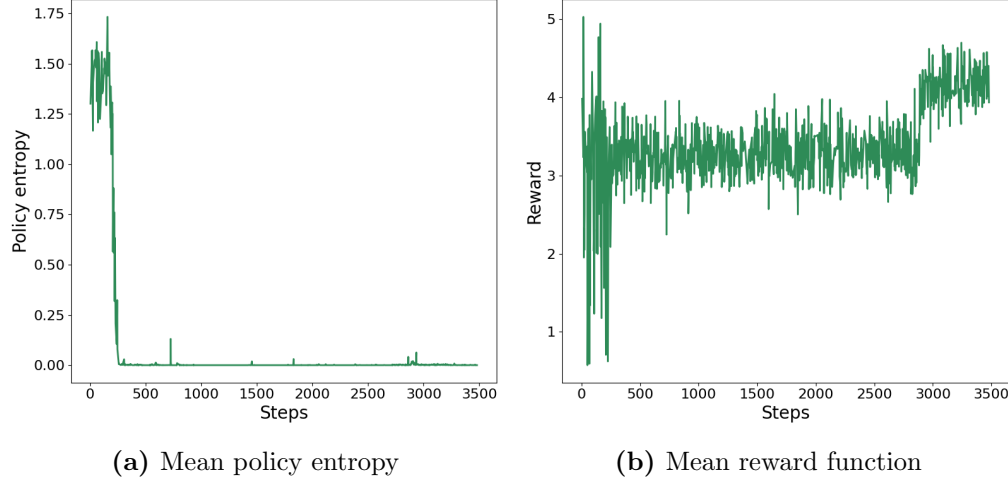


Figure B.2: Learning curves for RL policy. The plotted metrics are the policy entropy and the reward function. The metrics are plotted against the number of optimization steps and averaged over the samples in the corresponding step.

In addition, the same metrics are monitored during the RLAI training. The metrics are visualized against the number of optimization steps in Figure B.3.

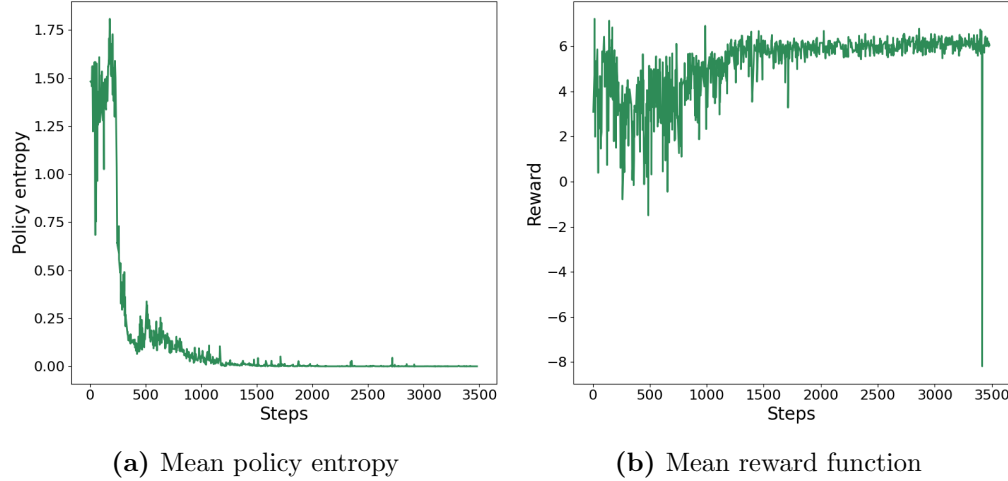


Figure B.3: Learning curves for RLAI policy. The plotted metrics are the policy entropy and the reward function. The metrics are plotted against the number of optimization steps and averaged over the samples in the corresponding step.

