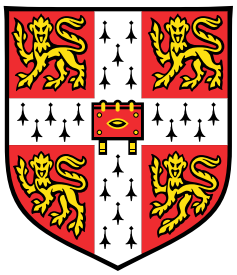
Eliciting Latent Knowledge from Language Reward Models



Augustas Macijauskas

Department of Engineering
University of Cambridge

This dissertation is submitted for the degree of
Master of Philosophy in Machine Learning and Machine Intelligence

Robinson College October 2023

I would like to dedicate this thesis to my loving parents.

Declaration

I, Augustas Macijauskas of Robinson College, being a candidate for the MPhil in Machine Learning and Machine Intelligence, hereby declare that this report and the work described in it are my own work, unaided except as may be specified below, and that the report does not contain material that has already been used to any substantial extent for a comparable purpose. The code[1] for this project has been developed by me in Python, mostly using the `PyTorch`, `transformers` and `datasets` libraries, except for the following cases:

1. The `elk` library was used to train and evaluate DLK models in Chapters 3 and 4. Minor bug fixes were contributed, as well as some custom prompt templates were introduced, but the code was used unchanged otherwise. This final version of the code was used.

2. The Transformer Reinforcement Learning (`trl`) library without modifications was utilized to perform reinforcement learning fine-tuning experiments in Chapter 4.

3. The Open LLM Leaderboard and the `big-refactor` versions of the Language Model Evaluation Harness library were utilized to evaluate the models in Chapter 4. Custom prompts were introduced, but no other modifications were made.

The word count for this thesis is 14997 words including appendices.

Augustas Macijauskas
October 2023

[1] https://github.com/AugustasMacijauskas/mlmi-thesis

Acknowledgements

With profound appreciation and heartfelt gratitude, I acknowledge the individuals and institutions that have both directly and indirectly guided, supported, and empowered me throughout this academic journey.

First and foremost, I extend my sincere thanks to my thesis advisors, Samuel Albanie and Herbie Bradley. Your guidance has been instrumental in my learning process, enabling me to overcome even the most formidable challenges. I am grateful for your patience, the abundance of ideas and resources you have shared, and for teaching me the essentials of thinking and acting as a researcher. The opportunity to work with both of you has been invaluable.

I would also like to express my appreciation to the members of the EleutherAI and CarperAI communities on their respective Discord servers. Your assistance with understanding and utilizing your libraries, as well as your insights into theoretical and conceptual ideas, have greatly facilitated my work on this thesis.

To my peers in the MPhil MLMI 2023 cohort: spending this year with you has been both rewarding and enjoyable. I take pride in being a part of such a strong and intellectual community, and I treasure the shared experiences of our collective journey.

To my friends, both longstanding and newly made in Cambridge: your support and companionship have genuinely enriched my time here. I look forward to expressing my personal thanks to some of you when we meet again.

To my family, particularly my parents and brother: your unwavering support, whether standing by me at my best or cheering me on at my worst, has been a cornerstone of my success. I owe much of my progress to your encouragement and assistance.

Lastly, I extend my thanks to StabilityAI for generously providing the computing resources necessary for this project.

Abstract

Large language models (LLMs) have revolutionized the field of *natural language processing* (NLP) with their unparalleled ability to generate human-like text. These black box models are trained to predict the next *token* in a sequence of text sourced from the internet as accurately as possible [37, 38], which makes then poorly aligned with human intent and prone to producing hallucinated outputs [2, 23, 29]. *Reinforcement learning from human feedback* (RLHF), the current state-of-the-art method to align LLMs, excels at improving their *helpfulness* and *harmlessness*, but not *honesty* [35, 2, 3, 51]. To tackle this, we propose a framework for building reward models that promote "truthfulness" and demonstrate that they can be effectively used to execute *reinforcement learning* (RL) fine-tuning of a pre-trained LLM.

Specifically, we utilize techniques from a subfield of machine learning model interpretability known as *discovering latent knowledge* (DLK), which aims to discern what the models' internal *"beliefs"* are irrespective of their outputs [11, 13]. Recent advancements in the domain have proposed both supervised and unsupervised lightweight *probes* that can accurately execute the task using solely the activations of a language model [11, 5]. Our method proposes how to integrate these probes with a pre-trained language model to build reward models for "honesty".

We demonstrate that using these newly built reward models to fine-tune pre-trained LLMs improves their "truthfulness" on a target dataset. Furthermore, we observe that only a reasonably small amount of data is required, that there is no degradation in the language model's performance on more general NLP tasks, and that the model's "honesty" improves in a more general sense than we initially define. These findings indicate that our method is an exciting step towards achieving scalable language model alignment.

Table of contents

List of figures

List of tables

Chapter 1

Introduction

Large language models (LLMs) are powerful models used for *natural language processing* (NLP). They are black boxes trained on large quantities of internet data to produce human-like text [37, 38]. An issue arising from this training objective is that LLMs may not be aligned with human intent and struggle with a phenomenon known as *hallucination*, where they generate false or misleading text [23, 29].

The domain of LLM *alignment* is concerned with tackling this and aims to ensure that LLMs behave in line with human values and do not pursue unintended or undesirable goals [33]. *Reinforcement learning from human* (RLHF) is the most successful current method that is capable of improving the *helpfulness* and *harmlessness* of LLMs [2, 35, 3] and has facilitated the creation of tolls like ChatGPT [9, 34]. However, a limitation of this technique is that it struggles to meaningfully improve the *honesty* of LLMs [3]. Furthermore, RLHF requires human labellers to rank large amounts of LLM outputs, which is both a costly and time-consuming process.

The observation that prompting can guide LLMs towards producing accurate and truthful responses [41, 57] suggests that they may possess some internal representations of the truthfulness of the data they are processing irrespective of what their outputs are. These representations are referred to as *latent knowledge*, and extracting them is an unsolved problem in machine learning known as *eliciting latent knowledge* (ELK) [13]. An exciting piece of work towards achieving this goal is *Contrast-Consistent Search* (CCS), an unsupervised method capable of predicting an LLM's response to a binary question solely from its unlabelled activations [11]. The technique does not rely on the model's outputs, which makes it effective even if the model hallucinates.

The aim of this research is to use methods that *discover latent knowledge* (DLK), such as CCS, to address the limitations of RLHF. Specifically, we investigate ways to use DLK techniques to build reward models that promote "honesty". We first try to answer whether

this is possible given the capabilities of the current DLK methods and then explore whether these reward models can be successfully used to fine-tune pre-trained LLMs to be more "truthful" using *reinforcement learning* (RL). The significance of this research is that it would contribute towards improving the honesty of LLMs, which is an important unsolved problem within LLM alignment. Furthermore, the use of DLK methods would allow achieving this contribution without requiring a large dataset of human preference data.

This thesis will use the terms language model "honesty" and "truthfulness" interchangeably and will only focus on a narrow definition for them, which encompasses a model's capability to accurately respond to questions from selected binary *question-answering* (QA) datasets. This is the definition that we mean when enclosing the terms in quotations, while a lack of quotations signifies the more general definition of the truthfulness of a language model as outlined by Askell et al. [2]. Moreover, both unsupervised and supervised DLK methods will be considered when building reward models for "honesty". This means that the requirement for labelled data would not be removed altogether, but rather than requiring a new dataset to be collected, an existing dataset would suffice.

1.1 Contributions

A summary of contributions made by this thesis is presented below. As far as we are aware, all of these are novel contributions.

- Develop a framework that allows turning existing DLK methods into reward models for "honesty". This includes transforming existing datasets to be suitable for reward modelling and RL fine-tuning, integrating both unsupervised and supervised probes with an existing pre-trained language model, as well as investigating the performance, advantages and disadvantages of each method (Chapter 3).
- Show that our reward models can be successfully used to perform RL fine-tuning to improve the "truthfulness" of LLMs. We describe the strategies and regularizations that we utilized to stabilize the fine-tuning process. We observe that our method only requires a small amount of training data, and does not deteriorate the models' abilities on general NLP tasks (Chapter 4).

1.2 Thesis Outline

Below, an outline of the structure of the thesis is presented, along with a short description of each chapter.

Chapter 2 provides an overview of the relevant topics and prior work, beginning with a discussion of language modelling, the transformer architecture, and advanced fine-tuning techniques. This is followed by a summary of state-of-the-art DLK methods. Finally, the topic of using RL to fine-tune pre-trained LLMs is presented.

In Chapter 3, we present our approach that utilizes DLK techniques to build reward models for "honesty". We first elaborate on our design choices and the datasets used, before explaining how we identified the most promising reward models for RL fine-tuning.

Chapter 4 discusses how we used the constructed reward models to perform RL fine-tuning to improve the "truthfulness" of a large language model. We begin by examining a naive approach that proved unsuccessful and then describe the iterative improvements that we made to stabilize the training procedure. The chapter concludes with an evaluation of the fine-tuned models both on our targeted and broader NLP datasets.

Finally, in Chapter 5, we conclude the thesis, discuss the broader impacts and limitations of our work, and outline promising directions for future work.

Chapter 2

Background and Related Work

This chapter offers an overview of the essential concepts and methodologies required to understand the subsequent sections of the thesis, as well as the relevant pieces of prior research. Initially, it describes language modelling, the specific model architectures employed in this project, and cutting-edge fine-tuning techniques. Next, it introduces the concept of *discovering latent knowledge* (DLK) and explores existing methods related to it. Finally, the chapter discusses the *reinforcement learning from human feedback* (RLHF) technique, including its achievements and limitations.

2.1 Language Modelling

In *auto-regressive* (or *causal*) *language modelling* the joint probability of a sequence $\mathbf{s} = (s_1, \ldots, s_T)$ is represented as a product of conditional probabilities [22, 6, 38]:

$$p(\mathbf{s}) = \Pi_{i=1}^{T} p(s_i | s_1, \ldots, s_{i-1})$$

Here, T denotes the length of the sequence. The conditional probabilities express the likelihood of the next element in the sequence given only the current elements and no additional information. Meanwhile, in *sequence-to-sequence* language modelling the joint probability of $\mathbf{s}$ is conditioned on another sequence $\mathbf{e} = (e_1, \ldots, e_L)$ of length L ("e" here stands for "encoded"):

$$p(\mathbf{s}|\mathbf{e}) = \Pi_{i=1}^{T} p(s_i | s_1, \ldots, s_{i-1}, e_1, \ldots, e_L)$$

The role of machine learning is then to approximate these two types of conditional probabilities. The first decomposition corresponds to *decoder-only* language models, while

the second corresponds to *encoder-decoder* language models (hence the use of "e" for "encoded"). In this context, *encoding* refers to the transformation of each element in an input sequence to derive another sequence of the same length. Both model types will be elaborated upon in Section 2.1.1.

Tokenization and vectorization

In the context of modelling natural language, the sequences **s** and **e** are typically sequences of words. For practical reasons, these words are usually broken down into smaller units called *tokens* using a process known as *tokenization* [55]. Various tokenization strategies exist, each characterized by how it segments the sequence into these smaller elements. The resulting set of tokens used by a particular strategy is referred to as the *vocabulary*.

Tokenization strategies can range from splitting on the character level, resulting in a small vocabulary but very lengthy input sequences, to splitting on individual words, yielding shorter sequences but a large and often unwieldy vocabulary. A common compromise is *byte pair encoding* (BPE) [46], which divides the sequence at the subword level, producing vocabularies ranging from 32,000 to 64,000 words [38]. This approach balances the need for manageable vocabulary size with input sequence length.

Since machine learning models require numerical inputs rather than strings, the tokens must be converted into numbers through a process called *vectorization*. This involves creating a lookup table that assigns a unique integer to each token in the vocabulary. The input vector to the language model is then formed by mapping the tokenized sequence to a vector of integers.

Decoding

Decoding is the process of transforming the output probabilities from a language model back into human-readable strings. As described above, these output probabilities represent the conditional probabilities of the next token given the current sequence of tokens. A naive method to determine the next token is through greedy search [21], in which the token with the highest probability is selected at each time step i

$$s_i = \arg\max_s(p(s|s_1,\ldots,s_{i-1}))$$

This token is then appended to the decoded sequence, and the process continues until the end-of-sequence (`eos`) token is produced.

Alternatively, randomness can be introduced by selecting the next word according to a categorical distribution, with probabilities determined by the language model's outputs [21].

While numerous other decoding strategies exist, this thesis will focus exclusively on this stochastic decoding method.

2.1.1 Transformer Language Models

The transformer architecture

Introduced by Vaswani et al. [52], the transformer neural network architecture has quickly become the prevailing model in not only natural language processing but also various other subfields of machine learning. The architecture takes a sequence of T vectorized tokens as input and, as a first step, maps them to a d-dimensional vector through an *embedding* layer. These vectors are subsequently stacked to create a $T \times d$ matrix X. This matrix is then passed to the core component of the architecture, the *scaled dot-product attention* mechanism, which can be summarized by the following formula [52]:

$$Y = \text{Attention}(X) = \text{softmax}\left(\frac{QK^T}{\sqrt{d}}\right)V$$

where Y is the output sequence, and the $T \times d$ matrices $Q = XW_Q$, $K = XW_K$, and $V = XW_V$ are referred to as *queries, keys, and values*, respectively. In this expression, W_Q, W_K, and W_V are learnable $d \times d$ matrices, and the softmax operation is applied row-wise. Following this computation, the $T \times d$ matrix Y is combined with the input X through a residual connection and subsequently passed through layer normalization. The output is then fed through a two-layer MLP and another residual connection and layer normalization. This entire computation is visualized in Figure 2.1 on the left.

Some other important aspects of the architecture are that multiple attention mechanisms are stacked in parallel to form what is known as the *multi-head attention mechanism*. Multiple such layers where multi-head attention follows a feed-forward MLP are then stacked sequentially where the output of the previous layer is passed as the input to the next layer. It must also be noted that positional embeddings are added to the embedded input so that models could understand the ordering of a sequence [52]. These embeddings are usually formed by taking sine and cosine functions of different frequencies and require setting a maximal sequence length (in tokens) that the model can process at a time, which is known as the *context size*.

The main advantage of transformer models is that they are much more parallelizable and therefore faster than their competitor RNN architectures which are sequential by nature [52]. Transformer models can be both encoders (Figure 2.1, left side) and decoders (Figure 2.1, right side), or combined into an encoder-decoder model. Decoder models have an additional

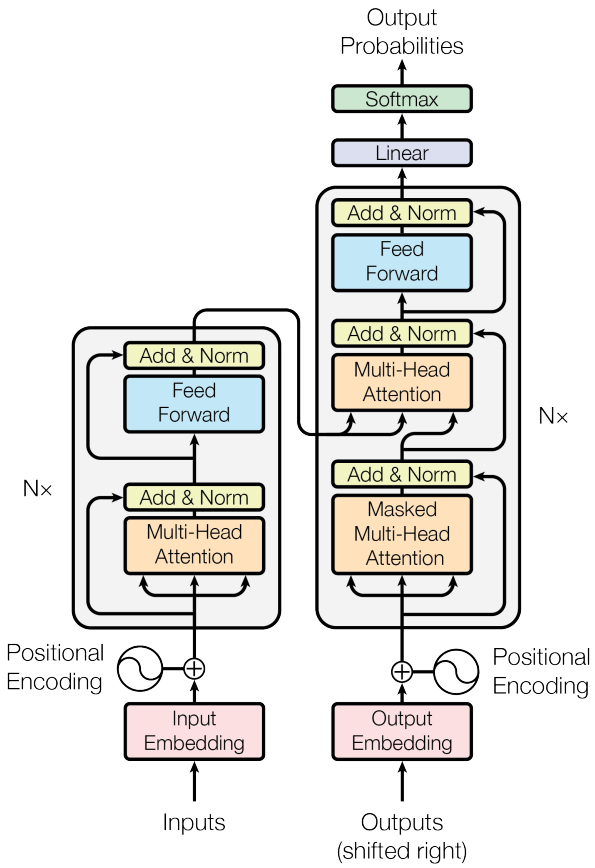


Fig. 2.1 The architecture of the transformer model. Figure source: [52].

linear projection followed by the softmax operation which maps each element of the sequence to the probability of the next token. Masking is used to prevent the model from accessing the information from subsequent tokens in the sequence [52]. The different types of models are utilized for various applications, for example, encoder models are employed for text classification, encoder-decoder models excel in sequence-to-sequence tasks (e.g. translation), and decoder models are suitable for text generation. It is important to note that in the case of an encoder-decoder model, the keys and values are derived from the encoder portion of the model, rather than the previous layer of the decoder. In this thesis, we will utilize both decoder-only and encoder-decoder versions which correspond to the two decompositions presented at the beginning of Section 2.1.

