Degree Project in Computer Science and Engineering

First cycle, 30 credits

Optimizing a Swedish Distractor Generator for Multiple-Choice Questions using AI Preferences

ELIAS LUNDGREN

Optimizing a Swedish Distractor Generator for Multiple-Choice Questions using AI Preferences

ELIAS LUNDGREN

Degree Programme in Computer Science and Engineering
Date: June 16, 2024

Supervisor: Johan Boye
Examiner: Olov Engwall
School of Electrical Engineering and Computer Science
Swedish title: Optimering av en Svensk Distraktorgenerator för Flersvarsfrågor med AI-Preferenser

Abstract

Multiple-Choice Questions (MCQs) are common in educational settings to test and improve reading comprehension. Generating full MCQs, or just the incorrect answer options (called distractors), can help teachers spend their time more efficiently. Both of these topics have been studied, but there is a notable gap in research concerning languages other than English.

In this thesis, a distractor generator for Swedish MCQs is developed and compared against a previously introduced BERT model. An extension of the dataset used by the BERT model is used to train Mistral 7B, a newer Transformer-based model using state-of-the-art techniques like QLoRA and DPO on consumer-grade hardware.

Human feedback is often used to evaluate and further train models while avoiding the need to formulate complex goals or modify training data. With the introduction of widely available LLMs like ChatGPT that can solve intricate tasks, one goal was to evaluate whether this technology could efficiently replace human feedback with AI feedback without impacting output quality.

The final DPO trained Mistral 7B model scored similarly to the baseline with two teachers approving $\approx 50\%$ of its distractors, while also giving some valuable insights on possible future improvements. Performing DPO using AI preference data resulted in some improvements. However, human preference data would most likely have yielded even better results. Compared to the baseline BERT model which was pre-trained only on Swedish data, Mistral 7B is primarily trained on English, which may be a second reason for the lackluster performance. Lastly, evaluators only accepted 61.6% of the included human-made distractors - revealing the dataset quality to be sub-par. Most likely this limited the quality of the final model. This could be avoided in the future by involving the evaluators in the dataset-making process.

The model, Swedish training and preference datasets, and evaluation scripts are open-sourced to provide a simple starting point for future work.

Keywords

Transformer Architecture, Multiple-Choice Question, Distractor Generation, Large Language Model, Direct Preference Optimization

Sammanfattning

Flersvarsfrågor är vanligt förekommande i utbildningssammanhang. De används bland annat i samband med examinationer, men även för att förbättra läsförståelse hos elever. Genom att generera hela flersvarsfrågor, eller enbart distraktorerna - de felaktiga svarsalternativen - kan lärare spara tid som de kan använda till viktigare uppgifter. Dessa två sätt att generera flersvarsfrågor har undersökts av andra forskare, men majoriteten har fokuserat på engelska flersvarsfrågor. Mindre vanliga språk som svenska har fått begränsad uppmärksamhet.

I denna avhandling utvecklas en distraktorgenerator för svenska flersvarsfrågor som jämförs med en tidigare introducerad BERT-modell med samma syfte. En expanderad version av BERT-modellens dataset används för att träna Mistral 7B – en nyare modell som också är byggd på transformerarkitekturen. Denna modell tränas därefter med hjälp av spjutspetsmetoder som QLoRA och DPO på allmänt tillgänglig hårdvara.

Mänsklig återkoppling används ofta för att utvärdera och förbättra modeller istället för att behöva artikulera komplexa mål eller redigera träningsdata. Med lanseringen av lättillgängliga samtalsinriktade stora språkmodeller som ChatGPT, har det blivit möjligt att automatisera krångliga textbaserade uppgifter. Ett mål i denna avhandling är att undersöka om mänsklig återkoppling kan ersättas med återkoppling från en AI, utan att påverka kvaliteten, när man skapar preferensdata.

Den slutgiltiga DPO-tränade Mistral 7B-modellen presterade på en liknande nivå sett till BERT-modellen och gav flertalet värdefulla insikter om framtida utvecklingspotential. Det sista träningssteget, som involverade DPO med AI-preferensdata, resulterade i små förbättringar. Mänsklig återkoppling bör ge ännu bättre resultat. Jämfört med BERT-modellen, som enbart har förtränats på svensk data, har Mistral 7B primärt förtränats på engelsk data. Detta är troligtvis en ytterligare anledning till det svaga resultatet. Utvärderarna godkände enbart 61.6% av distraktorerna som skapats av en människa, vilket påvisar att träningsdatan var av bristande kvalitet – något som säkerligen också påverkat kvaliteten på den slutgiltliga Mistral 7B-modellen. I framtida arbeten kan detta undvikas genom att involvera utvärderarna i processen av att skapa träningsdatan.

Modellen, den svenska tränings- och preferensdatan, samt koden för utvärdering och återkoppling delas som öppen källkod för att erbjuda en enkel utgångspunkt för framtida arbeten.

Nyckelord

Transformerarkitektur, Flersvarsfrågor, Distraktorgenerering, Stor Språkmodell, Direkt Preferensoptimering

Acknowledgments

I would like to thank my supervisor, Johan Boye, for his continued support and valuable feedback, as well as for the many interesting and insightful conversations.

Stockholm, June 2024
Elias Lundgren

Contents

Chapter 1

Introduction

The field of Artificial Intelligence (AI) has experienced rapid progress in recent times, with one of its pivotal advancements being the transformer architecture. This has led to many new exciting use cases for AI, like the well-known conversational model ChatGPT [1]. Large Language Models (LLMs) based on the transformer architecture may also be trained to perform more narrow tasks such as summarizing texts or generating Multiple-Choice Questions (MCQs).

MCQs are commonly used in educational settings to test and improve reading comprehension. Most of the time they consist of a context - some text to be read before to gain knowledge of the topic, as well as a question. To answer the question a set of possible answers are given to pick from - where one (or in some instances multiple) is the correct answer and the rest are distractors (incorrect answers). By generating whole MCQs, or portions of them, a teacher's valuable time can be spent elsewhere. When teachers know which questions to ask and their accompanying answers, it makes sense to only generate the distractors. An English reading comprehension MCQ with distractors ranging in quality could for example look like this:

Marie visited several European cities last summer. She loved the art and culture in Rome, enjoyed the vibrant nightlife in Berlin, and marveled at the historic sites in Madrid. However, her favorite city was Paris, the capital of France, where she spent hours exploring the Louvre and walking along the Seine.

What is the capital of France?

(a) Paris *(Correct Answer)*

(b) It's Paris *(Bad Distractor)*
(c) Germany *(Okay Distractor)*
(d) Stockholm *(Good Distractor)*
(e) Rome *(Great Distractor)*

As shown in this example, multiple factors like inconsistent grammar or containing the answer (It's Paris), may immediately disqualify a distractor. In more simple MCQs, grabbing the distractor from the context (Rome) is generally the best solution. Otherwise picking a word or phrase similar to the answer often works (Germany), and even better if it's more narrow (Stockholm).

The topic of distractor generation has received less attention, despite being the most demanding and time-intensive aspect of creating multiple-choice questions (MCQs) [2]. Some previous attempts based on the Transformer Architecture have performed well at this task, even using languages other than English and small-scale datasets [3].

Transformer models may be trained on grand corpora of text, in a process called pre-training to learn the linguistic properties of one, or multiple languages [4]. Thereafter, it is common to rely on human preferences to judge model output, and then use that preference data to refine the model's text generation to output to better align it with abstract goals and criteria.

Creating a dataset containing human preferences is generally a tedious task that takes many hours to do by hand. But luckily, recent AI breakthroughs might help once again. Conversational models like ChatGPT (GPT-3.5 Turbo) and its successor GPT-4 excel at logical and repetitive tasks - like creating datasets based on preferences. However, the degree to which this automation may compete with a human is still unclear, especially in a less common language like Swedish.

There is a clear correlation between the number of trainable parameters and the final model performance. This has made it nearly unfeasible to train many models without access to large data centers. Some researchers have made it a priority to focus on smaller, more efficient models, among them Mistral and their Mistral 7B model [5]. This model optimizes the transformer architecture to make it significantly outperform similarly sized models, and even compete with ones double its size. By relying on some of the state-of-the-art optimization techniques for both training and inference - a model of this size can be trained using consumer-grade hardware.

This thesis aims to further the research on Swedish distractor generation using [3] as a baseline and improving it. Starting with Mistral 7B, it is trained

to generate distractors using state-of-the-art techniques like Quantized Low-Rank Adaptation 2.6.2 and Direct Preference Optimization 2.6.4. A core part of this involves evaluating to which degree a LLM can replace a human in the process evaluating model outputs and creating a dataset containing its preferences. Additionally, this thesis aims to showcase how every step of the fine-tuning process can be performed using consumer-grade hardware, resulting in a model that can be run locally for cheap.

This requires some datasets and software to be built, available in 7. The datasets extend the SweQUAD-MC dataset introduced in [3] with some additional data collected by the same authors. And the software is an environment to rate distractors - either automatically with an LLM or manually using human input.