Appendix C

Tables

The third and final part of the appendices is dedicated to tables. The first section presents an additional example Reddit post with corresponding reference summary and generated summaries. The following section presents the restrictive hyperparameters and choices made for the RL experiment discussed in Chapter 5. The final section presents the evaluation metrics obtained by the RL experiment with more restrictive settings.

C.1 Generations

Table C.1 presents an example Reddit post, the corresponding reference summary and the summaries generated by the SFT, SFTAI, DPO and DPOAI models, respectively.

Reddit post	I am a college freshman and I’m trying to get my confidence back after having it destroyed. A lot of people see me as being shy, but that’s not technically true. As a kid I was a natural born extrovert, the rare class clown who also got straight A’s. I was born with that kind of lion-esque confidence that some people have. In middle school I had that confidence beaten out of me, either with fists or with words. By high school I trusted no one and assumed that everyone was out to get me. While I can now see how stupid that was, I couldn’t then. Hating everyone didn’t get me many friends, or girlfriends. After I lost my virginity I got a little of that confidence back and started getting with girls pretty regularly, but that didn’t last long. After the first ugly breakup with a girl I went right back into a shell. So now I’m a freshman at a small community college and it’s very hard to meet people. I have a few close friends who I rarely see, I’m in a band, and I can feel that old confidence coming back, but it’s coming back very slowly. I would appreciate any advice on speeding up the “recharging process” and also on meeting new people.
Reference summary	Born an extrovert. Bullied into being an introvert. Shit sucked, but I got over it. Gaining confidence back, slowly though. All advice on that and meeting new people appreciated.
SFT generation	I’m a shy college freshman who is trying to get my confidence back.
SFTAI generation	I’m a shy freshman who’s trying to get my confidence back.
DPO generation	I lost my confidence in middle school and it’s been hard to get it back. I’m a freshman at a small college and I’m having trouble meeting new people. I would appreciate any advice on speeding up the “recharging process” and also on meeting new people.
DPOAI generation	I’m a college freshman trying to get my confidence back after it was destroyed in middle/high school, and I would appreciate any advice on speeding up the “recharging process” and also on meeting new people.

Table C.1: Example post, reference summary and generated summaries. The Reddit post and reference summary are taken from the test dataset. The generated summaries are generated by the SFT, SFTAI, DPO and DPOAI models.

C.2 Hyperparameter Settings

The restrictive hyperparameters and choices made for the RL policy experiment discussed in Section 5.5 are summarized in Table C.2. The table also includes the hyperparameters and choices made for the RL policy, whose results are presented in Section 4.3 for comparison.

Parameter	RL	RL _{restrictive}
Rank r	16	16
Scaling factor η	32	32
Coefficient β_1	0.9	0.9
Coefficient β_2	0.999	0.999
Learning rate α	$5 \cdot 10^{-5}$	$2 \cdot 10^{-5}$
Learning rate type	Linear	Linear
Warmup ratio	0.1	0.1
Discount factor γ	1.0	1.0
Clip range ϵ	0.2	0.1
Exponential weight discount λ	0.95	0.95
Coefficient c_1	0.1	0.1
KL divergence estimator	Third	Third
Coefficient β	0.05	0.1

Table C.2: Restrictive hyperparameters and choices. The restrictive hyperparameters and choices used to train the RL policy discussed in Section 5.5. The hyperparameters and choices used to train the RL policy, whose results are presented in Section 4.3 are included for comparison.

C.3 Performance

The evaluation metrics obtained for the experiment with restrictive settings are presented in Table C.3.

Model	F1 _{ROUGE-1}	F1 _{ROUGE-2}	F1 _{ROUGE-L}	P _{BERT}	R _{BERT}	F1 _{BERT}
RL	0.200	0.089	0.155	0.168	0.460	0.303
RLAI	0.212	0.091	0.157	0.168	0.450	0.299

Table C.3: Performance of RL and RLAI policies with restrictive settings. Performance is assessed using ROUGE and BERTScore. The metrics are computed to compare the summaries generated by the RL and RLAI policies with the reference summaries in the test dataset.