Pre-training

Pre-training refers to the process of training transformer models on large corpora of internet text [38]. The idea is that it is useful to teach the models to compute good representations and therefore model language (i.e. predict the next token) very well using vast amounts of

training data. The objective function used for pre-training is the cross-entropy loss between the predicted and the target tokens [37]. What makes the method so powerful is that no human labelling is required for this process as the labels can just be set to be the next token in a long sequence of text. Transfer learning can then be used to fine-tune such models on *downstream* tasks where usually only much smaller amounts of labelled data are available [16, 18, 37, 59].

Large language models

The term *large language models* is used to characterise massively scaled-up versions of transformer models that were pre-trained on very large text corpora. From a certain scale, these models start exhibiting *in-context* learning capabilities when a model can perform a downstream task given only a description of the task without its weights being modified (i.e. without requiring fine-tuning) [8]. Such inference method is called *zero-shot* prediction [38, 9, 50]. The model might also be given a few examples of how to perform the task which is then called *few-shot* prediction. It is often the case that the choice of prompt greatly affects the zero-shot/few-shot prediction performance [41, 57].

GPT-2

GPT-2 is a decoder-only transformer architecture introduced by Radford et al. [38]. The main achievement of this work was to show that language models can transfer to downstream tasks in a zero-shot manner, as described above. The model was pre-trained on the then newly introduced dataset of millions of webpages called WebText. The model has a context size of 1024 and comes in four different sizes ranging from 117 million to 1.54 billion parameters. The latter is referred to as *GPT-2 XL* and is one of the models used in this thesis.

UnifiedQA-v2

UnifiedQA-v2 [24] is an encoder-decoder model based on the T5 architecture [39]. The main contributions of the paper that introduced the latter are a unified approach that tackles all NLP problems in a text-to-text format, as well as a new dataset called the Colossal Clean Crawled Corpus (C4). Another important aspect is that T5 uses relative positional embeddings [47, 20] which allow it to process input sequences longer than the fixed context length of 512 that it was trained on. This will prove useful to us as some of the datasets that we will be using contain sequences that are longer than the model's context size.

The UnifiedQA-v2 is a successor of the UnifiedQA [25] model which was introduced as a single pre-trained model for the *question answering* (QA) task. The main difference

between the two versions is that the second version was trained on 20 datasets instead of 8 which allowed to achieve better intra-domain and inter-domain performance [24]. The models come in sizes of 60 million, 220 million, 770 million and 2.8 billion (which is referred to more succinctly as the "3B" variant) and 11 billion parameters [39], all of which will be considered in Chapter 3.

LLaMA

LLaMA is a suite of open-source decoder-only large language models introduced by Touvron et al. [50]. The models come in four different sizes ranging from around 7 to around 65 billion parameters. The model architecture is very similar to the original transformer architecture introduced by Vaswani et al. [52], with minor modifications which were found to be useful over the years [50]. A context length of 2,048 is used to train the models.

Apart from releasing powerful open-source language models, another important contribution of this paper is showing that state-of-the-art models can be trained on trillions of tokens obtained from exclusively publicly available datasets. Specifically, the "7B" version that will be relevant to us was pre-trained on 1 trillion tokens, most of which were used only once [50].

LLaMA 2 [51] is the second and even more performant iteration of the models. The main differences from the older models are a longer context length of 4,096, more tokens used for training (2 trillion for all model sizes), and new versions of the model fine-tuned for harmlessness and safety, as well as dialogue applications. The models have been trained on a novel collection of publicly available data, similar to the first iteration of the models [50, 51].

Vicuna

Vicuna [12, 58] is a collection of open-source chatbots that have been trained by fine-tuning the LLaMA 1 and 2 models in a supervised manner on conversations with ChatGPT that users have shared on a platform called ShareGPT[1]. The new evaluation method presented by the authors showed that Vicuna significantly outperforms the base LLaMA model, as well as closes some of the existing performance gap from the GPT-3.5 [9] and GPT-4 models [34].

The dataset of user conversations with ChatGPT consists of approximately 140 thousand samples which were appropriately transformed and filtered out to maintain the quality of the training data [12]. The authors report using a similar training regime to that of Stanford's Alpaca [49], with minor improvements to allow for the use of multi-turn conversations and reduce memory usage and training costs. The models come in sizes of 7 billion, 13 billion

[1] https://sharegpt.com/

and 33 billion parameters. We will be using the 7B-v1.3 and 7B-v1.5[2] versions of the chatbot which were fine-tuned starting from the LLaMA 1 and LLaMA 2 models, respectively.

2.1.2 LoRA

Low-Rank Adaptation (LoRA) [19] is a strategy that enables efficient fine-tuning of large language models. The work was inspired by Aghajanyan et al. [1], who argued that heavily overparameterized models can be described using fewer dimensions than the dimensionality of their constituent weight matrices. Consequently, the authors of LoRA hypothesize that the changes in weight matrices during fine-tuning could also be modelled by low-rank matrices.

More concretely, consider an $m \times n$ weight matrix W in a pre-trained language model. During fine-tuning, this matrix will be transformed into a new matrix $W' = W + \Delta W$. The matrix ΔW can then be decomposed into two lower-rank matrices A and B, with dimensions $m \times r$ and $r \times n$, respectively (where r is significantly smaller than m or n), such that $\Delta W = AB$ has a rank of at most r [19]. To control the contribution of the LoRA weights, a scaling parameter α is introduced, leading to the update rule $W' = W + \alpha \Delta W$. Moreover, the inputs are passed through a dropout layer with probability p before the LoRA weights are applied. It is also noteworthy that the modules to which the LoRA weights are applied can be chosen arbitrarily [19]. In this thesis, only the query and value attention weights will be adapted, with all other parameters, including the key attention weights and MLP layer weights, held fixed. This setup has been found to be sufficiently powerful for our purposes.

Overall, the introduction of the lower-rank trainable matrices allows for a significant reduction in the number of parameters that need to be updated during fine-tuning, thereby increasing efficiency. Other benefits of the technique include lower memory requirements, and the fact that models trained with this technique perform similarly or even outperform models that were fine-tuned in their entirety [19].

2.2 Discovering Latent Knowledge

For the purposes of this thesis, we refer to "discovering latent knowledge methods" as techniques that uncover the latent knowledge of language models by training a new, lightweight model on their activations or weights. Such models are referred to as *probes*. This thesis will specifically focus on probes that take the form of a logistic regression classifier and only use the model activations as input. Such a narrowed-down version of the DLK problem is

[2]https://github.com/lm-sys/FastChat/blob/main/docs/vicuna_weights_version.md

considered because our method will build on top of previous research, which is introduced below [11, 5].

We consider DLK to be a subset of *eliciting latent knowledge* (ELK) [13]. Equating the two would be inaccurate given the current limitations of DLK techniques and a lack of understanding of what knowledge they are exactly capable of extracting [43]. While ELK seeks to elicit the model's actual "beliefs", existing DLK methods are most likely only capable of extracting what humans would perceive to be the model's "beliefs" [13, 10] (we note that above and throughout the rest of the thesis we refer to model "beliefs" in quotation marks because of an ongoing debate about the nature and even the existence of "beliefs" in current state-of-the-art language models [11, 10, 43]). For instance, for a subjective piece of text, the "beliefs" uncovered by DLK probes might merely reflect opinions commonly found on the internet rather than the model's genuine internal stance on the subject. While current DLK methods show promise of aligning with the objectives of ELK in the future, they are not yet at the required level of sophistication.

Problem formulation

Considering that the methods described below both address the same problem, we describe it here. Suppose we have a dataset of binary questions $q_1, \ldots, q_n$, such as "Is the Earth round?". Corresponding contrast pairs x_i^+ and x_i^- are constructed by pairing the question q_i with its two possible answers. For the example above, x_i^+ and x_i^- would be expressed as "Is the Earth round? *Yes*" and "Is the Earth round? *No*", respectively. These contrastive pairs $\{(x_i^+, x_i^-)\}_{i=1}^n$ are the inputs to the language model whose "beliefs" we will be attempting to extract.

The next step involves extracting the model's activation (or embedding) vectors, denoted by $\mathbf{emb}(\cdot)$, for each element in the contrastive pair, namely, $\mathbf{emb}(x_i^+)$ and $\mathbf{emb}(x_i^-)$. These vectors will be the inputs to a linear probe, which is a linear projection followed by the sigmoid activation function (analogous to a logistic classifier). The core objective is then to find a probe capable of effectively differentiating between the embeddings of truthful inputs and those of deceitful ones.

Before proceeding to the rest of the technical details, we would like to take a moment to clarify that, although the original intent of this project was to build reward models that rely on human-labelled data as little as possible, we will operate in a regime where labels are known. This is because we will be synthesizing the contrast pairs ourselves, so we will know which element in the pair is correct as a byproduct. While this may initially appear as an unnecessary limitation, we would like to point out that the current DLK methods (described below) are only *partially* unsupervised anyway, as they require labels to resolve sign ambiguity (see

Section 3.1). Furthermore, as will be justified in Section 3.1, supervised probes will be considered in addition to unsupervised ones to improve our chances of achieving satisfactory fine-tuning results.

With the understanding of our assumptions, we now return to describing the problem. Mathematically, the probe can be written as

$$y = \sigma(\mathbf{w}^{\mathrm{T}}\mathbf{x})$$

where $\mathbf{x}$ is the input vector (what exactly $\mathbf{x}$ is for each method will be detailed below), $\mathbf{w}$ is the weight vector to be learnt, and σ denotes the sigmoid activation function.

2.2.1 CCS

The *Contrast-Consistent Search* (CCS) method was introduced by Burns et al. [11]. It is a (partially) unsupervised method whose main theoretical grounding is the observation that truth has to satisfy logical consistency properties. In simpler terms, the method looks for a projection of the language model's activations that would ensure that the probability of a statement is one minus the probability of the opposite statement.

In practice, to find the weight vector $\mathbf{w}$ described above, the method first computes the probabilities p_i^+ and p_i^- as follows (note that the extracted activations are normalized before being passed into the probe):

$$p_i^+ = \sigma(\mathbf{w}^{\mathrm{T}}\mathbf{emb}(x_i^+))$$

$$p_i^- = \sigma(\mathbf{w}^{\mathrm{T}}\mathbf{emb}(x_i^-))$$

with the embeddings chosen to be the activations corresponding to the last token from a particular layer of a transformer language model. The last layer is used by default, but other options are also considered in the paper [11]. Then, the loss function that ensures logical consistency can be formulated as

$$\mathscr{L}_{\text{consistency}}(q_i) = (p_i^+ - (1 - p_i^-))^2$$

$$\mathscr{L}_{\text{confidence}}(q_i) = \min\{p_i^+, p_i^-\}^2$$

$$\mathscr{L}_{\text{CCS}} = \frac{1}{n}\Sigma_{i=1}^{n}\mathscr{L}_{\text{consistency}}(q_i) + \mathscr{L}_{\text{confidence}}(q_i)$$

where the consistency loss ensures that the probabilities of a statement and its negation add to one, and the confidence loss pushes the classifier to be confident in its predictions (otherwise predicting 0.5 for both p_i^+ and p_i^- would be a degenerate solution).

The weight vector $\mathbf{w}$ is then found by optimising the objective function using gradient descent. The authors note that they repeat the optimisation process 10 times and choose the classifier with the lowest loss [11].

It is important to note that since the classifier is trained on both positive and negative members of the contrast pairs, both p_i^+ and $1 - p_i^-$ could be candidates for the probability of q_i being true. To resolve this, the authors propose using

$$p(q_i) = 0.5[p_i^+ + (1 - p_i^-)]$$

as the probability of q_i being true [11]. It must be noted that this assumes that p_i^+ is true, which might not necessarily be the case. A technique to solve this ambiguity is discussed in Section 3.1.

Advantages. The paper shows that the unsupervised method outperforms zero-shot accuracy by 4% on average. They also find that the method performs well even when the model's outputs are not meaningful, and that the resulting probe exhibits decreased prompt sensitivity.

Limitations. The authors themselves pointed out that the method does not work very well for decoder-only language models, and that the method is not as good as fitting a supervised logistic regression classifier on the contrast pairs using the labels. Moreover, the authors motivated their work by claiming that the logical consistency properties that truth has to satisfy are "unlikely to be satisfied by many other features" [11]. However, later works have shown that the method does not always generalize and have given examples of other features that could satisfy the consistency properties and therefore be found by the probe instead of the actual model's "beliefs" [43, 28].

2.2.2 VINC

The *Variance, Invariance, Negative Covariance* (VINC) method [5] builds on top of the ideas of CCS and proposes a new objective that can be optimised to find the weight vector $\mathbf{w}$. They formulate the objective function as finding the vector $\mathbf{w}^*$ that satisfies

$$\mathbf{w}^* = \underset{||\mathbf{w}||_2^2=1}{\arg\max}\{\mathbf{w}^{\mathrm{T}} A_{\mathrm{VINC}} \mathbf{w}\}$$

where the matrix A_{VINC} is defined as

$$A_{\mathrm{VINC}} = \alpha A_{\mathrm{confidence}} + \beta A_{\mathrm{invariance}} + \gamma A_{\mathrm{consistency}}$$

Here, α, β, γ are scalar multipliers, $A_{\text{confidence}}$ is a matrix responsible for encouraging the classifier to make confident predictions, $A_{\text{invariance}}$ pushes the classifier to be invariant across different prompt templates used on the same input, and $A_{\text{consistency}}$ ensures negation consistency (which now also supports both binary and multi-class question answering). The three matrices are simple functions of the covariance and cross-covariance matrices of the vectors $\mathbf{emb}(x_i^+)$ and $\mathbf{emb}(x_i^-)$ [5].