1.1 Research Questions

There are three distinct research questions which this thesis aims to answer:

1. To what extent can direct preference optimization, utilizing AI-generated preferences, enhance the accuracy and effectiveness of distractors generated by a LLM fine-tuned on a small-scale Swedish dataset for MCQs?

2. Can feedback from Large Language Models (LLMs) effectively replace human feedback in evaluating the quality of generated distractors for Swedish multiple-choice questions (MCQs), while maintaining high-quality standards?

3. Is it feasible to train a model for Swedish distractor generation, utilizing the latest techniques, without relying on enterprise-grade hardware, and what are the implications of this approach?

Chapter 2

Background

This chapter first introduces multiple-choice questions, their purpose and use cases in 2.1. The section also includes how full multiple-choice questions, but particularly their distractors can be generated using state-of-the-art language models to help teachers become more efficient - the purpose of this thesis. Thereafter the next section includes some general knowledge about language models is covered in 2.2, before deep diving into the current state-of-the-art, transformers, in the next section 2.3. How these transformer based models are trained to model a language is thereafter described in 2.4. In 2.5, some key implementations of the Transformer in language models are highlighted, as well as their improvements and impact on the field. Among them is Mistral 7B, a small but efficient model that's later used as a starting point in this project 3.1. The next section covers the process of fine-tuning, a key step in the creation process of a language model. This section includes state-of-the-art solutions to aligning a model's output to perform some task, like generating distractors. In the last section before the summary, 2.7, recent developments in fine-tuning models using preference data to further improve them are covered.

2.1 Multiple-Choice Questions

Assessing someone's knowledge on a topic can be done using multiple-choice questions. A well-designed MCQ contains several answer alternatives, one (or sometimes multiple) of which is correct, while the rest are distractors, incorrect but plausible. MCQs provide great feedback to instructors on knowledge gaps and can aid students in learning [6].

Testing someone's knowledge after reading a text, a process called active learning, has been shown to improve reading comprehension and learning,

and has been extra handy in online education. Additionally, MCQs are a great and efficient method from a teacher's perspective as grading can simply be automated. However manual construction of questions is very time-consuming, particularly when a large amount of questions are required. This can be both expensive for the employer, as well as draining for the teachers as it requires a high level of skill and thinking[2].

2.1.1 MCQ Generation

Substantial research has gone into using natural language generation to generate questions and answers from an input text [7]. Current state-of-the-art question generators are based built upon encoder/decoder models, often transformers [8]. These models have taken many shapes, but most involve some similar core steps like text preprocessing, question-answer generation and lastly distractor generation like in [8] and [9].

Publicly available large-scale conversational models like ChatGPT [10] have simplified the generation of multiple-choice questions. Because of the question-answering structure of ChatGPT, simply supplying a text and asking for a certain number of multiple-choice questions based on it is often sufficient, while simultaneously surpassing the quality of previous techniques. [11] compared 50 ChatGPT-generated MCQs with 50 MCQs drafted by university professors, and found no significant difference in quality between the two batches.

2.1.2 Distractor Generation

In some scenarios, the teacher wants more control in the generation process. A less studied area is distractor generation - where based on context, question, and answer - the model outputs a specified amount of distractors. This part is often seen as the most taxing [12], as it requires careful consideration of whether a distractor is reasonable, and still provides a challenge to the user. Using ChatGPT to generate distractors exceeds previous state-of-the-art solutions, with an average of 53% of the distractors being rated as high quality by teachers [13].

It is important to note that these evaluations were done in English, the language ChatGPT performs the best in, and solutions in other languages may result in worse distractors. This is logical as the majority of the training data, including the human feedback used during fine-tuning, was in English [1].

Using an API-based closed-source solution limits the applications for

distractor generation, and might also be somewhat ”overkill” for this simple task. Previous research has shown that smaller open-source models can perform similarly. T5-small, a 220M parameter model by Google that was fine-tuned on the RACE dataset managed to reach an average one-gram BLEU score of 37 on its distractors [8]. RACE is a large-scale multiple-choice question dataset with nearly 100,000 questions, taken from English examinations in China [14].

BLEU is short for Bilingual Evaluation Understudy. It is an evaluation metric built for evaluating text quality of machine-translated text [15]. But it and other similar metrics like ROGUE [16], have also found a use case for comparing the likeness of sentences.

The multilingual version on T5, mT5, was fine-tuned to generate distractors in Spanish by using a machine-translated version of RACE to great results when evaluated using 1-4 n-gram BLEU, ROUGE and cosine-similarity [17]. The latter being a direct evaluation of the distance between two different answers' high-dimensional vector embeddings. The authors argue this is a better evaluation metric than BLEU and ROGUE. There have been multiple articles directing critique towards the RACE dataset for being low quality. For example [18], where the authors showcase multiple severe problems with the dataset and rank the different questions based on quality.

Previously Kalpachi & Boye showcased how SweQUAD-MC, a small scale, Swedish dataset could be used to fine-tune a BERT-model pre-trained on Swedish [3]. The dataset contains a context - which should be read before answering the question, the question and the correct answer - as well as three distractors created by a human. Additionally, they introduce a list of criteria for manually crafting, or generating a great distractor - and thereafter using humans to evaluate their final model.

2.1.3 Distractor Evaluation

When evaluating distractors semantic closeness described by evaluation metrics like BLEU, ROUGE or cosine similarity may give some indication of their quality. However, since their introduction, there have been many critiques against their use cases as general evaluation metrics within the field of machine learning.

BLEU has been shown to only correlate with human judgments in specific tasks and settings, and often diverges a lot. Sometimes BLEU gravely underestimates the quality of a great input sentence [19], and at other times can give an infuriatingly high score to low-quality input [20]. ROUGE has

faced similar critiques, and while there have been subsequent improvements to improve it [21], none of them match human judgment.

Evaluation still remains the most accurate when relying on human professionals. In the evaluation of the final model in [3] the authors evaluated their multiple-choice questions on teachers and students, and relied on carefully selected quantitative metrics for the specific purpose of generating grammatically correct distractors. The article also introduces some of the more common reasons for a distractor being rejected by a teacher, accompanied by examples [3].

For each reason, an example of a bad distractor is shown. The example question is *How tall is the boy?* and the answer to the example is *190 cm*.

Not Wrong, the distractor is correct in addition to the answer, either because it means the same thing as the answer, contains the answer, or is simply another correct answer. *Example Distractor: 1.90 m*

Unreasonable, the distractor makes no sense in the context of the question. A distractor like this is too easy for a student to disregard, and thus cannot be included. *Example Distractor: 20 tablespoons*

Grammatically Wrong, the distractor simply doesn't answer the question with the correct grammar or structure. *Example Distractor: is 180 cm*

Identical, the distractor is the same as the answer, and for that reason cannot be included. *Example Distractor: 190 cm*

Obviously Wrong, the distractor makes a claim that the reader knows is incorrect. *Example Distractor: 360cm*

As shown in this section, distractor generators make use of state-of-the-art language models to achieve great results and ultimately help teachers become more efficient.

2.2 Language Models

Since the dawn of Natural Language Processing (NLP) - the ability for a computer to support and manipulate human language - language modeling

has been a integral part to the field. Language modeling in itself is a broad category, containing a variety of tasks such as: speech recognition, machine translation and natural language generation to name a few.

In their early days language models were purely statistical and often relied on n-grams - models that assume that the next word in a sentence can be decided using the probabilities of a fixed size window of n preceding words.

The statistical models were later superseded by neural networks, built upon layers of nodes to be trained and thereby learn to perform complex tasks. These models quickly grew in size in respect the amount of training data and trainable parameters. The term Large Language Model (LLM) was given to models with many millions, or even billions of parameters. Today, most new LLMs are built using the transformer architecture.

2.3 Transformers

Long-Short Term Memory (LSTM) and other Recurrent Neural Networks (RNNs) were previously seen as the state-of-the-art solutions sequence-to-sequence tasks like language modeling and machine translation since the 90s [22]. RNNs rely on a hidden state h_t, that for each step forward relies on the previous hidden state h_{t-1} [23]. This is sequential in nature, and even though improvements using factorization [24] propelled the technology forward, this constraint remained a performance bottleneck.

Attention mechanisms had previously been used in conjunction with RNNs to better capture long-range dependencies between the encoder and decoder. But the Transformer Architecture [25], visualized in 2.1, changed this status quo by removing it and instead focusing on an attention mechanism in combination with other parallelizeable components.

For readers new to the transformer model, see [26] which explains the architecture from the ground up in an easy to understand manner.

2.3.1 Encoder / Decoder Architecture

Transformers and RNNs are based on encoders and decoders. The encoder is used to process a sequence of input tokens to capture the semantic and syntactic relationships tokens within the sequence. Tokens are words, or small parts of them, which together make up a sentence (or sequence). The decoder thereafter relies on the calculations done on the input and previously generated tokens to generate the next token.