Importantly, prediction is performed slightly differently in VINC. The method first computes the *credences* c_i^+ and c_i^- (sometimes also simply referred to as "logits"):

$$c_i^+ = (\mathbf{w}^*)^{\mathrm{T}}\mathbf{emb}(x_i^+)$$

$$c_i^- = (\mathbf{w}^*)^{\mathrm{T}}\mathbf{emb}(x_i^-)$$

and then obtains the probability $p(q_i)$ of q_i being true as

$$p(q_i) = \sigma(c_i^+ - c_i^-) = \sigma((\mathbf{w}^*)^{\mathrm{T}}(\mathbf{emb}(x_i^+) - \mathbf{emb}(x_i^-)))$$

so that, effectively, the VINC classifier operates on the difference between the "positive" and "negative" activations. This is also how $p(q_i)$ is computed for the logistic regression models in the library that we are going to use as an implementation of the DLK methods.

The main advantage of VINC over CCS is that, since A_{VINC} can be shown to be symmetric, the objective function attains the *global* maximum when $\mathbf{w}^*$ is chosen to be the eigenvector corresponding to the largest eigenvalue of A_{VINC} [5]. This makes the method much faster than CCS. In terms of performance, the VINC project is still a work in progress, so one of the first things that we will do in Chapter 3 is compare the technique against CCS.

2.3 Reinforcement Learning from Human Feedback

Reinforcement learning from human feedback (RLHF) is a method that was first introduced by Christiano et al. [14] to train deep RL agents to perform complex tasks by only providing the agents with human feedback on which trajectory they prefer better and no tractable reward function. Later work showed that RLHF can be used to improve summary quality [48] and to tackle the instruction following [35] and alignment [2, 3] problems. The latter line of work is what we are going to focus on in this thesis.

Dataset

The first step of the technique is collecting a dataset of 10,000 to 100,000 samples [48, 35, 3] of human preferences over the outputs of a pre-trained language model, as shown in Figure 2.2 on the left. The first step in the process is generating pairs of outputs for a particular prompt and asking crowdworker labellers to choose which of the outputs is better. The preferences have to reflect a particular qualitative aspect of the output. For example, in the case of a summarization task, the labellers have to choose the output that summarizes the given input better [48]. Researchers communicate these qualitative aspects to the crowdworkers through instructions and take precautions to ensure that there is good agreement on the output rankings between the labellers and the researchers [48, 35, 3]. In the case of using RLHF for alignment, the crowdworkers were asked to choose the more helpful and honest of the two responses (or the more harmful when training for harmlessness) [3].

Supervised fine-tuning

The next (optional) step is to use a portion of the collected data to do *supervised fine-tuning* (SFT) on a pre-trained LLM [48, 35], as illustrated in Figure 2.2 on the left. The idea of this step is to make the LLM better at modelling the kind of text that humans prefer. If this step is performed, the resulting model is then used as the starting point to train a reward model (described in Section 2.3 below), and the base pre-trained model is used otherwise.

Reward modelling

The third and crucial step is training a *reward model* (RM) that essentially has to mimic how humans would rank the model responses, as shown in Figure 2.2 in the middle. Starting with a pre-trained LLM (or a supervised fine-tuned LLM) with the output layer replaced by a randomly initialized layer that outputs a single scalar, the objective is to train the reward model to predict which response would be preferred by a human. The reward model, therefore, receives the prompt and the two possible responses during training time. The loss function for a single sample can then be formulated as [48, 3]

$$\mathscr{L} = -\log(\sigma(r_\theta(x, y_{\text{chosen}}) - r_\theta(x, y_{\text{rejected}})))$$

where r_θ denotes the scalar output of the reward model, x is the initial prompt, y_{chosen} and y_{rejected} are the chosen and rejected responses, and σ is the sigmoid function. Such a loss function encourages the model to give high rewards to preferred outputs and low rewards to rejected outputs so that the two become well separated after training.

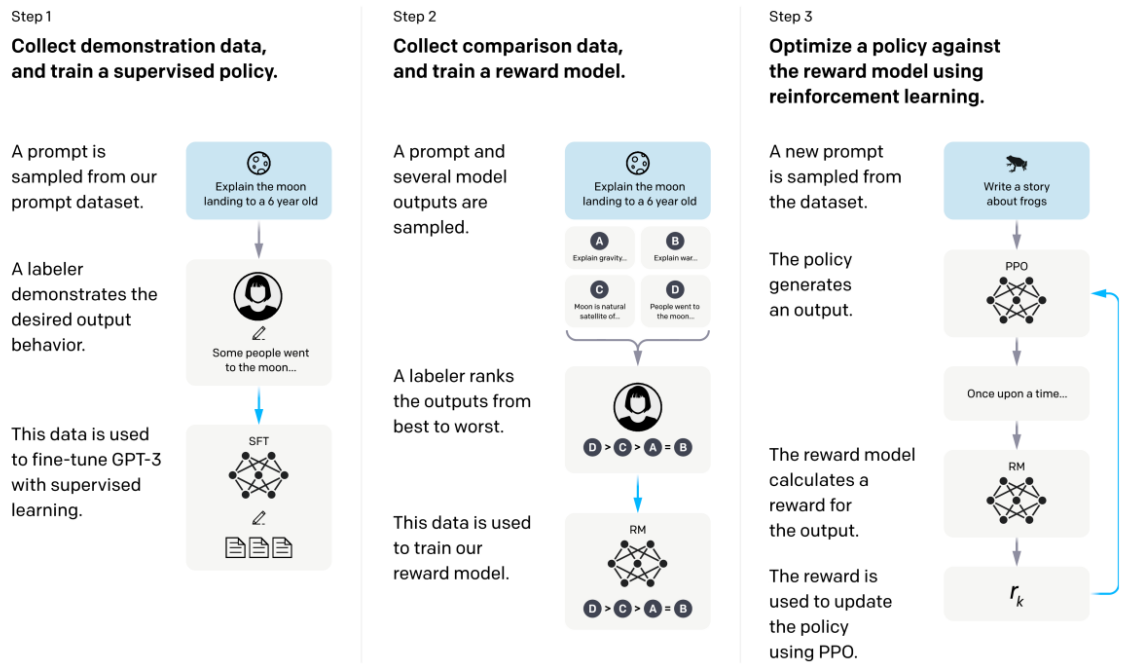


Fig. 2.2 An illustration of the steps of the RLHF fine-tuning method. (left) supervised fine-tuning (SFT), (middle) reward model (RM) training, and (right) reinforcement learning fine-tuning. The blue arrows indicate that this data is used to train one of the models. In the reward model training step, the boxes A-D denote LLM responses ranked by human labellers. Figure source: [35].

Fine-tuning using reinforcement learning

The final step is using the trained reward model to fine-tune a base pre-trained LLM (or the supervised fine-tuned version of it). The model that is being updated is often referred to as the *policy model*. The process is depicted in Figure 2.2 on the right and has the following stages [35]:

1. A prompt is sampled from the dataset.
2. The policy generates a response for the sampled prompt.
3. The reward model takes the sampled prompt and the generated response and outputs a reward.
4. The policy is updated using the *Proximal Policy Optimization* (PPO) [45] reinforcement learning algorithm to maximize the reward.

Note that the value function that is required for PPO training is also a separate language model, but is often initialized with the same weights as the reward model at the beginning of training [48, 35].

An important thing to note is that it was found that including an extra term in the reward is useful to prevent the policy from steering away too much from the output distribution that it has learnt during pre-training. The final reward for a sampled prompt x and the generated response y is then:

$$R(x,y) = r_\theta(x,y) - \beta \log[\pi_\phi^{\text{RL}}(y|x) - \pi^{\text{initial}}(y|x)]$$

where β is a scaling factor, r_θ is the reward model's output, $\pi_\phi^{\text{RL}}(y|x)$ is the policy parameterized by ϕ and $\pi^{\text{initial}}(y|x)$ is the pre-trained version of the model. Including this term helps to ensure that the model outputs do not deteriorate and that the reward model can accurately assess the response quality, as well as to encourage exploration [48, 35].

Important results

Importantly, [3] has found that RLHF can be used to train a *helpful* and *harmless* [2] language models with little or no degradation in performance on common NLP tasks. They have also found that there is a tension between helpfulness and harmlessness (increasing one can lead to a decrease in the other) and that RLHF can help increase truthfulness, but only marginally [35, 3].

Chapter 3

Using DLK Methods to Build Reward Models

This chapter presents our methodology and results on building unsupervised and supervised reward models. As mentioned before, the main motivation to build these kinds of reward models is to be able to perform RL fine-tuning to improve a pre-trained LLM's "honesty".

We will begin by checking that we can reproduce the results in the CCS paper [11]. Then, we will move on to explaining the design choices for our datasets and the reward model, before presenting the results of the best models that we will later use for RL training. Finally, we will summarize the chapter and outline the most important findings.

Datasets

For reward model building, we choose to use the same datasets that were used in the CCS paper, apart from the STORY_CLOZE [32] dataset which was not easily usable in the `elk`[1] open-source library that implements the DLK methods. Namely, the used binary question answering datasets are AG_News [56], Amazon_polarity [31], BOOLQ [15], COPA [42], DBpedia_14 [27], IMDB [30], PIQA [7], QNLI [54] and RTE [54]. We also use the same prompt templates for these datasets as in the original CCS paper [11] as they were readily available in the `elk` library.

The templates are mainly based on the `promptsource` library[2] [44] with a few additional prompt templates added by the authors of the CCS paper [11]. Additionally, we will utilize slightly transformed versions of these datasets, along with newly created templates, as presented in Section 3.2.1 below.

[1] https://github.com/EleutherAI/elk
[2] https://github.com/bigscience-workshop/promptsource

The fact that we are going to use these datasets also influences the definition of "truthfulness" that we are going to use. As briefly described above, we narrowly refer to either "truthfulness" or "honesty" as being able to correctly answer the binary questions formed using the datasets above and the prompt templates that come together with them. Using RL fine-tuning to improve "truthfulness" means improving a model's ability to correctly answer the formed binary questions in a zero-shot manner.

Models

We are going to use the UNIFIEDQA [25] and UNIFIEDQA-V2 [24] models to build the reward models. The former will only be used when reproducing the results from the original CCS paper [11], and the latter is what we are going to use to build our reward models because it outperforms the first iteration of the model (see Appendix A.1). The reason to use only this particular transformer architecture for reward modelling is that the original CCS paper found it to perform the best [11].

It is important to note that both versions of the model were trained on the BOOLQ dataset, and the second version was additionally trained on the PIQA dataset [25, 24]. Moreover, both versions of the model were trained on the SQuAD 1.1 dataset [40] from which the QNLI dataset was derived [54]. However, we do not worry about the possible data leakage and its effect on generalization since we will be evaluating on the corresponding test sets which the models have not seen during training.

Hidden states

An important consideration that was not previously discussed is how to choose the activations that are extracted from the model. The CCS paper chooses to report their main results for the last token of the last layer [11]. However, just as the paper itself finds, it is not always the last layer that results in the best-performing probe which is desirable for us to build the best possible reward models. Therefore, unless stated otherwise, we try the last token of each layer of a transformer language model and use the activations and report the results for the layer that gives the best test set performance.

3.1 Comparing VINC with CCS

As mentioned in Section 2.2.2, the VINC method is still a work in progress [5], so we begin by comparing its performance against CCS. This reproduction also serves as a sanity check

that the open-source implementation of DLK methods is reliable and checks out with the results in the paper.

We are using accuracy as the evaluation metric in this section because this is the metric that was used in the CCS paper [11]. However, in later sections, we will be using the AUROC metric instead of accuracy which is more informative than accuracy (accounts for different decision thresholds) and is not sensitive to class imbalance. We also note that in this section we report the results for the last token of the last layer instead of for the best-performing layer to match what was done in the paper. Moreover, for the most part, we will be using 1000 samples from each dataset for both training and evaluation of our probes as this is the number that was used in the original paper [11].

Differences in implementations

There are a few differences between the CCS implementation in the original paper and the one in the `elk` library. The first difference is data normalization, with two techniques being available in the library. The first method, which we will refer to as *correlation removal*, tracks cross-covariance statistics between the input activations and the output labels and then normalizes the data by using a projection matrix obtained by performing a singular value decomposition of the cross-covariance matrix. The matrix removes the subspace responsible for correlations between the inputs and the outputs from the activations [5].

The second normalization method is called LEACE[3] and is a more general method that can be used to "remove" a concept from a representation, meaning that no linear classifier can detect that concept using the "scrubbed" representations [4]. In this particular case, the concept of a contrast pair is removed [5]. This makes sense since, as described in the original paper, the contrast pair elements have (1) different endings (e.g. "Yes" or "No"), and (2) one of them is true and the other is false [11]. LEACE removes (1) and only leaves (2), the truth value of the contrastive pair elements, which is what we want the classifiers to pick up on.

The other difference from the original implementation is how sign ambiguity is handled. Sign ambiguity here refers to the fact that when making a prediction using DLK methods, it is assumed that p_i^+ is the contrast pair element that corresponds to truth, but that is not necessarily the case. This can be remedied by choosing the sign of the weight vector accordingly. In the original paper, both signs were tried and the one that performed better was picked (although it was also suggested that conjunctions could be used to resolve the problem [11]). This is not a very satisfying solution, especially in our case where we need to know the sign to make predictions.

[3]https://github.com/EleutherAI/concept-erasure

In contrast, the `elk` library utilizes Platt scaling [36] which is most commonly used to calibrate classification models. However, in the current case, the technique works by fitting *scale* and *bias* parameters a and b of the form

$$p_i^{\pm} = \sigma(a \cdot c_i^{\pm} + b)$$

where the parameters are found using maximum-likelihood estimation to match the output probabilities $p_i^{\pm}$ with the actual labels as best as possible. The effect is that the sign of a is learnt so that p_i^{+} can be assumed to correspond to the truth. The fact that labels are used is a bit unfortunate because it makes the method no longer fully unsupervised, but since we will be using supervised probes anyway (see Section 3.1 and Chapter 4), this does not concern us too much.

Justification for using supervised probes

It could be argued that using only unsupervised probes to build reward models for "honesty" would be more desirable because it would allow completely removing the need for human-labelled data. However, our decision to explore supervised probes is motivated by the following reasons:

1. We seek to determine whether reward models obtained using DLK methods could be applied for RL fine-tuning at all. This is important, as advancements in DLK method research might lead to the development of stronger unsupervised probes in the future. Our success with supervised probes serves as a proof of concept, suggesting that it may be worthwhile to explore the integration of more sophisticated unsupervised probes with RL fine-tuning once they become available.

2. Secondly, it should be noted that for current state-of-the-art DLK methods, the unsupervised methods are, at best, only *partially* unsupervised (as a reminder, labels are employed to resolve sign ambiguity, see Section 3.1). Furthermore, the datasets are formed synthetically and the labels are known in any case. As a result, we believe that exploring the use of supervised probes in addition to unsupervised ones is a reasonable and justified approach.

Reproduction of CCS and comparison with VINC

Before comparing the two normalizations, we check whether the results in the original paper are reproducible using the `elk` library and how the VINC method compares to CCS. The results are presented in Table 3.1. The results are more or less identical for the logistic

Table 3.1 CCS reproduction results (first 4 columns) and VINC comparison to CCS (last column). The UNIFIEDQA 11B model was used with 1000 data points for both training and evaluation. Correlation removal normalization is used as it is more similar to the normalization used in the CCS paper. The results show that the `elk` library is a reliable implementation of CCS, with minor differences likely influenced by the use of a different normalization technique compared to the original paper [11]. Furthermore, the VINC method performs on par with CCS.

Dataset	Logistic regression		CCS		VINC
	original	elk	original	elk	elk
AG_News	92.0%	93.5%	92.0%	74.2%	64.7%
Amazon_polarity	96.0%	96.6%	96.0%	90.6%	93.3%
BOOLQ	86.0%	87.6%	89.0%	86.9%	86.2%
COPA	93.0%	89.1%	88.0%	89.8%	87.9%
DBpedia_14	99.0%	99.0%	99.0%	98.4%	91.1%
IMDB	95.0%	92.8%	94.0%	85.5%	87.3%
PIQA	70.0%	70.3%	54.0%	59.7%	51.3%
QNLI	89.0%	89.9%	56.0%	55.8%	60.9%
RTE	81.0%	81.0%	64.0%	65.4%	63.5%

regression classifier (the first two columns). For the CCS method (the 3rd and 4th columns), we find that the results are slightly worse for the Amazon_polarity, BOOLQ and IMDB datasets, much worse for the AG_News dataset, and mostly similar for the other datasets. We believe that these differences occur because of the different normalizations that were used. Since no other fully-fledged replications of the original paper exist, we deem that the implementation in the `elk` library is reliable enough for our purposes.

As for the comparison between CCS and VINC, we find that VINC performs even worse on the AG_News dataset, performs slightly worse on DBPedia_14 and PIQA, but also outperforms CCS on Amazon_polarity and QNLI. Therefore, since training a VINC probe is much faster than a CCS probe, to simplify the design space and decrease the number of experiments to run, we will primarily be using VINC in our later experiments.

Normalization

As discussed above, here we compare the two different normalizations available in the `elk` library. The results are presented in Table 3.2. Both normalizations perform on par, so we have chosen to use the "correlation removal" version of normalization for the subsequent experiments in this thesis. This decision was made mainly because the LEACE method is a work in progress, which would require rerunning evaluations each time a new update comes

Table 3.2 Comparison of the different data normalization strategies available in the `elk` library, using the UNIFIEDQA 11B model with 1000 data points for both training and evaluation. Results for the COPA dataset using LEACE normalization could not be obtained due to an edge case that was not addressed in the library's implementation. The two normalization methods perform nearly identically, but LEACE is less reliable.

Dataset	CCS		VINC	
	correlation removal	LEACE	correlation removal	LEACE
AG_News	74.2%	76.9%	64.7%	64.8%
Amazon_polarity	90.6%	90.7%	93.3%	93.2%
BOOLQ	86.9%	86.9%	86.2%	85.9%
COPA	89.8%	–	87.9%	–
DBpedia_14	98.4%	98.2%	91.1%	91.2%
IMDB	85.5%	85.7%	87.3%	87.3%
PIQA	59.7%	59.3%	51.3%	52.0%
QNLI	55.8%	58.3%	60.9%	60.7%
RTE	65.4%	63.4%	63.5%	62.7%

out. Moreover, we had concerns about the uncertainty surrounding the reliability of the LEACE normalization.

3.2 Reward Model Design

In this section, we outline the design of the reward models that were later utilized for RL fine-tuning. The main factor that influenced the architecture of the reward model is that the CCS, VINC and logistic regression probes operate on contrast pairs instead of free-form text. Additionally, inspiration was also drawn from the way the activations are passed into the probe and the internals of the probe training process in the `elk` library.

3.2.1 Datasets

Prompting for "truthfulness"

Ideally, our "honesty" reward model should accept a piece of text as input and output a number between 0 and 1 that indicates the probability of the input being truthful. However, as we have previously discussed, the current state-of-the-art unsupervised DLK methods require contrast pairs to be passed in as inputs.

To accommodate this requirement, we develop new prompt templates that can be utilized to form the contrast pairs required to assess the truthfulness of a piece of text. The prompt

templates are structured similarly to the following, where the <input_text> represents the text whose truthfulness we want to evaluate:

> Consider the text below in triple backticks:
> ‘‘‘
> <input_text>
> ‘‘‘
>
> Question: is the given text truthful? Answer either yes or no.
> Answer: *yes / no*

We develop five new templates in total, as presented in Appendix A.2.1. We also evaluate which template exhibits the best performance in Section 3.2.2. As can be seen, these templates streamline the process of forming contrast pairs to simply applying the template and appending one of the contrasting responses (e.g. “yes” and “no”) to form the contrast pairs. These can then be passed as inputs into a DLK probe.

Custom “truthfulness” datasets

Having created the prompt templates, we now need data that we could use with the templates to train the DLK probes. Unfortunately, the original datasets cannot be used directly as-is, since the data points within them do not have associated truthfulness values. However, since the datasets are designed for binary question answering, we can readily transform them into statements with an associated truth value by using ground truth labels. This transformation is achieved by utilizing the prompt templates from the `promptsource` library, along with those contributed by Burns et al. [11].

Specifically, given a data point from one of the datasets, the transformation is performed by applying one of the available templates to it. For example, the entries in the IMDB dataset contain `text` and `label` fields, representing the text of the movie review and the ground truth label, respectively. By applying one of the templates, the entry is transformed into something like the following:

```
<text>
This movie review is <label>
```

If the sentiment of the review is correctly identified, the resulting prompt is truthful; otherwise, it is considered deceitful. In other words, the prompt now has a truth value associated with it. Importantly, to avoid having only truthful prompts, we randomly flip the label with a uniform probability during the transformation process. Apart from different templates and fields, the technique works exactly the same for other datasets.

We use the process described above to form multiple new versions of the original datasets which we call *custom "truthfulness" datasets*. The various versions will be used for different purposes later in the thesis. We note that we only apply the transformation to the whole evaluation split of the datasets, but only to a portion of the training data. The rest of the training data is left untransformed to be used for RL fine-tuning. The fact that we randomly flip the label ensures that the resulting datasets have approximately balanced class distributions. The specific custom datasets that we formed are described below:

1. **All datasets combined into one.** We first tried to take samples from all of the original datasets and combine them. We took 1,200 training samples from each dataset (apart from COPA which only has 400 training samples) for a total of 10,000 points for probe training. In a similar fashion, we ended up with 1,200 data points for probe evaluation and used 10,000 untransformed data points from the training set for RL fine-tuning.

2. **Individual datasets.** After facing challenges with RL fine-tuning on the combined dataset (see Section 4.1), we sought to identify a single simpler dataset suitable for both reward modelling and RL fine-tuning. Consequently, individual transformed versions of the original datasets were created. We sampled 1,500 data points for both probe training and validation from each dataset, except for COPA and RTE, which had fewer than 1,500 data points available. We found our reward models to work well on the custom IMDB dataset (see Appendix A.3), leading us to conduct numerous RL fine-tuning experiments on the 15,000 untransformed data points sampled from its training split, see Section 4.1 and Appendix B.1.

3. **QNLI Vicuna dataset.** Finally, upon discovering that we had to switch our policy model from a GPT-2 XL to Vicuna (see Section 4.1), we started looking for a new dataset and prompt template. The search was performed in an ad-hoc fashion. We first observed that Vicuna models were much more capable than GPT-2 XL and performed well on the BOOLQ, IMDB and RTE datasets even without fine-tuning. However, we then tested the models on the QNLI dataset and found potential for improvement through RL fine-tuning (58.3% and 69.0% zero-shot accuracy out of the box for the 7B-v1.3 and 7B-v1.5 models, respectively). Consequently, we created a new prompt template that would encourage the Vicuna model to answer in the correct format (see Section 4.2 and Appendix A.2.2). The final dataset used for probe training comprised 10,000 training and approximately 5,500 validation samples, with the remaining roughly 95,000 training samples reserved for RL fine-tuning.

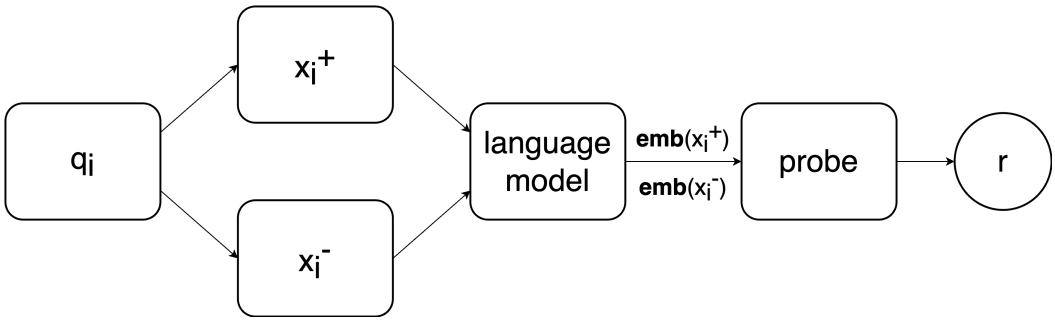


Fig. 3.1 The pipeline of the reward model.

3.2.2 The Reward Model Architecture

Our pipeline to obtain the reward for some text generated by the policy model during RL fine-tuning is depicted in Figure 3.1. The main steps are outlined below:

1. Take a piece of generated text, say q_i.
2. Form the contrast pairs x_i^+ and x_i^- using the "truthfulness" template.
3. Pass the contrast pairs to the language model and record the activations $\mathbf{emb}(x_i^+)$ and $\mathbf{emb}(x_i^-)$.
4. Pass the activations to a previously trained probe which returns the scalar reward r.

Choosing the best template to prompt for "truthfulness"

Here we compare the performance of our custom prompt templates. The results are presented in Table 3.3. As we can see, although by a narrow margin, the second template outperforms the other four, so we will be using it from hereon.

3.3 Finding the Best Probes

Encoder vs decoder hidden states

Before examining which probes perform best on the custom "truthfulness" datasets, we conduct an experiment to determine whether the encoder or the decoder part of the UNIFIEDQA-V2 model should be utilized for extracting activations. This consideration is grounded in the findings of the original CCS paper, which noted that both parts can demonstrate reasonable performance [11].

Table 3.3 Comparison of the performance of different "truthfulness" prompt templates on the *combined* dataset, utilizing the UNIFIEDQA-V2 model with 1,000 training and validation samples. The AUROC metric is used for the comparison. Note that the logistic regression performance is included solely for reference; originally, we did not consider supervised probes and selected a template based only on the VINC probe performance. Consequently, the second template is used for later experiments.

Template	VINC	logistic regression
1	0.657	0.862
2	**0.663**	0.837
3	0.661	0.841
4	0.546	0.835
5	0.596	0.784

Table 3.4 Comparison of the encoder and the decoder of the UNIFIEDQA-V2 11B model. The probes were trained using 1,000 data points for both training and evaluation. The AUROC metric is used for the comparison. The values for COPA, PIQA and RTE are missing because running them on the encoder resulted in a `ValueError` inside the `elk` library which we were not able to debug. The decoder will be used for future experiments since it outperforms the encoder in almost all cases, with the IMDB dataset being the only exception.

Dataset	Encoder	Decoder
AG_News	0.569	0.637
Amazon_polarity	0.980	0.982
BOOLQ	0.924	0.936
COPA	–	0.993
DBpedia_14	0.984	0.988
IMDB	0.966	0.956
PIQA	–	0.857
QNLI	0.827	0.918
RTE	–	0.886

The results are presented in Table 3.4. As we can see, for the cases where we were able to obtain the results, the decoder part of the model almost always outperforms the encoder. Additionally, we consider the fact that we were unable to extract activations for 3 datasets out of 9 due to instabilities in the `elk` library as a serious concern. Based on these findings, we decide to use the decoder part of the model to extract the model's activations for the remainder of the project.

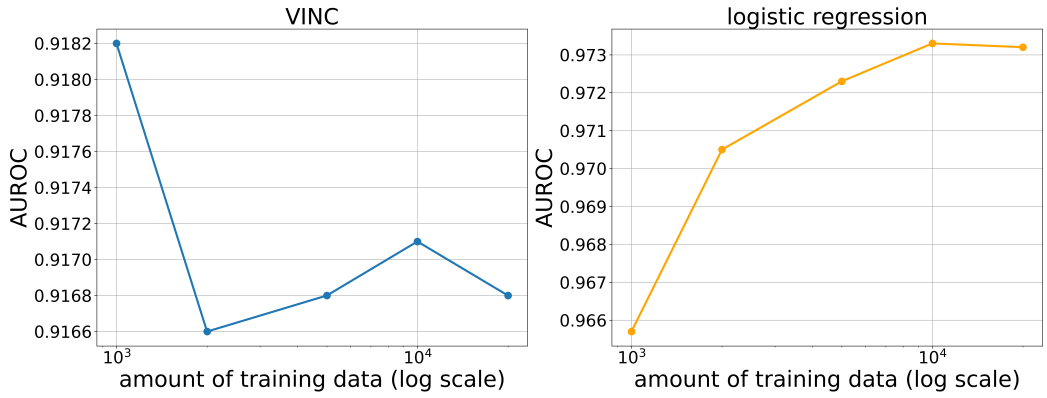


Fig. 3.2 Probe performance plotted against the number of samples used for training on the original QNLI dataset. The numbers of training samples used are 1,000, 2,000, 5,000, 10,000, and 20,000. 1000 validation samples are used. The UNIFIEDQA-V2 11B model is used. The performance decreases for the VINC method, whereas the performance improves and plateaus at 10,000 training samples for the logistic regression probe.

Amount of training data

In this section, we investigate the optimal number of training samples for both unsupervised and supervised probes. As shown in Figure 3.2, for the VINC probe, increasing the amount of training data actually decreases the performance, whereas for the logistic regression probe, using more training samples positively affects the performance. Therefore, to achieve maximum performance in the shortest amount of time, we will use 1,000 data points for unsupervised probe training and 10,000 data points for supervised probe training. For reference, training a logistic regression probe on top of the UNIFIEDQA-V2 11B model using 10,000 training samples with 1,000 validation samples took around 25 minutes, whereas the duration was 40 minutes while using 20,000 training samples with the same number of validation samples. These experiments were run on a setup with 8 NVIDIA A100 GPUs.

The language model size

In this section, we investigate the impact of the size of the UNIFIEDQA-V2 language model, from which we extract activations, on the performance of the probe. The detailed results are presented in Table 3.5. As might be expected, the overall trend is that the performance tends to increase with the size of the model. The 11B model exhibits the best performance in almost all instances. However, we also find that the 3B model often performs at a comparable level, especially for supervised logistic regression models, and even surpasses the 11B model on the custom IMDB dataset. Taking GPU memory constraints into consideration, we will be exclusively using the 3B version of the model for RL fine-tuning.

Table 3.5 Impact of the size of the UNIFIEDQA-V2 models on the performance on our custom datasets. A total of 10,000 data points were used for training, and as many as were available for each dataset were used for evaluation. The AUROC evaluation metric is used. "LR" stands for logistic regression. The results indicate that performance generally improves with increasing model size. The 11B models perform the best, with the 3B models performing comparably well.

Model size	Combined		IMDB		QNLI	
	VINC	LR	VINC	LR	VINC	LR
small	0.515	0.561	0.524	0.598	0.524	0.691
base	0.544	0.668	0.524	0.839	0.556	0.797
large	0.582	0.808	0.636	0.942	0.557	0.883
3B	0.625	0.878	0.750	**0.969**	0.536	0.940
11B	**0.662**	**0.908**	**0.815**	0.966	**0.582**	**0.962**

Table 3.6 Comparison of different supervised logistic regression probe training regimes. The UNIFIEDQA-V2 11B model is used. A total of 10,000 data points were used for training, and as many as were available for each dataset were used for evaluation. The AUROC evaluation metric is used. Cross-validation refers to the method used to find the level of L2 penalty. The results demonstrate that training without regularization yields better performance.