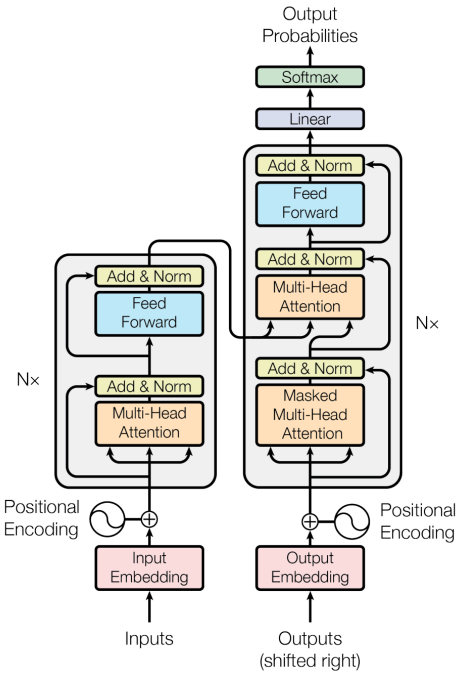


Figure 2.1: Visualization of the Transformer Architecture introduced in [25]

The original Transformer architecture relies on six identical layers in the encoder and decoder respectively, these are often referred to as encoder/decoder stacks (of layers). As input the encoder receives a sequence of tokens given by the user, and the decoder its previously generated tokens. The first step in both stacks is to convert the input tokens into vectors in high-dimensional space, commonly referred to as "embeddings". Because of the parallel nature of the transformer, it lacks the ability initially understand the order of tokens. Hence a step called positional encoding is required to connect the tokens in the order they are supposed to be read. Most common is left-to-right processing, but right-to-left, or any other way is also valid [25].

Once embedded and positionally encoded, a sequence is ready to pass through the different layers of the transformer. The sequence of token embeddings is represented as a 2D matrix of the dimensions Sequence tokens $\times$ Embedding dimension, and often referred to as the feature representation. As each token embedding already contains positional data - the columns which contain each token embedding are not ordered.

By interconnecting each encoder layer with a decoder layer, the next token generated by a transformer relies on both the user input and the previously generated tokens. Most models only rely on the decoder stack if the purpose

is to generate new text from left to right. Instead of giving the input sequence of tokens as input into the encoder, it is given to the decoder as preceding output. Without the encoder, the calculations required to generate the next token just rely on the previous output, which is all that is needed in left-to-right generation.

The encoder and decoder layers are structured into two sub-layers, an attention mechanism and a feedforward network. Both the layers also incorporate a residual connection and layer normalization after each sub-layer. This is often referred to as one add-and-normalize operation.

2.3.2 Attention Mechanism

An attention mechanism is a technique that enables models to focus on specific parts of the input data when performing a task [25]. For each token j in the input, the attention mechanism calculates a score for all the other tokens $k_{1...n}$ in the sequence k.

Example sequence: The cat caught a mouse because it was hungry.

To simplify, let us assume each word in this sentence is a token. If you were to calculate the attention that the token **it** gives to the other tokens in the sentence, you would see **The** and **Cat** score the highest. By thereafter adding the contextualized vectors of these tokens scaled by their attention score to the contextualized vector of **it**, a resemblance of understanding that **it** refers to **The cat** is created. It is worth noting that this is an over simplified description, and the attention calculation often is less intuitive to a human observer.

There are multiple ways to implement an attention algorithm. The original transformer implementation uses scaled dot-product attention, a variation of dot-product attention with a scaling parameter to avoid out of regions with extremely small gradients 2.4.1.

The attention calculation for each token relies on three vectors: Query (Q), Key (K) and Value (V) which are used to efficiently create a weighted sum output defining the attention of each token. These three vectors are created by multiplying the original token embedding with three distinct matrices which are produced during the training process of the model. The vectors Q and K have the size d_k and the V vector has the size d_v.

During the scoring process, the attention that token t_1 gives to the other tokens $t_2...n$ is calculated using the dot product between the query vector Q_1

of t_1 and the key vector K_j of the t_j for each j. This is where the scaling is applied, before passing the result through a softmax function resulting in some attention weights. By multiplying these weights with the Value vector for each token, a contextualized representation of the input sequence is created.

Multi-Head Attention is used instead of a single large-scale calculation of the size d_{model}. This process involves splitting Q, K and V using projection via different learned linear projections into $h = 8$ attention heads (in this implementation). These projections can then be processed in parallel by each head, which helps the calculation take a constant time [25].

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

With multi-head attention, the scaled dot product attention is calculated out over each attention head using the previously mentioned linear projections of Q, K and V. This leads to a total of 8 different matrices after the calculation is complete. As the other components in the architecture expects one matrix of the original input size, the results from every head are concatenated and then multiplied with a different learned weight matrix W_0 to return to the original size.

2.3.3 Add and Normalize

After each attention operation, an **Add-and-Normalize** operation is used, containing a residual connection and layer normalization. The residual connection combines the input x of an operation with the output of the same operation $F(x)$, resulting in $x + F(x)$ as the final output. This connection allows the deeper layers in each stack to gain information from all the previous layers. Without a connection like this, each layer after the first would only use the output $F(x)$ of the previous one. This technique reduces the vanishing gradient problem, a problem related to gradients becoming too small to improve a neural network during training [25].

After residual connection, the result is normalized by first calculating the mean and standard deviation of all contextualized vectors. Thereafter the mean is subtracted from the each vector which is then divided by the standard deviation. This results in the vectors becoming standardized, with a mean of 0 and a standard deviation of 1 before they are passed on to the next operation.

2.3.4 Feedforward Network

At the end of each layer, a **Feedforward Network** (FFN) applied independently to each contextualized vector. The FFN uses linear transformations with two unique weight matrices W_1, W_2 and biases b_1, b_2 on each layer of encoder and decoder stacks. It also has a Rectified Linear Unit (ReLU) activation function that introduces non-linearity in the relations created with the transformations by clamping the values to above or equal to zero. The equation for the FNN takes a vector x as input.

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

2.3.5 Output

After passing through the different layers of the encoder/decoder, there is still one key step remaining - converting the matrix of contextualized vectors back into text. This process first involves using another learned linear projection. The weight matrix used for this is the same as when embedding the input [25]. Thereafter this is converted into next token probabilities between 0 and 1 using a softmax function.

As showcased in this section, the transformer contains many weight matrices inside its different components. Starting, these matrices are generally initialized following some rules, but to properly model a language - they need to be trained on substantial amounts of data.

2.4 Training

In the previous section, the forward pass - the process of taking an input and transforming it into an output - has been covered. This process is often referred to as inference, which is the neural network making predictions based on what it has learned. Models based on neural architectures like RNNs or the Transformer thus require some training to learn things before they become useful. The training involves three main parts and is done using one data point at a time. After one or multiple data points have been used to update the weights and biases once - a **step** has been taken. When every data point has been used, the first **epoch** has been completed. It is normal to train LLMs for multiple epochs, thus iterating over the same data multiple times [27].

First a forward pass is used to generate next token probabilities for each token in the data point, starting with the first token to calculate the probabilities for the second - thereafter with the first two tokens to calculate probabilities for the third - and so on. This results in probability distributions $\hat{y}_i$ of the next token, for each token probability i. At the start of training, the tokens with the highest probabilities are likely very far from the real next token defined in the data point.

Second the probabilities are compared using a Loss function. The original transformer implementation uses cross-entropy loss to compare probabilities [25]. Here y_i contains a vector for each token i with the probability for the true next token is set to 1 and all other tokens in the vocabulary are set to 0 (this structure is called one-hot encoding). Where N is the vocabulary size of the model.

$$L(y, \hat{y}) = -\sum_{i=1}^{N} y_i \log(\hat{y}_i)$$

The Cross-Entropy Loss is calculated as the negative sum of the logarithm of predicted probability, but only the term for the correct class (where $y_i = 1$) contributes to the sum. Lastly, as a loss value for the entire sequence is required the average cross entropy loss is used [27].

Third the weights need to be adjusted to make the model generate a higher probability for the correct tokens. This is done using backpropagation, sometimes referred to as a backward-pass through the model.

2.4.1 Backpropagation (BP)

The initial backpropagation implementation was done on more simple neural networks, containing an input layer, hidden layers in the middle, and an output layer at the end. Each layer contains neurons, or units, all having input connections i from the previous layer and output connections j to the next layer. i and j typically index the neurons within the layers, rather than denoting the connections themselves. Specifically, i might index neurons in one layer, and j might index neurons in the subsequent layer with the weights w_{ji} representing the connections between the layers.

To adjust the weights, the first step of backpropagation is to figure out how a change in weights impact the loss. This is done by first calculating the gradient of the loss with respect to the output layer's weights, the Jacobian matrix, which indicates how much tweaking each weight impacts the loss. As these weights result from cascading effects from multiple layers of weights

and biases, modifications must account for this complexity. Now the process of propagating the gradient backwards begins, one layer at a time from the output back to the input layer. This is done by utilizing the chain rule in a recursive manner to determine the gradients of preceding layers. This process includes determining the gradients for the weights of the self-attention mechanisms, the weights of the FFNs, and any other parameters within the transformer that can be adjusted through learning. How much the weights are changed is tied to the learning rate, an initial number that's supplied when starting training, and often changed over time depending on the updating technique [27].