Training regime	Combined	IMDB	QNLI
No regularization	**0.908**	**0.966**	**0.962**
Cross-validation	0.898	0.961	0.958

Supervised probe training regime

The `elk` library supports two types of supervised probe training regimes, one where just the probe without regularization is trained, and another one where multiple probes are trained using different levels of L2 regularization and the best one is chosen using cross-validation. Surprisingly, the vanilla training regime seems to perform better than the regularized probe, as can be seen in Table 3.6. Therefore, we will not use cross-validation when training our best-performing supervised probes.

Inspecting the reward distributions

The supervised probes trained on the QNLI dataset will be extensively utilized for RL fine-tuning (see Section 4.2). Therefore, it is important to assess whether these reward models are proficient enough to evaluate the outputs of the policy models that will be used. To examine this, we conduct an experiment in which we ask one such policy model to generate responses

for a subsample of RL training data points, and then compute the rewards using these reward models.

The results are presented in Figure 3.3. As we can see from the top part of the figure, the rewards for correct outputs are concentrated near one, whereas the rewards for incorrect outputs are concentrated close to zero. This signifies that the reward models are indeed proficient at evaluating policy models' responses given that they come from the set of possible answer choices. In contrast, we would expect the responses not in that set to be given rewards close to 0, but the bottom part of the figure illustrates that both very low and very high rewards are common. The solution to this issue will be discussed in Section 4.2.

We conducted a similar posthoc analysis for both the VINC and CCS unsupervised probes, the results of which are presented in Appendix A.4. The findings are that the unsupervised probes generalize poorly and cannot produce reasonable rewards for RL training samples. This explains why we had to switch to supervised probes to achieve success, as will be further discussed in Sections 4.1 and 4.2.

3.4 Summary

We conclude this chapter by highlighting the five key achievements and findings towards our aim to use DLK methods to build reward models for "honesty" that can be used for RL fine-tuning:

1. DLK methods can indeed be used to build reward models that promote "truthfulness".
2. VINC performs on par with CCS and is faster, so we use it for our unsupervised reward models.
3. We form the custom "truthfulness" versions of the datasets used in the original paper [11] to make them suitable for reward modelling.
4. Supervised probes perform best in all cases on the custom datasets, but unsupervised ones can also perform comparably well on certain datasets (see Table 3.5).
5. Our reward models are capable of correctly assessing the outputs of the policy models that will be used in the next section.

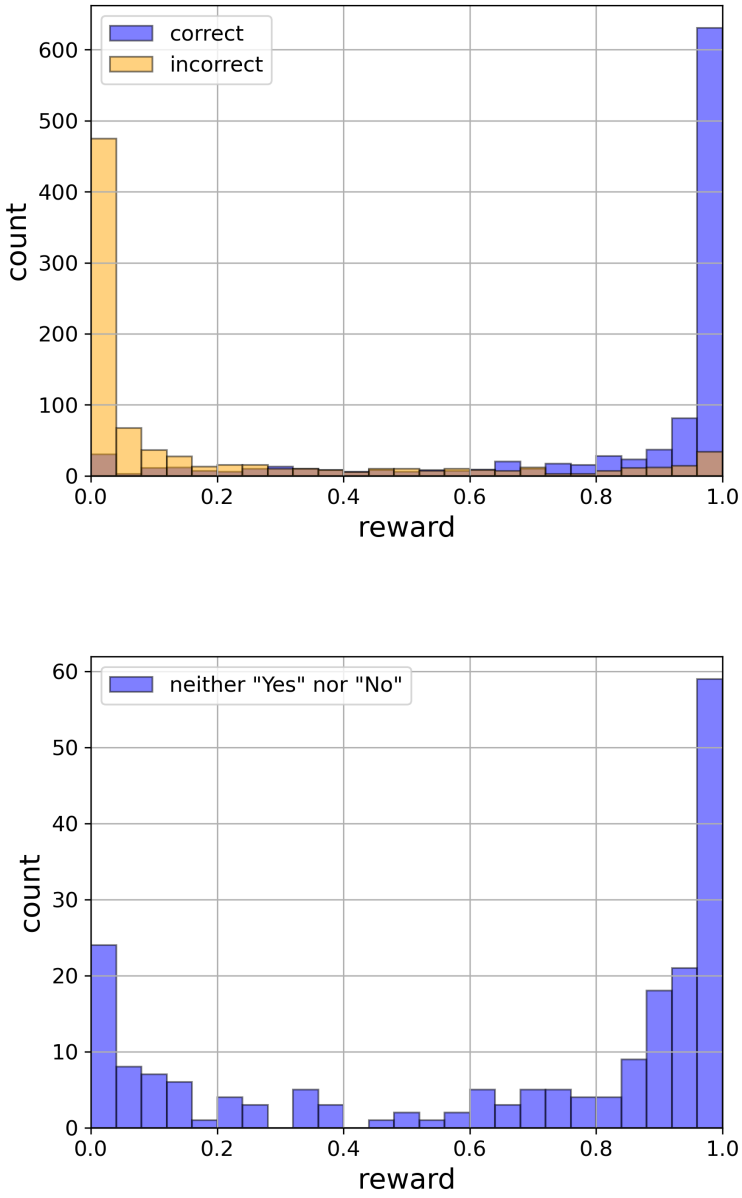


Fig. 3.3 The reward distributions for the outputs of the Vicuna 7B-v1.3 model, using the UNIFIEDQA-V2 3B model with a supervised probe as the reward model, and a total of 2,048 samples from the QNLI RL training dataset. *(top)* The rewards for only "Yes" and "No" outputs; rewards for other responses were filtered out, *(bottom)* The rewards for outputs that were neither "Yes" nor "No". The plots show that reward models based on supervised probes are sufficiently capable of discriminating truthful inputs from deceitful ones, provided that out-of-distribution inputs are filtered out.

Chapter 4

Fine-tuning Using Reinforcement Learning

In this chapter, we utilize the previously built reward models to perform RL fine-tuning with an aim to improve a pre-trained LLM's "honesty" in the sense described earlier in Section 3. We will begin by briefly presenting our initial naive approaches and the challenges we encountered, and subsequently explain how we solved each problem to ultimately arrive at a working method. We will finish the chapter with a summary of the most important results.

We used the `trl` library[1] [53] to run our RL fine-tuning experiments. We chose the GPT-2 XL [38] and Vicuna [12] as the policy models that we will try to improve. The datasets used are the samples that were set aside when forming our custom "truthfulness" datasets, as described in Section 3.2.1. We ran our experiments using the distributed data-parallel strategy on 8 NVIDIA A100-SXM4-40GB GPUs kindly provided by Stability AI[2]. Since training the 7 billion parameter models in their entirety would not have been feasible even with these GPUs, we utilized the LoRA approach [19], which drastically reduced the number of trainable parameters. The UNIFIEDQA-V2 3B models are used instead of 11B to further reduce the memory requirements. To evaluate our models, we used the Language Model Evaluation Harness[3] and its `big-refactor` branch[4] [17].

Hyperparamaters

Before diving into the experimental results, we present the hyperparameters that we found to be important to tune when performing RL fine-tuning:

[1] https://github.com/lvwerra/trl
[2] https://stability.ai/
[3] https://github.com/EleutherAI/lm-evaluation-harness
[4] https://github.com/EleutherAI/lm-evaluation-harness/tree/big-refactor

1. **Learning rate.** As for any model optimized using gradient descent-based methods, choosing a good learning rate is crucial. We test values of 1.4e-5, 1e-5 (default), 5e-6 and 1e-6.

2. **Batch size.** The number of samples that will be used for one update of the PPO algorithm. Choosing this as a power of two that still fits into the GPU memory will turn out to not necessarily be the best option (see Section 4.2.1).

3. **Policy batch size.** The policy model uses its own batching to generate the responses to the input prompts. It is best to choose this as large as possible to increase the training efficiency. The default value is 4.

4. **Number of training steps.** The number of batches that the model sees during training. In case more steps are desired than the amount of training data available, we perform multiple passes over the data.

5. **Value function coefficient.** A PPO algorithm hyperparameter that scales the value function loss before it is summed with policy loss [45]. We tried values of 0.1 (default), 0.2, 0.5 and 1.0.

6. **Initial KL coefficient.** This is the initial value of the β parameter from Section 2.3 responsible for weighing the KL term in the total reward. As a reminder, the term is used to control the policy model from diverging too much from the original output distribution. We test values of 0.2 (default), 0.5 and 1.0.

7. **Maximum number of new tokens.** The maximum number of new tokens that the policy model can generate for a given input prompt. For the *combined* dataset, we set it to the maximum length in tokens of the "golden" responses for a given input batch. For the custom IMDB dataset, we set it to 4 tokens, the minimum number allowed by the `trl` library at the time of running the experiments. Finally, for the custom QNLI dataset, we used either 1 or 2 tokens, following an update to the `trl` library that lifted the requirement for a minimum of 4 tokens.

We also note that we used the Adam optimizer [26] which is the default in the `trl` library. There are also numerous other hyperparameters that can be tuned [53], but we found that the default values worked well for our purposes.

4.1 Unsuccessful Experiments with the GPT-2 XL Model

Here we briefly present our unsuccessful initial naive approach and later iterations of it. These did not work, but provided essential lessons that allowed us to arrive at our final successful setup. A more comprehensive description of these experiments is presented in Appendix B.1.

We started out by performing RL fine-tuning using the *combined* dataset along with a UNIFIEDQA-V2 3B reward model with an unsupervised VINC probe. We found that the models would not surpass a reward of 0.5 which corresponds to random guessing (given that the model only outputs one of the possible answer choices, e.g. "Yes" or "No").

To investigate whether the policy models improve at all, we calculate the *baseline reward* by passing all of the RL training samples through the policy model, obtaining rewards for the generated responses and averaging them. We found that the models would improve upon the baseline only marginally.

We realized the need for simpler datasets and stronger probes, leading us to switch to the custom IMDB dataset. We experimented with both the unsupervised VINC and the supervised logistic regression probes as reward models, as the latter showed more potential to produce satisfactory results (see Section 3.3).

Unfortunately, we once again observed minimal to no improvement in the performance of the models. However, we discovered that the policy models' responses were incoherent and far from what we would have wanted them to output, which is what we believe to have been the cause of such results. For example, instead of outputting the sentiment of the movie review, the models would generate responses like "`____ Let us know`", "`\nSparse arrays`" or "`\n.\n\n`". We hypothesize that this aberrant behaviour explains the inferior performance of models trained using the supervised probe compared to the original model or the one trained with a VINC probe (see Appendix B.1). Consequently, we decided to transition to using the Vicuna models.

4.2 Successful Experiments with the Vicuna Models

After realizing that the generated responses are nonsensical, we made a judgement that the GPT-2 XL models were not sufficiently capable to produce the desired outputs. Consequently, we decided to explore new policy models. A series of prototyping experiments led us to the conclusion that the Vicuna models could be utilized. This conclusion was reached because we found them to be much more capable, and fine-tuning on ChatGPT conversations enabled the models to excel at responding with precisely what was requested from them, given an

appropriate prompt. However, such behaviour did not manifest automatically, so in this section, we will present the methodologies we employed to finally achieve a working setup using the Vicuna models.

Before presenting the techniques, we again reiterate that the custom QNLI dataset was used with the Vicuna models. This choice was made because it was the first one on which the models did not already perform outstandingly well (see Section 3.2.1).

Prompting

The first measure we undertook, which we had not done before, was to design an effective prompt template to guide the Vicuna models to output only one of the possible answer choices (which are either "Yes" or "No" for the QNLI dataset) in the desired format. By the latter, we mean that we wanted the models to output only the single word as the answer, rather than prefacing it with phrases such as "*The answer is . . .* " or repeating the "*Answer:*" prefix that was already present in the prompt. We took inspiration from the Prompt Engineering Guide[5]. The final prompt that proved highly effective is presented in Appendix A.2.2. However, prompting alone was insufficient to make the models work.

Maximum number of new tokens

Secondly, we discovered that an adjustment to the maximum number of new tokens generated by the policy model was necessary. As previously noted, the requirement that at least 4 tokens are generated was lifted in the `trl` library when we began working with the Vicuna models and the QNLI dataset. Consequently, we initially set the maximum number of new tokens to 1. Unfortunately, this resulted in the injected LoRA parameters becoming NaNs after the first batch update, for reasons that remain unclear.

We then increased the maximum number of tokens to 2. However, since the answer choices "Yes" and "No" corresponded to a single token for the Vicuna tokenizers, we had to implement additional post-processing strategies before passing the responses to the reward model. Specifically, we found that the second token generated would either be a space, a punctuation character, or a newline character. Fortunately, we were able to easily strip them and recover the desired format in which only a single word was generated.

Regularization

While executing experiments with the previously described adjustments, we observed that the policy models would still occasionally generate a response that was neither "Yes" nor

[5]https://www.promptingguide.ai/

"No". In such cases, the reward models, not having been trained to accurately assess those particular outputs, would produce rewards that were more or less arbitrary and not necessarily close to zero. This situation oftentimes led to instabilities in training, as the policy model would begin to generate undesired responses with increasing frequency because the reward model was not discouraging it to do otherwise.

To address this challenge, we introduced a regularization technique to enforce that the policy model only outputs one of the desired answer choices. Specifically, if the response did not match one of the required choices, rather than passing the response to the reward model, we assigned a reward of -1 to it. We found this method to be highly effective in encouraging the model to fixate on producing one of the desired output choices over time, resulting in much more stable training.

4.2.1 Results

In this section, we present the experimental results for the Vicuna models, which were trained using reinforcement learning on the QNLI dataset. The UNIFIEDQA-V2 3B with a supervised probe was utilized as a reward model. Evaluations of the fine-tuned models on Open LLM Leaderboard[6] datasets are also presented.

Setting the hyperparameters

We only tested one configuration for the newly injected LoRA matrices, which we found in one of the examples in the `trl` library. Specifically, the rank r was set to 16, the scaling constant α to 32, and the dropout probability p to 0.05. Moreover, we used 8-bit quantization to further reduce memory usage.

The space of other possible hyperparameters was described in Section 4. Utilizing LoRA reduced the number of trainable weights in comparison to when the full GPT-2 XL models were trained, thus enabling us to increase the policy batch size to 16. As previously explained, the maximum number of new tokens was set to 2.

As for the number of training steps, it was observed that **all** runs would invariably fail to complete one full pass through the training data (approximately 95,000 data points). In this context, we refer to failure as the previously described scenario when the LoRA parameters would become NaNs, leading the models to generate nonsensical responses and rendering further training ineffective. This highlights how unstable using reinforcement learning to fine-tune LLMs can be (for further discussion, see Section 5). Consequently, we set the

[6]https://huggingface.co/spaces/HuggingFaceH4/open_llm_leaderboard

number of training steps to be as much as would be required to make a full pass through the training data given a particular batch size and our setup with 8 GPUs.

We initially set the batch size to 128, the largest value that was feasible without running into memory issues. However, we later found that decreasing the batch size helped ensure that the models trained for longer before a failure occurred. We hypothesize that this was because, with a batch size of 128, the global batch size would be 1024 on an 8 GPU setup, leading to an update that proved excessively large for the PPO algorithm. In other words, more frequent and smaller updates appeared to offer greater stability than less frequent and larger updates. Therefore, a final batch size value of 16 was chosen.