2.4.2 Updating

After the backpropagation is completed, the actual descent part of gradient descent is left. Depending on which algorithm is used, this can be done in multiple ways. Today, the most common gradient descent optimization algorithm is the **Adam Optimizer**, which combines a couple of extra features into the regular stochastic gradient descent algorithm (weights are updated after each step) [28]. The first of these additions is the Adaptive Gradient Algorithm which maintains a separate adaptive learning rate for each parameter, instead of a fixed learning rate for all. The second improvement is Root Mean Square Propagation, which also uses per-parameter learning rates and adapts them using recent (over multiple steps) magnitudes per weight.

The amount of epochs and steps varies greatly depending on the purpose of the training. But eventually the loss converges to a minimum or a user-specified limit of steps or epochs have been reached.

Since the introduction of the transformer in 2018, new training techniques and architectural changes have brought great improvements to the original implementation. Next, a few of the most important advancements are covered.

2.5 GPT, BERT and Mistral

The Transformer model rapidly changed the state-of-the-art of natural language processing over the next years, with new implementations quickly surpassing previous versions.

2.5.1 Generative Pre-trained Transformer

One of the first implementations was the Generative Pre-Trained Transformer (GPT). The authors present a semi-supervised learning approach for capturing both sentence and word level semantics [4]. The process involves a self-supervised pre-training step, and a supervised fine-tuning step with labeled data for adapting the learning to a specific task.

Previous efforts showcased the ability to improve text classification by following a similar method, but due to the usage of an LSTM model, long-range dependencies could not be captured well enough [29]. However, the authors in this paper chose the Transformer as their underlying model, which excels at this task. Leveraging pre-training on large, unlabeled text corpora, coupled with minimal supervised learning for fine-tuning, significantly reduces the dependency on extensive manual labeling of text, traditionally a labor-intensive necessity.

As shown by [30] there is no need for the encoder part of the Transformer for left-to-right text generation. Thus the authors only rely on the decoder. By initializing the decoder output to the user-given input, the correct context can be captured without needing the encoder.

The authors of [4] concluded that unsupervised pre-training was very effective, surpassing state of the art in 9 / 12 datasets, while highlighting the reasons behind its success are complicated and somewhat unknown.

2.5.2 Bidirectional Encoder Representations from Transformers

Building upon the pre-training discoveries made in [4] a bidirectional implementation was created [31]. The implementation takes advantage of the encoder and compared to GPT and other early implementations that relied on the decoder.

To train this new model, the authors introduce a Masked Language Model (MLM) pre-training objective. Instead of generating the next word based on a context - a [MASK] token may be inserted anywhere in a sentence - and by using all surrounding text predict the missing word. BERT is also pre-trained on another task, next sentence prediction, to improve results in downstream tasks such as question answering. This training involved answering whether sentence B follows sentence A, with 50% of B being correct, and the other half a random sentence from the text corpus. After pre-training is done, fine-tuning is necessary to fit the model to a particular task, for example question

answering. Because of the bidirectionality and next sentence prediction many training tasks can be very straight forward

The authors of BERT generalize the findings that extensive unsupervised pre-training also presents benefits for bidirectional architectures. Lastly, they conclude that increased model size - as in the amount of parameters - improves the models as a whole, but also on small-scale tasks.

2.5.3 Mistral 7B

If the goal is to run inference on consumer devices, in addition to being publicly available and open source, the model needs to remain small. While larger models often perform better on various tasks, they require extensive hardware and incur more inference latency.

Mistral 7B was introduced to showcase how a smaller model can be engineered for high performance while remaining efficient [5]. It contains two important optimizations to reduce latency and memory requirements.

Grouped Query Attention is an evolution of Multi Query Attention (MQA), a previous improvement to the original multi-head attention from the original Transformer paper [25]. MQA uses the same key and value heads for every query head, instead of having a separate one for each. While MQA greatly reduces the bandwidth required to load attention key and value vectors every step, it may lead to quality degradation and instability during training [32]. Grouped Query Attention reduces the amount of key and value heads and shares them to groups of query heads - creating a midway point between Multi-head attention and Multi Query attention. This enables it to be quick while remaining accurate.

Sliding Window Attention reduces the number of previous tokens one token can visit on each layer, saving memory and making it possible to fetch more data directly from the GPU cache instead of VRAM. At first glance, this seems like it would hurt performance, but as each layer can reach n tokens back recursively, the theoretical attention span remains large [5].

Some of these models come pre-tuned, often for conversational purposes, but they all start as raw highly advanced next word predictors after the highly resource-intensive pre-training step is completed. To adapt any of these LLMs to perform some other task, fine-tuning may be used.

2.6 Fine-tuning

If you wish to align a pre-trained LLM to perform a specific task like answering questions, translating, or summarizing text - fine-tuning is required. This is done using supervised data - data labeled by humans (or other systems) containing input, as well as expected outputs from the model. In essence, fine-tuning adapts the parameters to this data like any other LLM training method but is generally done with a smaller dataset compared to the ones used during pre-training.

As language models expanded in size, the number of parameters requiring fine-tuning escalated correspondingly. This surge in scale rendered it more or less impossible for individuals lacking the necessary resources to pre-train a model to even fine-tune one. Techniques for fine-tuning more efficiently with less hardware often had other performance implications, like introducing inference latency or sacrificing sequence length [33]. However, one technique that sought to mitigate this succeeded and has now become the standard for fine-tuning any large language model.

2.6.1 Low-Rank Adaptation (LoRA)

LoRA builds upon the discovery that weights during fine-tuning have a lower dimension "intrinsic rank", inspired by [34] and [35] which both showed that learned over-parameterized models exist within a low intrinsic dimension. The pre-trained weights may be represented as $W_0 \in \mathbb{R}^{d\times k}$ based on the transformer configuration, where a fine-tuned version of the model is $W = W_0 + \Delta W$. LoRA introduces $\Delta W = BA$ where $B \in \mathbb{R}^{d\times r}$, $A \in \mathbb{R}^{r\times k}$ and $r << min(d, k)$ which represents the rank - the amount of dimensions used to approximate the weight matrix. Instead of continually adapting the weights of the model like previous fine-tuning methods, LoRA completely freezes W_0 and trains A and B independently [33]. LoRA can be applied to each trainable weight matrix in a Transformer model, meaning that individual pairs of low-rank matrices are added to each FFN and MHA in every layer. Although in this original implementation, only a few of the weight matrices are replaced using LoRA.

When running inference the matrices A and B can be multiplied together to create ΔW, which is often referred to as an adapter. ΔW is thereafter added to the frozen weights as offsets to create the fine-tuned weights W which can be used just like before without any additional inference latency. Afterwards, the original weights can be restored by simply removing the adapter $W_0 =$

$W - \Delta W$. This brings additional benefits to developers by making it easy to exchange adapters on the fly during development - without having to fill up their storage with full model parameters.

The authors showcase that LoRA manages to reduce the VRAM usage by up to $\frac{2}{3}$ depending on r and the size of the checkpoint - the file that's the result of the fine-tune - by roughly 10000x. In addition to these significant gains, the authors also compared LoRA training with other training methods on multiple models. They found that LoRA-trained models score similarly while having significantly fewer trainable parameters - on multiple benchmarks [33].

2.6.2 Quantized Low-Rank Adaptation

Quantization - a way to reduce precision (the amount of possible values) using rounding and truncation is a popular way to reduce memory usage when handling very large models. By changing the data type of each value in the weights and biases from FP16 (floating point using 16 bits) precision down to FP8 or even FP4 (using 8 and 4 bits respectively), the amount of bits required for each value can be reduced by a significant amount. When a model is quantized, quantization constants are used to map the initial high precision format down to the specified quantized format. Likewise these constants are used to reverse the quantization when calculations are performed. This compression and decompression process is what enables low memory usage while not reducing quality significantly.

In addition to the reduced memory, the mathematical operations done using the quantized weights are less complex - which speeds up both training and inference. With previous techniques like SmoothQuant [36] quantization was only possible during inference.

With three key innovations, [37] introduced QLoRA, a technique to use a quantized model for LoRA fine-tuning.

1. **4-bit NormalFloat**, a special quantization type that captures information on normally distributed data better than 4-bit integers and floats.

2. **Double Quantization**, which is a process of quantizing the quantization constants, which otherwise are stored with high precision data types. This helps save about 0.37 bits per parameter in the model - reducing memory even further.

3. **Paged Optimizers**, which uses the NVIDIA unified memory feature to allocate paged memory on CPU RAM in case the GPU memory is

overfilled. This makes it easy to transfer it back to the GPU at the next step without running into errors

The authors showcase how 4-bit QLoRA training matches 16-bit full fine-tuning on a 33B model, and thus greatly revolutionized what kind of models can be trained using a consumer-grade GPU [37].

2.6.3 Reinforcement Learning from Human Feedback (RLHF)

A key problem in the machine learning field for many decades has been making a ML model do more of what you want, and less of what you don't want. This is typically accomplished by implementing reward functions, which essentially serve as a system of incentives, rewarding the model whenever it aligns with our objectives [38]. When the objective is clear, for example in games, where the score is defined as the function of some parameters, this is often easy. But as the tasks and applications get more complicated scoring becomes less trivial. Even though the user might have a good grasp what they expect from the model, defining these expectations as rules is often difficult or outright impossible.

For this reason, using RLHF has become increasingly popular. By training a model, and thereafter evaluating which responses are good or bad, the difficulties with defining complex criteria can be avoided. However using reinforcement learning to align the model would require a human to be ever present during the training (to evaluate whether a change is good or bad), and that would be incredibly time inefficient.

2.6.3.1 Reward Modeling (RM)

This can be circumvented by abstracting the objectives behind a **Reward Model** that has been trained to be able to generalize human feedback. The RM can thereafter be used to automatically rate any data on how well it aligns with the objective. The process of creating the RM involves prompting the supervised fine-tuned (SFT) model to generate pairs of answers $(y_1, y_2) \sim \pi_{\text{SFT}}(y|x)$ for given prompts. These pairs are then presented to human labelers who indicate their preference, creating a dataset assumed to be indicative of some latent reward model $r^*(y, x)$ [39]. The Bradley-Terry model is often employed to model these human preferences, formalized as:

$$p^*(y_1 \succ y_2|x) = \frac{\exp(r^*(x, y_1))}{\exp(r^*(x, y_1)) + \exp(r^*(x, y_2))}$$

Using the preference dataset, a parametrized reward model $r_\phi(x, y)$ is estimated through maximum likelihood, aiming to minimize the negative log-likelihood loss of the model's predictions against the observed preferences. This process rests on two assumptions which have been proven to hold up well is subsequent research, and makes up one of the key technologies behind ChatGPT and the previous InstructGPT models [40][10], presented in [38]:

> (1) We can learn user intentions to a sufficiently high accuracy.(2) For many tasks we want to solve, evaluation of outcomes is easier than producing the correct behavior.

The learned reward function $r_\phi(x, y)$ guides the optimization of the models policy π_θ (essentially the strategy the model uses to decide its response), towards generating outcomes that maximize the perceived reward. It is subject to a divergence constraint from a reference policy π_{ref}, to maintain generation diversity and prevent mode collapse [39]. This leads to the optimization problem:

$$\max_{\pi_\theta} \mathbb{E}_{x\sim D, y\sim\pi_\theta(y|x)}[r_\phi(x, y)] - \beta D_{KL}[\pi_\theta(y|x) \,||\, \pi_{\text{ref}}(y|x)]$$

Where β is a tunable parameter and the KL divergence term, denoted as $D_{KL}[\pi_\theta(y|x) \,||\, \pi_{\text{ref}}(y|x)]$, acts as a regularization mechanism that penalizes deviations from the reference policy π_{ref}. However, this objective is not differentiable and is often optimized using reinforcement learning, maximizing this standardized reward model [41]:

$$r(x, y) = r\phi(x, y) - \beta(\log \pi_0(y|x) - \log \pi_{ref}(y|x))$$

2.6.3.2 Proximal Policy Optimization (PPO)

PPO is a policy gradient method for reinforcement learning. Like previous fine-tuning techniques it relies on gradient descent, like the Adam optimizer, to align the model. However, reinforcement learning requires direct and iterative feedback from its environment to change its parameters, using a trial-and-error approach, balancing exploration and exploitation to reach an optimal policy.

In this case, it uses signals derived from the reward model to iteratively minimize a specially designed objective function, refining its responses to align better with the user's expectations.

The authors [42] introduce two PPO techniques, Clipped Surrogate Objective and Adaptive KL Penalty Coefficient, and showcase how the Clipped Surrogate version generally performs better.

$$\mathcal{L}^{CLIP}(\theta) = \hat{\mathbb{E}}_t \left[\min \left(r_t(\theta)\hat{A}_t, \text{clip}(r_t(\theta), 1-\epsilon, 1+\epsilon)\hat{A}_t \right) \right]$$

The clipped of probability ratio $\text{clip}(r_t(\theta), 1-\epsilon, 1+\epsilon)$ constrains the ratio $r_t(\theta)$ to the interval $[1-\epsilon, 1+\epsilon]$, where ϵ is a small value. This prevents the policy from updating too quickly, contributing to more stable learning.

The advantage function estimator $\hat{A}_t$ at time-step t quantifies the benefit of choosing a specific action in a given state over what is typically expected from the current policy. It directs the optimization process towards actions that offer higher returns than average, thus encouraging the policy to improve where it matters most. This helps foster a more targeted and efficient learning process.

The expected value operator $\hat{\mathbb{E}}_t$ averages over a batch of sampled trajectories, enabling the gradient descent updates to be based on estimates derived from actual interactions with the environment.

2.6.4 Direct Preference Optimization (DPO)

The two-step process of training a reward model and performing reinforcement learning with it is somewhat of a detour. Recent methods show how it is possible to directly use preference data to align the model weights.

The current state-of-the-art model adaptation technique based on feedback is Direct Preference Optimization [39]. This method is not in the reinforcement learning family, but rather direct alignment from preferences (DAP). The authors showcase how their method can achieve the same result as existing RLHF algorithms - which is maximizing a reward and with a Kullback–Leibler (KL) divergence constraint. Each DPO update aligns the model towards the objective by increasing the relative log probability of preferred to dispreferred responses.

The DPO optimization objective presented by the authors in [39] is achieved by transforming the reinforcement learning objective from 2.6.3.1 into a closed-form solution for the optimal policy by leveraging a specific reward model parameterization. This is achieved by reparameterizing the reward function in terms of the optimal policy and a reference policy, using the Bradley-Terry model for direct optimization based on human preferences, thus circumventing the expensive computation of the partition function and removing the need for an explicit reward model.

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

This final loss function L_{DPO} is used to guide the optimization of parameters. Here, σ denotes a logistic function, which enhances the effect when the reward estimate is incorrect. The first log function scaled by beta that contains y_w is used to increase the likelihood of y_w, and the next, containing y_l decreases the likelihood of y_l.

When fine-tuning a model using DPO, it is rather common to see some amount of over-fitting, meaning that the model becomes too aligned to the training data and wont be able to generalize its knowledge on other similar tasks [43]. Overfitting can usually be avoided by not training the model for as many epochs, and even though some subsequent improvements have been made to DPO, those wont be explored in this thesis as they do not support some optimization techniques at this time. Regular DPO is however easily combined with QLoRA to instead optimize some low-rank adapters while keeping similar results in terms of alignment and precision.

In the two previously sections, feedback on model output has been an integral part. Aligning a large language model can be done without adapting the model based on feedback after the fine-tuning stage, but generally, the SFT model might have some flaws - and additional training using feedback can fix them.

2.7 Feedback

The final training step using feedback exists because of the abstract nature of LLMs, which are often referred to as black boxes for a good reason. After the supervised fine-tuning stage is completed there is no way to tweak the model manually. There are only two ways to proceed: either change the dataset and re-train from scratch, or use other data to train it even further in hopes of making it better aligned.

While it may be easy to judge whether text generated by the SFT model is either good or bad, the reasons why are generally a lot more abstract. What has been learned during SFT might have achieved some unwanted and inexplicable effects, which cannot be erased by removing or changing entries in the data.

As mentioned in 2.6 this is the reason for simplifying it into preferences.

Many researchers and companies have relied on outsourcing the preference rating task to students, teachers or random people via online platforms like Prolific*. However, this may come with some unintended consequences if the goal alignment and preferences aren't properly communicated. relying on someone knowledgeable on the topic might be more effective and result in a better final preference dataset. However hiring an industry professional to perform this task is not only costly, but can also be time consuming. By the continued widespread usage of flawed metrics like BLEU and ROGUE, the industry's need for quick evaluation is evident and often seems to surpass the quality requirement.

In an age where conversational language models can perform most text-based tasks, and take on the role of any sort of professional, this should no longer be an issue. Lately, some scholars have used models like ChatGPT and Gemini Pro as a replacement of human evaluators to great success [44].

2.7.1 Reinforcement Learning from AI Feedback (RLAIF)

On of the first papers about using an AI to train another AI was [45] where the idea of a Constitutional Artificial Intelligence (CAI) was introduced. The authors describe how a model aligned to a certain rule set - or constitutions - can be used to align future models more efficiently to optimize both harmlessness and helpfulness.

However leaving supervision to another AI also has some downsides and dangers, one being further obscured decision making. This is a real issue as some crucial information about the system might go unnoticed. That said, the authors make a good point when comparing this to the reward modeling process in previous RLHF implementations which also obscure the decision-making.