Regarding the learning rate, we experimented with values of 1e-5, 5e-6, and 1e-6. We discovered that reducing the learning rate would slow down the rate of the policy model's improvement, but would not necessarily prevent training instabilities. Consequently, a learning rate of 1e-5 was selected. As for the value function coefficient and the initial KL coefficient, we explored all of the values listed in Section 4 in a somewhat ad-hoc manner. It was found that values of 1.0 and 0.2 worked well for the value function coefficient and the initial KL coefficient, respectively.

Experimental runs

The results of the best runs for the Vicuna 7B-v1.3 and 7B-v1.5 policy models are presented in Figure 4.1. As can be seen, both models exhibit substantial improvement over their respective baselines, peaking at a value of approximately 0.8. We also note that failures were still present in both runs.

Evaluation on the QNLI validation set

We then evaluated the models on the original QNLI dataset to see if RL fine-tuning has improved their performance. We note that we utilized the same prompt for evaluation that was employed during training, instead of the original prompt present in the Language Model Evaluation Harness. As previously mentioned, this ensures consistency between the model training and evaluation stages.

As illustrated in Table 4.1, fine-tuning substantially improves the performance of both models. Specifically, the 7B-v1.3 model improves by 22.3%, while the performance of the 7B-v1.5 model improves by 18.0%. We emphasize that these improvements were achieved with at most 8,192 processed training samples and no more than 2 hours of training on 8 GPUs.

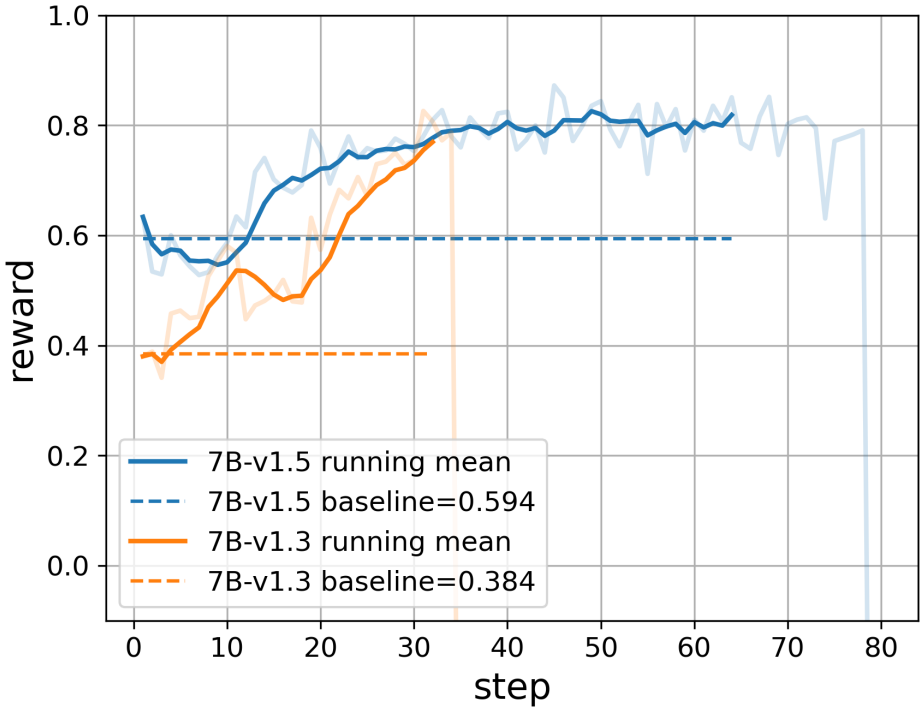


Fig. 4.1 The graph illustrates the running mean of reward against training iteration for Vicuna 7B-v1.3 and 7B-v1.5 policy models. The reward model is a 3B UNIFIEDQA-V2 model with a supervised probe. A rolling window smoothing with a window size of 5 is applied. The lighter lines in the graph indicate that both runs decline to a value of -1 after a certain point (the y-axis range has been trimmed for clarity). Both models substantially improve over the baseline. The last checkpoints that were saved for each model were after 32 and 64 steps for the 7B-v1.3 and 7B-v1.5 models, respectively. This corresponds to 4,096 and 8,192 processed training samples, which took about 1 and 2 hours on our hardware.

Evaluation on the Open LLM Leaderboard Datasets

We also evaluate the models on the Open LLM Leaderboard datasets before and after RL fine-tuning. This is done to investigate whether performing RL fine-tuning adversely affects the models' performance on general NLP tasks. This is especially important since the models are encouraged to output only one of the possible answer choices during training.

As shown in Table 4.2, rather than deteriorating, the models' average performance actually improves. As for the individual datasets, we observe that performance improves in all cases, with the exception of the 7B-v1.5 model on the HellaSwag dataset where it remains unchanged. We also observe that the most substantial improvements occur on the TruthfulQA

Table 4.1 Comparison of the models' zero-shot accuracy on the QNLI dataset before and after RL fine-tuning. Our custom prompt is used for evaluation. We observe a significant improvement in this specific task for both kinds of policy models that we used.

Model	QNLI
7B-v1.3	58.3%
7B-v1.3 + RL	**81.2%**
7B-v1.5	69.0%
7B-v1.5 + RL	**87.0%**

Table 4.2 Comparison of the models' zero-shot accuracy on the Open LLM Leaderboard datasets before and after RL fine-tuning. We note that the values can differ slightly from those presented in the leaderboard since we perform 8-bit inference for the ARC, HellaSwag datasets, and different seeding might be used in some cases. The results indicate that the models' performance on general NLP tasks does not degrade and even improves slightly. Interestingly, the largest improvements occur for the TruthfulQA dataset, amounting to 1.6% and 1.5%, respectively. This indicates that even though we only aimed to improve "truthfulness" in a very specific way that we defined above, the models' truthfulness also improves in a more general sense.

Model	ARC	HellaSwag	MMLU	TruthfulQA	Average
7B-v1.3	50.9%	77.6%	48.1%	47.0%	55.9%
7B-v1.3 + RL	51.4%	77.8%	48.3%	48.6%	**56.5%**
7B-v1.5	53.5%	77.2%	51.1%	49.9%	57.9%
7B-v1.5 + RL	53.9%	77.2%	51.3%	51.4%	**58.5%**

dataset, which indicates that the models' truthfulness improves in a broader sense than we initially defined.

4.3 Summary

In conclusion, in this chapter, we have taken the reward models built in the previous chapter and used them to perform RL fine-tuning. After initial unsuccessful attempts, we have demonstrated that our method can be successfully applied to improve the performance of Vicuna models on the QNLI dataset. The key findings are:

1. Reward models built using DLK methods can indeed be used to fine-tune LLMS to be more "truthful".

2. Using the current unsupervised DLK methods leads to reward models that are not strong enough to be used in RL fine-tuning, so supervised probes have to be used.

3. The pre-trained models that are being fine-tuned have to already be capable enough so that they could be guided to respond in the requested format.

4. Multiple techniques have to be employed to improve the training stability of reinforcement learning.

5. Our method achieves a performance improvement on the QNLI dataset with only a relatively small amount of training data and without compromising the models' performance on broader NLP tasks. Additionally, truthfulness is improved in a more general sense than how we initially defined it.

Chapter 5

Conclusions, Limitations and Future Work

This thesis investigated whether DLK methods could be used to build reward models for "truthfulness". We first explored how to train the most performant DLK probes and how to convert them into reward models by combining them with existing pre-trained LLMs. We discovered that both unsupervised and supervised probing techniques can be reliable at classifying truthful text depending on the dataset used, but supervised probes currently perform equally well or better than their unsupervised counterparts.

We then conducted RL fine-tuning experiments with the newly designed rewards models. We describe various techniques employed to stabilize the training procedure. We observed that fine-tuning improved the models' "truthfulness" on the QNLI dataset when a reward model based on a supervised probe was used. We also found that this enhancement only required a small amount of data, did not reduce the models' capabilities on other NLP tasks, and improved truthfulness in a broader sense than we initially defined.

Broader impacts

As previously outlined, the broader motivation for building reward models using techniques like DLK is to use them to improve the "honesty" of LLMs, which current methods, such as RLHF, are not capable of. Furthermore, unsupervised DLK methods would allow obtaining these reward models without requiring labelled data. Unfortunately, we could not fully realize these objectives due to the limitations of current DLK methods. However, it is worth noting that our framework for building rewards models is agnostic to how the probes are

obtained. Consequently, should future breakthroughs in DLK methods occur, our approach could be readily applied.

Furthermore, we believe that our work could be the missing piece of the puzzle, illustrating how DLK methods might be used not only for interpreting language models but also for aligning them. Specifically, one of the authors of the CCS paper has highlighted that the broader aim of their work is to find techniques that would allow for the interpretation of even superhuman LLMs [10]. A method such as ours could then potentially serve as a complement to this line of work, enabling it to be used for alignment in addition to interpretability.

Limitations and Future Work

Finally, we present the limitations of our work and potential directions for future research:

1. **Unsupervised DLK methods are too weak.** We were not able to achieve successful results with unsupervised DLK probes. It would be interesting to try our method with more robust unsupervised DLK techniques if they become available in the future.

2. **Narrow definition of "truthfulness".** We use only a very narrow definition of "truthfulness". Although we still observe an improvement in truthfulness in a more general sense, it would be interesting to look for datasets that would capture it in a broader sense, but would still be suitable for reward modelling.

3. **Instability of RL fine-tuning.** As we observed, RL fine-tuning can be very unstable. It would be desirable to pin down the cause of these instabilities and see if training for longer would improve the performance even more.

4. **Prompt sensitivity.** As depicted in Appendix B.2, the fine-tuned models remain sensitive to prompting. Although this phenomenon is applicable to most LLMs, it would be desirable that the improvement in "truthfulness" was less prompt dependent.

5. **Dependence on supervised instruction fine-tuning and RLHF.** Both of our policy models were fine-tuned to follow instructions more effectively, and the Vicuna 7B-v1.5 is based on LLaMA 2 model, so it may have been fine-tuned using RLHF as well. We found that it was challenging to enforce other models (e.g. the plain LLaMA 1) to produce outputs in the desired format. Removing this limitation could be an exciting direction for future exploration.

6. **Integration with Other Reward Models**: It would be interesting to see if our "honesty" reward models could be incorporated with reward models for helpfulness and harmlessness and jointly used to fine-tune a pre-trained LLM.

Finally, it would be interesting to investigate how our finding that only a small number of narrow data is needed to improve the truthfulness in a more general sense would stand against scaling the policy models.

References

[1] Aghajanyan, A., Zettlemoyer, L., and Gupta, S. (2020). Intrinsic dimensionality explains the effectiveness of language model fine-tuning.

[2] Askell, A., Bai, Y., Chen, A., Drain, D., Ganguli, D., Henighan, T., Jones, A., Joseph, N., Mann, B., DasSarma, N., Elhage, N., Hatfield-Dodds, Z., Hernandez, D., Kernion, J., Ndousse, K., Olsson, C., Amodei, D., Brown, T., Clark, J., McCandlish, S., Olah, C., and Kaplan, J. (2021). A general language assistant as a laboratory for alignment.

[3] Bai, Y., Jones, A., Ndousse, K., Askell, A., Chen, A., DasSarma, N., Drain, D., Fort, S., Ganguli, D., Henighan, T., Joseph, N., Kadavath, S., Kernion, J., Conerly, T., El-Showk, S., Elhage, N., Hatfield-Dodds, Z., Hernandez, D., Hume, T., Johnston, S., Kravec, S., Lovitt, L., Nanda, N., Olsson, C., Amodei, D., Brown, T., Clark, J., McCandlish, S., Olah, C., Mann, B., and Kaplan, J. (2022). Training a helpful and harmless assistant with reinforcement learning from human feedback.

[4] Belrose, N., Schneider-Joseph, D., Ravfogel, S., Cotterell, R., Raff, E., and Biderman, S. (2023). Leace: Perfect linear concept erasure in closed form.

[5] Belsore, N., Mallen, A., Ghosh, D., Laurito, W., O'Brien, K., Wan, A., Wright, B., Asai, A., and Elazar, Y. (2023). elk.

[6] Bengio, Y., Ducharme, R., Vincent, P., and Janvin, C. (2003). A neural probabilistic language model. *Journal of Machine Learning Research*, 3:1137–1155.

[7] Bisk, Y., Zellers, R., Bras, R. L., Gao, J., and Choi, Y. (2020). PIQA: Reasoning about physical commonsense in natural language. In *Thirty-Fourth AAAI Conference on Artificial Intelligence*.

[8] Bommasani, R., Hudson, D. A., Adeli, E., Altman, R., Arora, S., von Arx, S., Bernstein, M. S., Bohg, J., Bosselut, A., Brunskill, E., Brynjolfsson, E., Buch, S., Card, D., Castellon, R., Chatterji, N., Chen, A., Creel, K., Davis, J. Q., Demszky, D., Donahue, C., Doumbouya, M., Durmus, E., Ermon, S., Etchemendy, J., Ethayarajh, K., Fei-Fei, L., Finn, C., Gale, T., Gillespie, L., Goel, K., Goodman, N., Grossman, S., Guha, N., Hashimoto, T., Henderson, P., Hewitt, J., Ho, D. E., Hong, J., Hsu, K., Huang, J., Icard, T., Jain, S., Jurafsky, D., Kalluri, P., Karamcheti, S., Keeling, G., Khani, F., Khattab, O., Koh, P. W., Krass, M., Krishna, R., Kuditipudi, R., Kumar, A., Ladhak, F., Lee, M., Lee, T., Leskovec, J., Levent, I., Li, X. L., Li, X., Ma, T., Malik, A., Manning, C. D., Mirchandani, S., Mitchell, E., Munyikwa, Z., Nair, S., Narayan, A., Narayanan, D., Newman, B., Nie, A., Niebles, J. C., Nilforoshan, H., Nyarko, J., Ogut, G., Orr, L., Papadimitriou, I., Park, J. S., Piech, C., Portelance, E., Potts, C., Raghunathan, A., Reich, R., Ren, H., Rong, F., Roohani, Y.,

Ruiz, C., Ryan, J., Ré, C., Sadigh, D., Sagawa, S., Santhanam, K., Shih, A., Srinivasan, K., Tamkin, A., Taori, R., Thomas, A. W., Tramèr, F., Wang, R. E., Wang, W., Wu, B., Wu, J., Wu, Y., Xie, S. M., Yasunaga, M., You, J., Zaharia, M., Zhang, M., Zhang, T., Zhang, X., Zhang, Y., Zheng, L., Zhou, K., and Liang, P. (2022). On the opportunities and risks of foundation models.

[9] Brown, T. B., Mann, B., Ryder, N., Subbiah, M., Kaplan, J., Dhariwal, P., Neelakantan, A., Shyam, P., Sastry, G., Askell, A., Agarwal, S., Herbert-Voss, A., Krueger, G., Henighan, T., Child, R., Ramesh, A., Ziegler, D. M., Wu, J., Winter, C., Hesse, C., Chen, M., Sigler, E., Litwin, M., Gray, S., Chess, B., Clark, J., Berner, C., McCandlish, S., Radford, A., Sutskever, I., and Amodei, D. (2020). Language models are few-shot learners.

[10] Burns, C. (2022). How "discovering latent knowledge in language models without supervision" fits into a broader alignment scheme.

[11] Burns, C., Ye, H., Klein, D., and Steinhardt, J. (2022). Discovering latent knowledge in language models without supervision.

[12] Chiang, W.-L., Li, Z., Lin, Z., Sheng, Y., Wu, Z., Zhang, H., Zheng, L., Zhuang, S., Zhuang, Y., Gonzalez, J. E., Stoica, I., and Xing, E. P. (2023). Vicuna: An open-source chatbot impressing GPT-4 with 90%* ChatGPT quality.

[13] Christiano, P., Cotra, A., and Xu, M. (2021). Eliciting latent knowledge: How to tell if your eyes deceive you. Technical report, Alignment Research Center.

[14] Christiano, P., Leike, J., Brown, T. B., Martic, M., Legg, S., and Amodei, D. (2023). Deep reinforcement learning from human preferences.

[15] Clark, C., Lee, K., Chang, M.-W., Kwiatkowski, T., Collins, M., and Toutanova, K. (2019). BoolQ: Exploring the surprising difficulty of natural yes/no questions.

[16] Dai, A. M. and Le, Q. V. (2015). Semi-supervised sequence learning.

[17] Gao, L., Tow, J., Biderman, S., Black, S., DiPofi, A., Foster, C., Golding, L., Hsu, J., McDonell, K., Muennighoff, N., Phang, J., Reynolds, L., Tang, E., Thite, A., Wang, B., Wang, K., and Zou, A. (2021). A framework for few-shot language model evaluation.

[18] Howard, J. and Ruder, S. (2018). Universal language model fine-tuning for text classification.

[19] Hu, E. J., Shen, Y., Wallis, P., Allen-Zhu, Z., Li, Y., Wang, S., Wang, L., and Chen, W. (2021). LoRA: Low-rank adaptation of large language models.

[20] Huang, C.-Z. A., Vaswani, A., Uszkoreit, J., Shazeer, N., Simon, I., Hawthorne, C., Dai, A. M., Hoffman, M. D., Dinculescu, M., and Eck, D. (2018). Music transformer.

[21] Ippolito, D., Kriz, R., Kustikova, M., Sedoc, J., and Callison-Burch, C. (2019). Comparison of diverse decoding methods from conditional language models.

[22] Jelinek, F. and Mercer, R. L. (1980). Interpolated estimation of Markov source parameters from sparse data. In *Proceedings of the Workshop on Pattern Recognition in Practice*, Amsterdam, The Netherlands: North-Holland.

[23] Ji, Z., Lee, N., Frieske, R., Yu, T., Su, D., Xu, Y., Ishii, E., Bang, Y. J., Madotto, A., and Fung, P. (2023). Survey of hallucination in natural language generation. *ACM Computing Surveys*, 55(12):1–38.

[24] Khashabi, D., Kordi, Y., and Hajishirzi, H. (2022). UnifiedQA-v2: Stronger generalization via broader cross-format training.

[25] Khashabi, D., Min, S., Khot, T., Sabharwal, A., Tafjord, O., Clark, P., and Hajishirzi, H. (2020). UnifiedQA: Crossing format boundaries with a single QA system.

[26] Kingma, D. P. and Ba, J. (2017). Adam: A method for stochastic optimization.

[27] Lehmann, J., Isele, R., Jakob, M., Jentzsch, A., Kontokostas, D., Mendes, P. N., Hellmann, S., Morsey, M., van Kleef, P., Auer, S., and Bizer, C. (2015). DBpedia – a large-scale, multilingual knowledge base extracted from wikipedia. *Semantic Web*, 6(2):167–195.

[28] Levinstein, B. A. and Herrmann, D. A. (2023). Still no lie detector for language models: Probing empirical and conceptual roadblocks.

[29] Lin, S., Hilton, J., and Evans, O. (2022). TruthfulQA: Measuring how models mimic human falsehoods.

[30] Maas, A. L., Daly, R. E., Pham, P. T., Huang, D., Ng, A. Y., and Potts, C. (2011). Learning word vectors for sentiment analysis. In *Proceedings of the 49th Annual Meeting of the Association for Computational Linguistics: Human Language Technologies - Volume 1*, HLT '11, page 142–150, USA. Association for Computational Linguistics.

[31] McAuley, J. and Leskovec, J. (2013). Hidden factors and hidden topics: Understanding rating dimensions with review text. In *Proceedings of the 7th ACM Conference on Recommender Systems*, RecSys '13, page 165–172, New York, NY, USA. Association for Computing Machinery.

[32] Mostafazadeh, N., Roth, M., Louis, A., Chambers, N., and Allen, J. (2017). LSDSem 2017 shared task: The story cloze test. In *Proceedings of the 2nd Workshop on Linking Models of Lexical, Sentential and Discourse-level Semantics*, pages 46–51, Valencia, Spain. Association for Computational Linguistics.

[33] Ngo, R., Chan, L., and Mindermann, S. (2023). The alignment problem from a deep learning perspective.

[34] OpenAI (2023). GPT-4 technical report.

[35] Ouyang, L., Wu, J., Jiang, X., Almeida, D., Wainwright, C. L., Mishkin, P., Zhang, C., Agarwal, S., Slama, K., Ray, A., Schulman, J., Hilton, J., Kelton, F., Miller, L., Simens, M., Askell, A., Welinder, P., Christiano, P., Leike, J., and Lowe, R. (2022). Training language models to follow instructions with human feedback.

[36] Platt, J. (2000). Probabilistic outputs for support vector machines and comparisons to regularized likelihood methods. *Adv. Large Margin Classif.*, 10.

[37] Radford, A., Narasimhan, K., Salimans, T., and Sutskever, I. (2018). Improving language understanding by generative pre-training.

[38] Radford, A., Wu, J., Child, R., Luan, D., Amodei, D., and Sutskever, I. (2019). Language models are unsupervised multitask learners.

[39] Raffel, C., Shazeer, N., Roberts, A., Lee, K., Narang, S., Matena, M., Zhou, Y., Li, W., and Liu, P. J. (2020). Exploring the limits of transfer learning with a unified text-to-text transformer.

[40] Rajpurkar, P., Zhang, J., Lopyrev, K., and Liang, P. (2016). SQuAD: 100,000+ questions for machine comprehension of text.

[41] Reynolds, L. and McDonell, K. (2021). Prompt programming for large language models: Beyond the few-shot paradigm.

[42] Roemmele, M., Bejan, C., and Gordon, A. (2011). Choice of plausible alternatives: An evaluation of commonsense causal reasoning. *2011 AAAI Spring Symposium Series*.

[43] Roger, F. (2023). What discovering latent knowledge did and did not find.

[44] Sanh, V., Webson, A., Raffel, C., Bach, S. H., Sutawika, L., Alyafeai, Z., Chaffin, A., Stiegler, A., Scao, T. L., Raja, A., Dey, M., Bari, M. S., Xu, C., Thakker, U., Sharma, S. S., Szczechla, E., Kim, T., Chhablani, G., Nayak, N., Datta, D., Chang, J., Jiang, M. T.-J., Wang, H., Manica, M., Shen, S., Yong, Z. X., Pandey, H., Bawden, R., Wang, T., Neeraj, T., Rozen, J., Sharma, A., Santilli, A., Fevry, T., Fries, J. A., Teehan, R., Bers, T., Biderman, S., Gao, L., Wolf, T., and Rush, A. M. (2022). Multitask prompted training enables zero-shot task generalization.

[45] Schulman, J., Wolski, F., Dhariwal, P., Radford, A., and Klimov, O. (2017). Proximal policy optimization algorithms.

[46] Sennrich, R., Haddow, B., and Birch, A. (2016). Neural machine translation of rare words with subword units.

[47] Shaw, P., Uszkoreit, J., and Vaswani, A. (2018). Self-attention with relative position representations.

[48] Stiennon, N., Ouyang, L., Wu, J., Ziegler, D. M., Lowe, R., Voss, C., Radford, A., Amodei, D., and Christiano, P. (2022). Learning to summarize from human feedback.

[49] Taori, R., Gulrajani, I., Zhang, T., Dubois, Y., Li, X., Guestrin, C., Liang, P., and Hashimoto, T. B. (2023). Stanford alpaca: An instruction-following LLaMA model. https://github.com/tatsu-lab/stanford_alpaca.

[50] Touvron, H., Lavril, T., Izacard, G., Martinet, X., Lachaux, M.-A., Lacroix, T., Rozière, B., Goyal, N., Hambro, E., Azhar, F., Rodriguez, A., Joulin, A., Grave, E., and Lample, G. (2023a). LLaMA: Open and efficient foundation language models.

[51] Touvron, H., Martin, L., Stone, K., Albert, P., Almahairi, A., Babaei, Y., Bashlykov, N., Batra, S., Bhargava, P., Bhosale, S., Bikel, D., Blecher, L., Ferrer, C. C., Chen, M., Cucurull, G., Esiobu, D., Fernandes, J., Fu, J., Fu, W., Fuller, B., Gao, C., Goswami, V., Goyal, N., Hartshorn, A., Hosseini, S., Hou, R., Inan, H., Kardas, M., Kerkez, V., Khabsa, M., Kloumann, I., Korenev, A., Koura, P. S., Lachaux, M.-A., Lavril, T., Lee, J., Liskovich, D., Lu, Y., Mao, Y., Martinet, X., Mihaylov, T., Mishra, P., Molybog, I., Nie, Y., Poulton, A., Reizenstein, J., Rungta, R., Saladi, K., Schelten, A., Silva, R., Smith, E. M., Subramanian, R., Tan, X. E., Tang, B., Taylor, R., Williams, A., Kuan, J. X., Xu, P., Yan, Z., Zarov, I., Zhang, Y., Fan, A., Kambadur, M., Narang, S., Rodriguez, A., Stojnic, R., Edunov, S., and Scialom, T. (2023b). Llama 2: Open foundation and fine-tuned chat models.

[52] Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A. N., Kaiser, L., and Polosukhin, I. (2017). Attention is all you need.

[53] von Werra, L., Belkada, Y., Tunstall, L., Beeching, E., Thrush, T., and Lambert, N. (2020). Trl: Transformer reinforcement learning. https://github.com/lvwerra/trl.

[54] Wang, A., Singh, A., Michael, J., Hill, F., Levy, O., and Bowman, S. R. (2019). GLUE: A multi-task benchmark and analysis platform for natural language understanding.

[55] Webster, J. J. and Kit, C. (1992). Tokenization as the initial phase in NLP. In *Proceedings of the 14th Conference on Computational Linguistics - Volume 4*, COLING '92, page 1106–1110, USA. Association for Computational Linguistics.

[56] Zhang, X., Zhao, J., and LeCun, Y. (2015). Character-level convolutional networks for text classification. In *Proceedings of the 28th International Conference on Neural Information Processing Systems - Volume 1*, NIPS'15, page 649–657, Cambridge, MA, USA. MIT Press.

[57] Zhang, Z., Zhang, A., Li, M., and Smola, A. (2022). Automatic chain of thought prompting in large language models.

[58] Zheng, L., Chiang, W.-L., Sheng, Y., Zhuang, S., Wu, Z., Zhuang, Y., Lin, Z., Li, Z., Li, D., Xing, E. P., Zhang, H., Gonzalez, J. E., and Stoica, I. (2023). Judging LLM-as-a-judge with MT-bench and chatbot arena.

[59] Zhuang, F., Qi, Z., Duan, K., Xi, D., Zhu, Y., Zhu, H., Xiong, H., and He, Q. (2020). A comprehensive survey on transfer learning.

Appendix A

Using DLK Methods to Build Reward Models

A.1 Comparing Reward Model Versions

In this part of the appendix, we compare the UNIFIEDQA and theUNIFIEDQA-V2 models. The results are presented in table A.1. As we can see, the second iteration of the models is less performant only on the AG_News dataset, but improves on COPA, PIQA and QNLI. The two models perform on par on the other datasets.

Table A.1 Comparison of the UNIFIEDQA and UNIFIEDQA-V2 models on various datasets. The models were trained using 1,000 data points for both training and evaluation. The AUROC metric is used.

Dataset	UNIFIEDQA	UNIFIEDQA-V2
AG_News	0.855	0.637
Amazon_polarity	0.978	0.982
BOOLQ	0.939	0.936
COPA	0.972	0.993
DBpedia_14	0.989	0.988
IMDB	0.949	0.956
PIQA	0.781	0.857
QNLI	0.670	0.918
RTE	0.873	0.886

A.2 Custom Templates

A.2.1 The Full List of "Truthfulness" Templates

Here we present all of the templates that we used with our custom "truthfulness" datasets. Note that we refer to the text whose truthfulness will be evaluated simply as `<text>`. The possible answer choices used to form the contrast pairs are presented in *italics*.

- Template 1

> Consider the text below in triple backticks:
> ```
> <text>
> ```
>
> Question: is the given text truthful? Answer either yes or no.
> Answer: *yes / no*

- Template 2

> Question: is the text given below in triple backticks truthful? Answer either yes or no.
> ```
> <text>
> ```
> Answer: *yes / no*

- Template 3

> ```
> <text>
> ```
>
> Question: is the text given above in triple backticks truthful? Answer either truthful or deceitful.
> Answer: *truthful / deceitful*

- Template 4

<text>
The text given above is an example of text that is truthful. *Yes / No*

- Template 5

<text>
Out of deceptive or truthful, the text given above is *truthful / deceitful*

A.2.2 The Template for the QNLI Dataset

This part displays the prompt that we used to prompt the Vicuna model [12] for the QNLI dataset [54]. The *system prompt* (the first two sentences) and the `USER` and `ASSISTANT` prefixes are required to be added to the prompt because this format was used when the model was trained.

A chat between a curious user and an artificial intelligence assistant. The assistant gives helpful, detailed, and polite answers to the user's questions. USER: Consider the sentence below in triple backticks and the corresponding question. Does the sentence contain enough information to answer the question? Your answer should be either yes or no.

Desired format:
Answer: <your_answer>
Do not print "Answer:" again, just what you think the answer is.

Sentence:
```
<sentence>
```
Question: <question>
Answer: ASSISTANT: *No / Yes*

A.3 Results on the Individual Custom Datasets

This section presents the comparison of probe performance on the individual custom datasets. The results are shown in table A.2. We find that the VINC probe performs best on the custom IMDB dataset and achieves an AUROC value of 0.740. Meanwhile, the supervised logistic

Table A.2 Performance of different probes on the individual custom datasets. The UNIFIEDQA-V2 3B model is used since it is the one that we used for RL fine-tuning. The models were trained using 1,500 data points for both training and evaluation. The AUROC metric is used for comparison. We stress again that all of the datasets below are our custom versions of the original datasets.

Custom dataset	VINC	LR
AG_News	0.565	0.947
Amazon_polarity	0.606	0.970
BOOLQ	0.680	0.902
COPA	0.530	0.907
DBpedia_14	0.548	0.982
IMDB	**0.740**	0.951
PIQA	0.499	0.655
QNLI	0.525	0.807
RTE	0.595	0.845

regression probe seems to work well on many datasets, with the 3rd best result achieved on IMDB.

A.4 Inspecting the Reward Distributions for Unsupervised Probes

Here we extend the analysis from section 3.3 to the unsupervised VINC and CCS probes. Figure A.1 illustrates our findings when using the UNIFIEDQA-V2 11B model with a total of 2,048 samples from the custom IMDB RL training dataset. While the unsupervised probes were observed to perform well at classifying truthful samples during training (see table 3.5), they do not seem to generalize effectively, failing to give reasonable rewards for RL training samples. The top part of the figure depicts the VINC rewards, showing minor peaks around 0 and 1 for both correct and incorrect responses. In contrast, the CCS probe rewards are much more concentrated around 0 and 1, but the separation between the rewards for correct and incorrect responses remains poor. This lack of discrimination could explain why we were able to achieve successful results with RL fine-tuning only when we switched to supervised probes.