One proven way to combat hidden reasoning is by using prompting techniques like Chain-of-Thought (CoT) [46]. The technique elicits the language model to explain its actions step by step, which offers several advantages. The first described by the authors is the decomposition of multi-step problems into smaller parts and increased accuracy when tackling those problems. An additional benefit is the ability to see each step of reasoning, and quickly identify where the model takes an incorrect step. In the context of a CAI [45] this is great, as it is easy to stop which type of behavior might slip through the cracks.

*https://www.prolific.com/

There are multiple ways to elicit CoT-style reasoning from a model. The most simple one is to ask the model to do a step-by-step style explanation. However, the results may vary using this technique, and the authors of [46] show how providing a previous example question and CoT answer in the prompt before asking the next question keeps the reasoning consistent.

A great upside of CoT-prompting is that it is applicable to all conversational models without any kind of modification necessary. The results of the technique may vary, often depending on the size of the model. The authors show how increasing the model size reduces all types of errors, which is consistent with the understanding that larger models have more detailed semantic understanding and logical reasoning. Lastly, there are no signs that CoT can result in a reduction of quality, so there is practically no reason to avoid it if the quality of the model output is of any importance.

2.7.2 Direct Alignment from AI Preferences

Very few articles have yet been written on the topic of using AI feedback with direct adaptation methods such as DPO. However, as the outcome of RLHF using PPO and DPO are similar, it is expected that a similar process using DPO would work as well.

In [44] DPO is used as a replacement in both RLHF and RLAIF to speed up the training process. The article compares which fine-tuning approaches a baseline Mistral 7B model benefits the most from. It is fine-tuned using completions and/or feedback from different ChatGPT models. The feedback is created by the chosen model by evaluating which is the best out of two alternative responses generated by a supervised fine-tuned model.

The authors conclude that SFT using GPT-4 completions greatly surpasses completions generated by GPT-3.5. A large improvement can be seen by using DPO on GPT-4 preference data on a model SFT-ed on GPT-3.5. However, just using SFT on GPT-4 completions still yields a better result. They also saw some small improvements on the GPT-4 SFT model after DPO using GPT-4 preference data, indicating that it still might be a worthwhile effort.

This article showcases how the original SFT dataset quality is of utmost importance, as starting with a better SFT baseline always seems to yield a higher quality model at the end of the process. Some have also tried using other models such as PaLM 2 Large and Gemini Pro as AI preference to great success [47].

2.8 Summary

The introduction of the Transformer revolutionized NLP tasks by replacing sequential processing with an attention mechanism, enabling the model to process input in parallel, significantly improving efficiency and the handling of long-range dependencies.

This advancement paved the way for the development of models like the Generative Pre-trained Transformer (GPT) and Bidirectional Encoder Representations from Transformers (BERT). These models leveraged the Transformer's architecture to set new benchmarks in language understanding and generation, demonstrating the effectiveness of pre-training on extensive corpora followed by task-specific fine-tuning. Additionally, the Mistral-7B model illustrated how transformers could be optimized for real-world applications, balancing performance and efficiency.

Innovations in fine-tuning and model adaptation methods, such as Low-Rank Adaptation (LoRA) and Direct Preference Optimization (DPO), represented further steps towards enhancing model performance with minimal computational resources. These techniques underscore the search for more efficient and effective ways to train NLP models, ensuring their adaptability and relevance to a wide range of applications.

The application of transformer-based models to generate multiple-choice questions (MCQs) exemplifies their versatility. By automating the generation of distractors, teachers can spend more time teaching, showcasing the potential of transformer based models beyond conventional NLP tasks like text summarization and translation.

Feedback mechanisms, including Reinforcement Learning from AI Feedback (RLAIF) and Direct Alignment from AI Preferences, have emerged as crucial components in the training and alignment of language models. These strategies highlight the importance of model alignment with human preferences and objectives, setting a direction for future research in the development of more responsive and aligned models.

In conclusion, this background section has explained the evolution of the NLP field, with a spotlight on the Transformer and how continuous innovation in model architecture, fine-tuning techniques, and the integration of feedback mechanisms has pushed the state-of-the-art even further and led to some practical applications of the technology.

Chapter 3

Method

This project began with a baseline BERT model presented in [3]. This model was pre-trained on a completely Swedish dataset, and later fine-tuned to generate distractors using an MCQ dataset, further discussed in 3.2. The different environments used for the AI and human evaluation, described in 3.5, were initially built to work with this model and were later extended to support the final Mistral 7B model.

As the baseline was built in 2021, and rapid developments have occurred in the field since then, it felt necessary to try a similar method using newer technologies. Additionally, recent methods such as DPO are not readily available for older models like BERT. Therefore, it was necessary to change the base model to attain all the project goals.

3.1 Model

As one core goal of the project was to train a model using consumer-grade hardware, the possible pre-trained model choices were rather limited. To be able to train, as well as run inference, a 7B parameter model requires up to 10GB of VRAM while using QLoRA as an optimization technique.

While there are many 7B models available on platforms like Huggingface*, many of them are already fine-tuned to perform specific tasks - or just generally unknown. So the choice stood between LLaMA-2 7B and Mistral 7B, and as Mistral has been shown to perform significantly better, sometimes even on the same level as 13B models, it was chosen for this project.

Mistral 7B has only been pre-trained, starting from this point is particularly great if the requirement for the model is to perform a more narrow text-based

*https://www.huggingface.com/

task, like generating distractors.

One thing worth noting here is that Mistral 7B is trained only to handle English [5]. However, it still outperforms most other common 7B models like LLaMA-2 7B and Eagle 7B in multilingual benchmarks [48]. Although it was not specifically designed for multilingual tasks, its performance suggests that it has been trained on Swedish texts to some extent. However, its proficiency in English is noticeably superior.

3.2 Data

The data used for supervised fine-tuning consists of the SweQUAD-MC dataset from [3] as well as some additional data collected by the same authors which were never used until now. The MCQs are reading comprehension tasks, with a context that accompanies each question, that contains the answer. In SweQUAD-MC all the distractors are extracted from the context, but this somewhat limits the possible questions to ask - this requirement was dropped for the additional data.

Both SweQUAD-MC and the additional data provided are in Swedish, provided in JSON format, with very few grammatical and spelling errors throughout. As previously mentioned, this is highly important as errors may persist and impact the final model. The dataset contains 2373 data points, created manually based on texts available on a multitude of Swedish government websites and reports.

3.2.1 Structure

Each data point in SweQUAD-MC consists of a list of choices, of which one is correct and the others are distractors. It also contains a context and the question in separate fields. Sometimes one context is reused for several different questions and answers, with a maximum of 4 different entries per context.

Listing 3.1: Structure of SweQUAD-MC

```
{
    "choices": [
        {
            "end": 1296,
            "extra": null,
            "start": 1290,
            "text": "100 kr",
            "type": "Correct answer"
        },
        {
            "end": 1184,
            "extra": {
                "comment": "3 kr"
            },
            "start": 1183,
            "text": "3",
            "type": "Distractor"
        },
        ...
    ],
    "context": "Företagets ansvar\nDen nya
       elsäkerhetslagen, som trädde i kraft den 1 juli
       2017, innebär att elinstallationsföretagen har
       fått ett helt nytt ansvar...",
    "question": "Hur mycket kostar Elsäkerhetsverkets
       handbok?"
}
```

The additional data is structured somewhat differently as it re-uses the same context for multiple groups of questions and answers. A few questions are simple yes-or-no questions, which can simply be ignored, and each context has a maximum of 6 questions.

Listing 3.2: Structure of Additional data

```
{
  "context": "Generaldirektörens kommentar\nUnder 2018
     har Brottsoffermyndigheten utvecklat arbetet för
     att sprida information..."
  "annotations": [
    [
      {
        "content": "Varför har antalet avgjorda ärenden
           rörande brottsskadeersättning minskat?",
        "marker": {
          "name": "Question",
          "order": 1
        },
        "group_order": 1
      },
      {
        "content": "Det har varit en personalomsättning
           på myndigheten.",
        "marker": {
          "name": "Correct answer",
          "order": 2
        },
        "group_order": 1
      },
      {
        "content": "Färre ansökningar har lämnats in.",
        "marker": {
          "name": "Distractor",
          "order": 3
        },
        "group_order": 1
      },
      ...
    ]
  ]
}
```

3.2.2 Processing

Both of the datasets showcased have a lot of extra tokens related to the JSON formatting. To optimize the fine-tuning process, the entries from the datasets were concatenated into a new dataset with a unified format without any unnecessary tokens.

Using a short Python script, the different datasets were loaded, and converted into small objects containing context, question, answer, and the distractors for each question. This resulted in 2438 data points. However, some (65) of the contexts were way too big (> 1024 tokens) and would have slowed down the training process significantly due to memory limitations. These entries were removed, resulting in a final dataset of size 2373.

To evaluate the model objectively, a test split was extracted from the dataset. It is important not to use any test data in the training split. A total of 772 contexts are used multiple times in different entries, so a small script was used to move over each context one by one, and all their corresponding entries into the test split until it contained 10% of all entries. This resulted in a training split with 2135 entries and a test split with 238 entries.