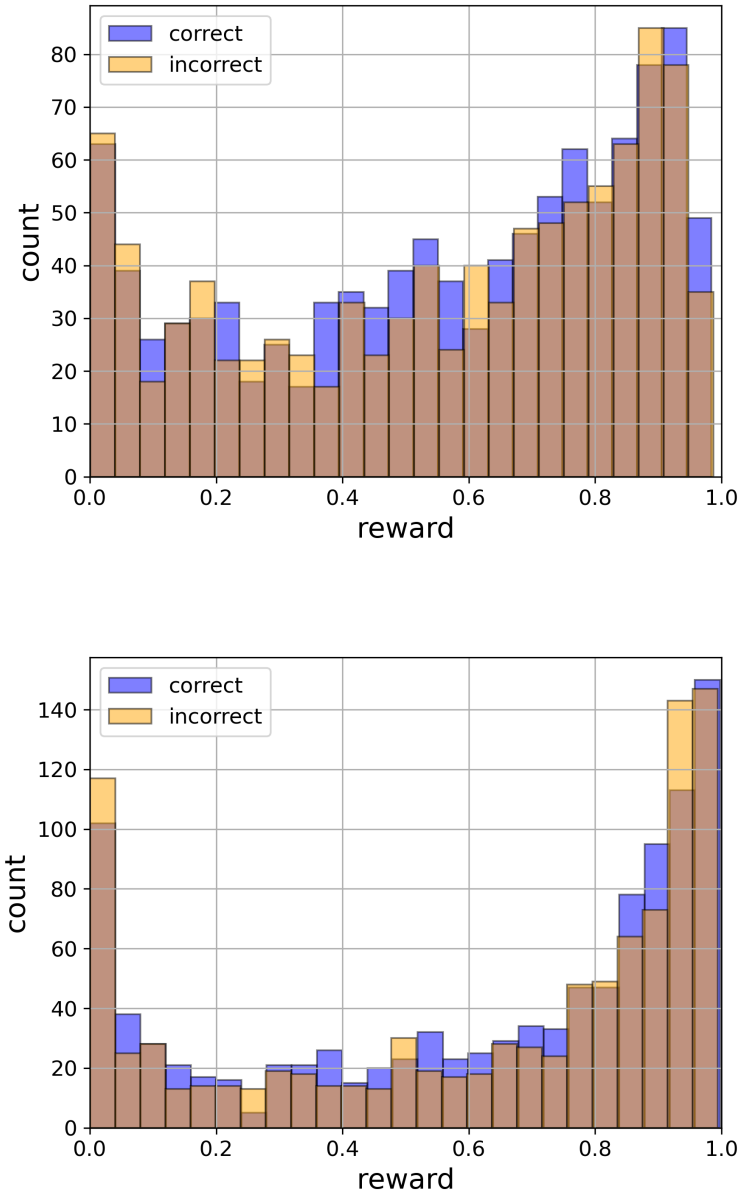


Fig. A.1 The reward distributions for the unsupervised probes built on top of the UNIFIEDQA-V2 11B model, using a total of 2,048 samples from the custom IMDB RL training dataset. The responses were synthetically obtained by taking the "golden" responses and randomly flipping about 50% of labels. The brown bars indicate an overlap between the two classes. *(top)* The rewards for the VINC probe. *(bottom)* The rewards for the CCS probe.

Appendix B

Fine-tuning Using Reinforcement Learning

B.1 Details of the Unsuccessful Experiments with the GPT-2 XL Model

Here, we provide a detailed examination of our experiments with the GPT-2 XL model. While not yielding successful results, they provided insights that allowed us to arrive at the final successful setup.

Combined dataset with a VINC probe

We first tried performing RL fine-tuning by using the combined dataset and a 3B reward model with an unsupervised VINC probe. As a reminder, the probe achieved an AUROC score of 0.625 (see table 3.5). We set the batch size to 32, the learning rate to 1.4e-5, 1e-5 or 5e-6, the number of steps to either 198 or 384 (which, considering that we had approximately 8,500 data points and used distributed data parallel training, amounted to 6 and 12 epochs over the data, respectively), and the rest of the values to defaults.

The results of the training runs are illustrated in figure B.1. The outcomes were not as anticipated, as even the best-performing run (depicted in green) only achieved a reward that was worse than random guessing. Additionally, our best model demonstrated only a marginal improvement of 0.040 over the baseline of 0.451.

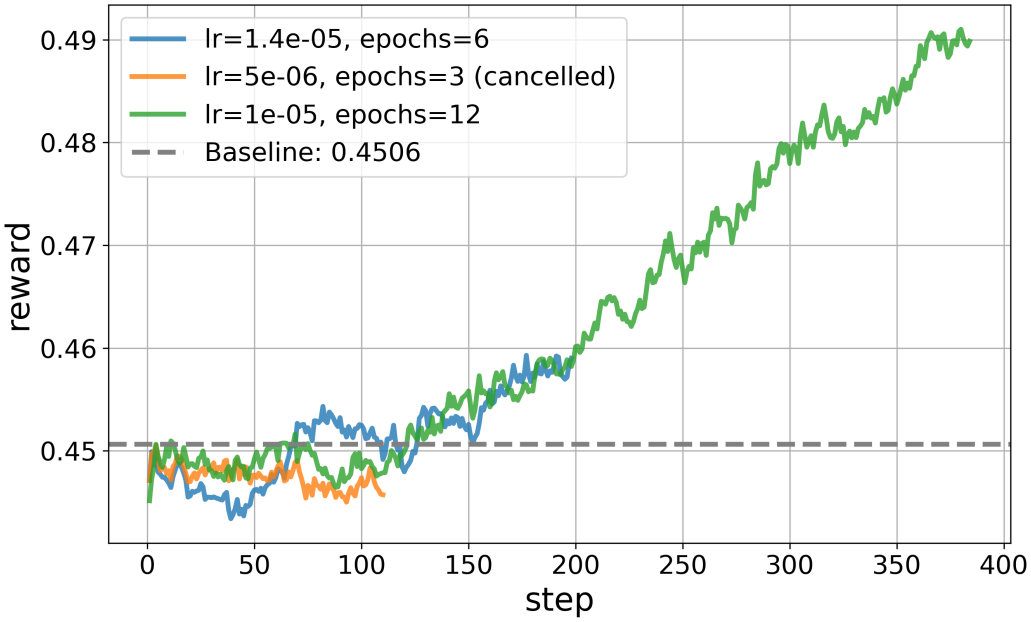


Fig. B.1 RL fine-tuning on the *combined* dataset using the GPT-2 XL as the policy model and a UNIFIEDQA-V2 3B model with a VINC probe as the reward model. An EMA smoothing with a coefficient of 0.8 is applied. The training run depicted in orange was terminated early due to an observed lack of improvement.

Custom IMDB dataset with a VINC probe

With the motivation to look for easier datasets and more performant probes, we also try using the custom IMDB dataset with a VINC probe. Again, we used the UNIFIEDQA-V2 3B as the backbone of our reward model. For these experiments, we used a batch size of 32, a learning rate of 1.4e-5 or 1e-5 and set the number of training steps to 384. This amounts to 12 epochs over the sample of 8,192 data points out of a total of 15,000 available for RL fine-tuning.

Figure B.2 presents the RL fine-tuning results. We find that the reward has improved compared to the combined dataset and the best training run now reaches a reward of 0.540 which is above random guessing. We also notice a higher improvement of about 0.070 over the baseline reward of 0.466 compared to before.

This prompted us to evaluate on the original IMDB dataset. Although we had trained the probes using all available prompt templates for this dataset (see section 3), we selected only 4 out of 13 templates to be used for RL fine-tuning. These particular templates were chosen because they contained the possible answer choices within them, a feature we hoped would guide the policy models towards generating desired answers more frequently. Consequently, we used the same templates for both training and evaluation to ensure consistency. Multiple

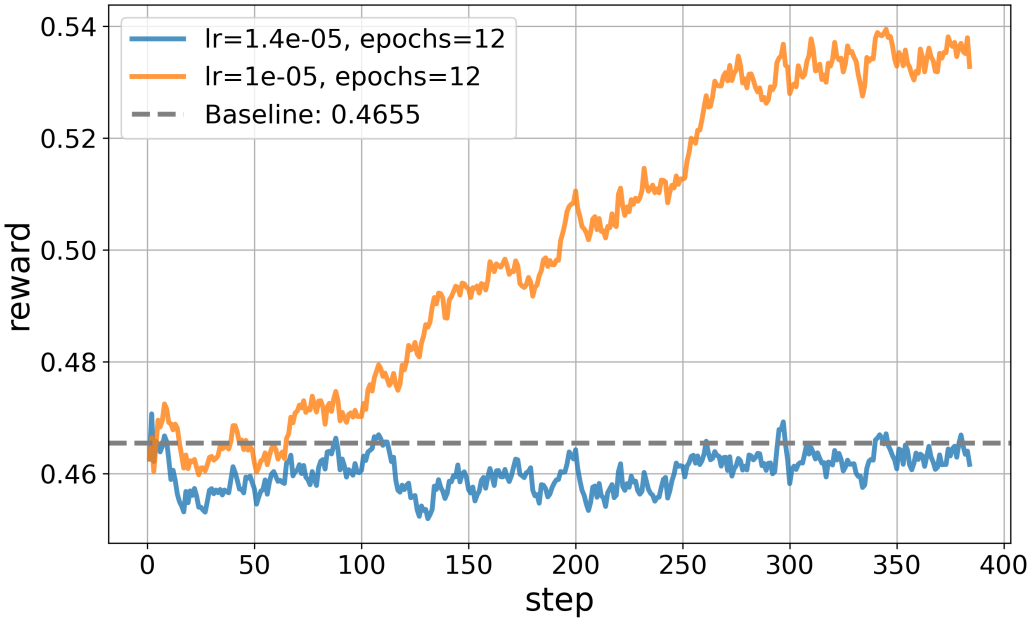


Fig. B.2 RL fine-tuning on the custom IMDB dataset. The policy model is GPT-2 XL and the reward model is a UNIFIEDQA-V2 3B model with a VINC probe. An EMA smoothing with a coefficient of 0.8 is applied.

versions of the dataset were formed using the different templates, and we averaged the accuracies for each template to arrive at the final results. As depicted in table B.1, RL fine-tuning provided only a relatively modest improvement of 0.9%.

Table B.1 Comparison of the accuracy of the GPT-2 XL models on the IMDB dataset before and after RL fine-tuning. The zero-shot accuracy is measured. "Vanilla" refers to the original pre-trained model, and "LR" stands for the logistic regression probe being used.

Model	Checkpoint	IMDB
Majority voting	–	50.0
Vanilla	–	52.9
GPT-2 XL + VINC RL	6/12 epochs	52.6
GPT-2 XL + VINC RL	12/12 epochs	**53.8**
GPT-2 XL + LR RL	4/12 epochs	51.1
GPT-2 XL + LR RL	8/12 epochs	51.7
GPT-2 XL + LR RL	12/12 epochs	51.8

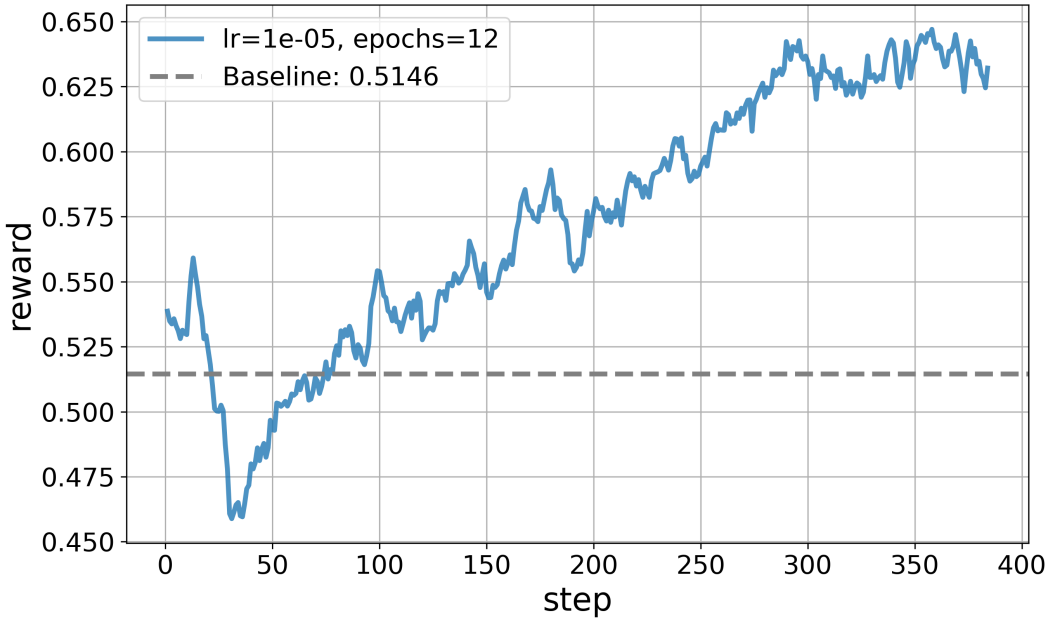


Fig. B.3 RL fine-tuning on the IMDB dataset. The policy model is GPT-2 XL and the reward model is a UNIFIEDQA-V2 3B model with a logistic regression probe. An EMA smoothing with a coefficient of 0.8 is applied.

Custom IMDB dataset with a logistic regression probe

Finally, we conducted an experiment with a supervised probe on the custom IMDB dataset. We set the batch size to 32, the learning rate to 1e-5 and trained for 384 steps (12 epochs over 8,192 data points). As a reminder, the reward model reached an AUROC value of 0.969 for this dataset (see table 3.5). As illustrated in figure B.3, the performance appears to have improved even further, with the model now reaching a reward of more than 0.625. The improvement over the threshold of 0.515 is now at least 0.11, which is also an improvement over previous results. However, the results of evaluating on the original IMDB dataset were disappointing, as all the models' performance appeared to have deteriorated during fine-tuning (see Table B.1).

As discussed in section 4.1, the GPT-2 XL policy models fail to generate coherent responses for our custom datasets. We hypothesize that this lack of capability is the reason why the models trained with a supervised probe underperformed compared to the original model or the one trained with a VINC probe. In both cases, whether trained with supervised or unsupervised probes, the models produced incomprehensible outputs, and neither type of probe was capable of accurately assessing the correctness of these irrational generations.

However, the supervised probes produced rewards that were closer to the extremes of 0 or 1, which resulted in a more pronounced reinforcement of the irrational behaviour compared to when the unsupervised probes were used.

B.2 Prompt Sensitivity

Table B.2 Comparison of the models' zero-shot accuracy on the original QNLI dataset before and after RL fine-tuning using the original prompt. The results show that the performance still improves, but much more modestly, indicating that the models remain sensitive to prompting.

Model	QNLI
7B-v1.3	55.8%
7B-v1.3 + RL	**60.3%**
7B-v1.5	68.5%
7B-v1.5 + RL	**70.5%**