To give the model a way to separate the different parts of each entry, XML tags were used to surround the part, for example <context> ... </context>, for the context. This particular method was chosen over simple newline characters, as newlines commonly occur in the dataset, and could confuse the model. Some models like BERT rely on [SEP] tokens to differentiate between input types, but as this doesn't specify which type each input is, it was also avoided. The following tags were used: <context>, <question>, <true> and <false>.

These were thereafter exported into JSON format, using the Alpaca style dataset structure, which consists of four separate input fields per data point:

Listing 3.3: Alpaca Dataset Formatting

```
{
"instruction": "...", // the model task
"input": "...", // optional context or input
"output": "...", // generated response by model
"text": "..." // additional text data (if applicable)
}
```

For this dataset, only the **instruction** and **output** fields were used. The tokens **[INST]** and **[/INST]** are automatically added around the instruction during training and inference to provide a way to distinguish which part is the instruction.

The following tags were put in the instruction prompts in order: <context>, <question>, and <true>, and the remaining <false> tags containing the distractors were put in the prompt.

Listing 3.4: SFT Datapoint Example

```
{
"instruction":
   "<context>...</context>\n<question>...</question>\n
<true>...</true>\n",
"output":
   "<false>...</false>\n<false>...</false>\n<false>...</false>\n"
}
```

In the preference dataset, **output** is replaced by a pair of responses in a list, the first being a preferred and the second a dispreferred output. This dataset ended up with 877 data points.

Listing 3.5: DPO Datapoint Example

```
{
"instruction":
   "<context>...</context>\n<question>...</question>\n
<true>...</true>\n",
"output": [<false>...</false>\n, <false>...</false>\n]
}
```

3.3 Hardware

For training a Nvidia RTX 3080 GPU was used. The card has 10GB of VRAM, sufficient for training a 7B model using QLoRA during SFT and DPO.

3.4 Tooling

The multi-purpose fine-tuning library **LLaMA-Factory** [49] was used for training. The library offers flexible fine-tuning of 100+ LLMs, including Mistral 7B, with full precision, LoRA, or QLoRA, and other, even newer optimization methods at all different fine-tuning stages.

LLaMA-Factory makes fine-tuning simple by enabling training by specifying what and how to train using a Python script and some arguments. Additionally, automatic evaluation of the models using ROUGE and BLEU is

available, as well as API-like inference for integrating the trained model into other processes.

Following are the command line arguments, and explanations used to train a model, starting from the base Mistral 7B.

```
CUDA_VISIBLE_DEVICES=0 python src/train_bash.py \
--stage sft \
--do_train \
--model_name_or_path mistralai/Mistral-7B-v0.1 \
--dataset <training dataset> \
--template mistral \
--finetuning_type lora \
--lora_target q_proj,v_proj \
--output_dir <output directory for sft checkpoint> \
--overwrite_cache \
--per_device_train_batch_size 1 \
--gradient_accumulation_steps 4 \
--lr_scheduler_type cosine \
--logging_steps 10 \
--save_steps 400 \
--learning_rate 5e-5 \
--num_train_epochs 5.0 \
--plot_loss \
--fp16 \
--quantization_bit 4
```

Listing 3.6: CLI arguments for Supervised Fine-Tuning

template is used to specify the transformer architecture, by choosing mistral, the improvements from [5] are used.

lora_target specifies which learnable weight matrices to apply LoRA to. The same method as in the original paper [33] is used, only applying LoRA to the Query and Value matrices in each attention mechanism. The Key matrices, FFN weights, and final output matrix are left in their original state.

gradient_accumulation_steps specifies how many steps are accumulated until the LoRA matrices are changed using the Adam Optimizer. 4 was set as a default value here, and it helped speed up the training process while not going over the memory limit.

fp16 is used to denote that the input model is using 16-bit floating point values

in its matrices.

quantization_bit describes the number of bits the integer used to quantize each floating point value. A size of 4 bits was chosen, as this was the lowest value that made training possible on the available hardware.

This model was then evaluated to create a preference dataset. This was done by using the included API demo script to set up a simple API for inference that could be used by another system. This was done using the following arguments.

```
CUDA_VISIBLE_DEVICES=0 python src/api_demo.py \
--model_name_or_path mistralai/Mistral-7B-v0.1 \
--adapter_name_or_path <directory for sft checkpoint> \
--template mistral \
--finetuning_type lora \
--quantization_bit 4
```

Listing 3.7: CLI arguments for API Inference

The API was used to build the preference dataset. After that was done, it was time to fine-tune the model based on the feedback received. But before that, the adapters from the SFT were merged onto the original Mistral weights as there was no support for using multiple adapters at the same time. The merging was done using the following arguments.

```
python src/export_model.py \
--model_name_or_path "mistralai/Mistral-7B-v0".1  \
--adapter_name_or_path <directory for sft checkpoint> \
--template mistral \
--finetuning_type lora \
--export_dir <output directory for merged model> \
--export_size 2 \
--export_legacy_format False
```

Listing 3.8: CLI arguments for Merging

export_size describes how many GB each shard of the model can be. The model is split into shards to make it easier to load it into memory.

export_legacy_format set to false saves the model as a .safetensor file instead of a regular binary. The "safetensor" format is structured not to allow harmful code injection upon loading a model.

Lastly, DPO could be performed on the newly exported model using the generated preference dataset and these training arguments.

```
CUDA_VISIBLE_DEVICES=0 python3 src/train_bash.py \
--stage dpo \
--do_train \
--model_name_or_path <directory for merged model> \
--create_new_adapter \
--dataset <preference dataset> \
--template mistral \
--finetuning_type lora \
--lora_target q_proj,v_proj \
--output_dir <output directory for dpo checkpoint> \
--per_device_train_batch_size 1 \
--gradient_accumulation_steps 4 \
--lr_scheduler_type cosine \
--logging_steps 10 \
--save_steps 200 \
--learning_rate 1e-5 \
--num_train_epochs 1.0 \
--plot_loss \
--fp16 \
--quantization_bit 4 \
```

Listing 3.9: CLI arguments for Direct Preference Optimization

Because of the risk of over-fitting a model using DPO, checkpoints were saved every 200 steps. To evaluate the model a included prediction script was used to generate distractors for the data points in the test split. As no significant improvement was seen after one epoch, and quality started to degrade, the closest checkpoint at 200 steps was used.

```
CUDA_VISIBLE_DEVICES=0 python3 src/train_bash.py \
--stage sft \
--do_predict \
--model_name_or_path <directory for merged model> \
--adapter_name_or_path <directory for dpo checkpoint> \
--dataset <test dataset> \
--template mistral \
--finetuning_type lora \
--output_dir <prediction data and results directory>
--per_device_eval_batch_size 1
```

```
--max_samples 100
--predict_with_generate
--fp16
--quantization_bit 4
```

Listing 3.10: CLI arguments for Distractor Generation

max_samples was set to 100, meaning the 100 first data points from the test split are used to generate distractors.

3.5 Environment

To be able to evaluate the responses of the LM in its different fine-tuning stages, an environment for rating generated distractors was set up. As this environment was built before the new model had been trained, it initially relied on the distractor generator presented in [3]. However, this was easily swapped out to work with the new model.

The environment was built as a terminal program using Python and the Curses library, allowing for a better user experience by giving full access to rendering in the terminal and listening to keyboard inputs. This program was used to show the context and question of each data point, together with a list of distractors with the correct answer mixed in. The program allows users to check a box for each distractor they accept as a good alternative. This is done by pressing the corresponding index on their keyboard. After all distractors have been selected the user may move on to the next data point using the space bar. At this point, the distractors and their scores were saved to a database.

This environment was used during the final model evaluation by teachers, described further in 3.7. The environment was also used to create the preference dataset, used during the DPO fine-tuning process.

3.6 AI Integration

As one key aspect of the project involves replacing human feedback with AI feedback, the rating environment was extended to incorporate OpenAI:s GPT-4 model via their paid API. This integration replaces the front-end component in the terminal with a simple call to the LLM with some prompt, and upon the response some data extraction, before the data point and rating is stored in a database like before.

The reason for selecting GPT-4 in particular, is that upon the start of the project, it was the most capable LLM [50]. Choosing an outsourced solution like this was preferred as the given task (rating distractors) is complex and the quality of the responses is very important - as they will make up the training data for the final model. For this reason, the best possible model was selected - and the only way to access it is either via OpenAIs website or API [51]. And even if it was publicly available in the same vain as Mistral 7B - the hardware required to run it would be out of scope for most. It is worth mentioning that this part of the method is not free - as you're borrowing someone else's machine to run the model for you.

GPT-4 is fine-tuned to work well in conversations [51] and OpenAI offers no additional way to fine-tune it. So, to align it with the task of rating distractors, a prompt had to be crafted. While it's quite simple for a human to decide whether a distractor is good or bad, but the reason why can vary 2.1.3. Following is the final prompt, and some explanations for the different choices made when crafting it.

```
Your task is to rate distractors in multiple choice
   questions.
You will receive a context for the question, a
   question, an anwer and a list of distractors.
The goal is to give a score to each distractor between
   1-5, with 1 being the lowest and 5 being the highest.

The score is calculated as the sum of points. Point are
   gained or lost for a distractor by fullfilling one
   of the following criteria.

Criteria that each ADD ONE POINT towards the score:
* The distractor is fully grammatically correct as an
   answer to the question.
* The distractor makes sense in the context of the
   question and answer.
* The distractor is not the same as, a synonym to or
   closely related to the answer.
* The distractor is not an overarching concept or group
   that also contains the answer.
* The distractor is challenging and requires some
   knowledge of the context.

Criteria that each SUBTRACT ONE POINT from the score:
```

```
* The distractor is worded in a way that makes it stand
   out from the answer.
* The distractor is in the wrong language.

The input language will be Swedish.
You are allowed to give half points based on intuition
   to differentiate your scores if they are too close
   but you should strive to be as objective as possible.
You should avoid giving the same score to multiple
   distractors if possible, re score them with this in
   mind.

For each distractor write down how you are evaluating
   it step by step, and which criteria it succeeds or
   fails
The scores for each distractor should be put in a list
   in json format at the end, surrounded by $
example: $[4,1,3]$ for a list of 3 distractors

[QUESTION] ...
[ANSWER] ...
[DISTRACTOR 1] ...
[DISTRACTOR 2] ...
[DISTRACTOR 3] ...
...
```

Listing 3.11: Instruction Prompt for GPT-4 Distractor Rating

Prompting an LLM to perform a specific task is an iterative process. There is no straightforward way to predict how a changed prompt may impact how the LLM performs the task. As mentioned previously, some techniques like CoT may help, but can do the opposite in some instances. This prompt was created iteratively and evaluated using sample inputs at each change.

The first decision in the prompt writing process was to use a score ranging from 1 to 5 instead of a simple boolean to describe how good a distractor is. For each point added point, the distractor has to fulfill one criterion important to the quality of the distractor. The criteria were based on the rules previously mentioned in 2.1.3. Two criteria for subtracting points were also included as some errors have an extra large impact on the quality.

The prompt also gives the model some freedom by letting it reevaluate scores, this is helpful for scenarios where multiple distractors scored the same. The prompt further emphasizes trying to avoid giving the same score

to multiple distractors, and allows the model to add or subtract half a point to differentiate similar scores. This increased ”freedom” resulted in ratings closer to what a human would give.

An instruction to write down each step taken during evaluation is also given. While full Chain-of-Thought 2.7.1 style prompting was tested, always incorporating a previous example in the prompt, led to confusion and generally worse responses - as well as double the token usage. But, telling the model to work through the problem step by step seemed to positively affect the rating quality.

At the end of the instruction prompt, a preferred output format was given - which the model seemingly never failed to follow. The specified format allowed for easy extraction of the final scores, without having to parse the other parts of the model output which often vary in wording and structure.

To determine whether a distractor had been accepted or not, their score was compared to a limit determined individually for each set of rated distractors. In the initial tests, all distractors rated higher than three were accepted. After some testing, it became apparent that the scores between the different distractors in each prompt often correlated, meaning if one was rated high, the others also were, or the opposite. Hence a weighted limit was introduced, defined as:

$$limit = 2.5 + \left(\frac{\sum_{i=1}^{n} scores_i}{n} - 2.5 \right) \cdot 0.5$$

Both English and Swedish prompts were tested, and throughout this process having an English prompt always seemed to yield a more logically consistent response. The Swedish version often resulted in the final distractor ratings that did not follow the specified rules in the prompt, and it was also prone to repeat the score of the first distractor for the subsequent ones. Both of these errors were common, and thus severely hurt the accuracy when comparing it to a human. For this reason, English was chosen as the prompt language.

For the input, a decision was made to exclude the context in the AI preference rating process. Some questions have quite a large context, and should generally not be required to judge whether a distractor is reasonable. Additionally, each input token contributes to the cost of using GPT-4, so removing the context helped there as well.

3.7 Evaluation

The quality of generated distractors is difficult to determine using simple evaluation metrics such as BLEU or cosine similarity. As mentioned in

2.1, these metrics are often inaccurate as they cannot capture the complex requirements that make a distractor good.

For creating the preference dataset, GPT-4 and prompting techniques were used to automate the evaluation to give as similar ratings as possible to a human. However, human evaluation is still considered the gold standard, so the distractors were evaluated by teachers, the same audience the model might benefit in the future. This was done to gather an as accurate result as possible.

In this thesis, two steps of fine-tuning are performed on a default Mistral 7B model. After each of these steps, an adapter, serving as a training checkpoint, is created as output from the QLoRA training as described in 2.6.1. The first adapter is created after the supervised fine-tuning (SFT) and the second adapter continues from the first, and applies direct preference optimization (DPO) to improve it further. These two adapters are applied upon the original weights of the Mistral 7B model individually to create two distinct models, representing the two stages of the training process. These models are used to generate sample distractors from 100 data points in the test split of the dataset - which contains data neither model has been trained on. These two distractor datasets labeled **SFT** and **DPO**, and the human-made distractors from the test split labeled **Human** together made up the three sources used during the evaluation.

The reason for including human-made distractors from the test set is to gain insights into how picky the rater is and/or how good the actual data in the training dataset is. The percentage of how many of these distractors are accepted can be used as a guide for the maximum achievable result.

A version of the evaluation program used during the preference rating stage was used to let a teacher rate distractors randomly selected from any of the sources (with a maximum of two distractors per source). Each data point contained 4 distractors, resulting in 400 entries in total per evaluator. The results were thereafter saved to a database file which could be evaluated afterwards.

Following is an example of viewing a data point within the environment, for clarity the sources of each distractor have been added - but during evaluation, those were not visible to the user.

```
Select good distractors using the NUMBER keys. Press
   ENTER / SPACE to submit. Press ESC to exit. (1/100)

Question: Vad ska läkarintyget från det land du är
   bosatt i visa?
Answer: att du inte kan få behandlingen där

1. [ ] att visumet har slutat gälla (source: Human)
2. [ ] att du har deponerat pengar i Sverige för den
   planerade behandlingen (source: SFT)
3. [ ] när du ska få din behandling och vad den innebär
   (source: Human)
4. [ ] att du har deponerat pengar i Sverige för den
   planerade behandlingen (source: DPO)
```

Listing 3.12: Evaluation Data Example

Two teachers, both familiar with the topic of distractor generation, helped evaluate the final models for the results. Instructions on how the evaluation program works, as well as some general guidelines on what makes a great distractor were supplied in addition to the program.

```
Instructions

Your task is to select good-quality distractors
   (incorrect alternatives) for multiple-choice
   questions. You will be shown a question, the correct
   answer, and a list of four distractors.

Distractors are marked using their corresponding number
   key. This key, along with a checkbox will be
   displayed next to each distractor. A distractor is
   marked as correct if the checkbox is checked. You
   can select as many distractors as you like, or none
   at all. When you are ready to submit your rating and
   proceed to next question, press the enter or space
   key. If you need to exit the program, press ESC, and
   your progress will be saved.

To avoid bias - do not look at evaluation.json before
   rating.

Rating instructions

- Each distractor should be rated from the perspective
   that you don't know the answer to the question.
- Focus on the distractor's grammar, relevance, and how
   well it fits the question.
- If the distractor is, or is similar to the correct
   answer, it should be rated as incorrect.
- If the same distractor is shown multiple times for
   one question, rate them all the same way, as if they
   were unrelated.

...
```

Listing 3.13: Evaluation Instruction

Because each distractor was selected randomly, the chance of seeing one distractor more than once is rather high. This is mainly because the SFT

and DPO checkpoints have been trained on the same data, and also that the distractor often is a part of the context. Thus, a specific instruction was given to the evaluators that they should always rate duplicate distractors the same. The evaluators were also given the instruction to view all distractors from the perspective that they do not know the answer. The reason behind this is that it is easy to think of a distractor as trivially incorrect if you are familiar with the topic.

Chapter 4

Results

This section presents the results, gathered from the two evaluators. First basic metrics and findings are presented regarding the distractors in 4.1. Thereafter the results from the two evaluators are compared 4.2. Lastly, some insights into the AI Preference data is given 4.3.

The following tables and graphs feature the results split into sources. These fields denote which model or dataset the rated distractors originate from. The final model (the DPO checkpoint) is shortened to just **DPO**, similarily the SFT checkpoint is **SFT**. The final source **Human** contains human-made distractors included as examples in the test split.

4.1 Model Performance

The randomized selection of distractors resulted in some differences in the number of distractors used from each source, visible in 4.1, but the percentages of accepted distractors from each source paint a clear picture of their respective performance. Scoring the highest were the human-made distractors included in the test split of the dataset. The two Mistral 7B fine-tunes (DPO and SFT) however scored similarly $\approx 45\%$ 4.1. This is compared to the baseline BERT model, which was trained on a subset of the data used in this thesis and scored 49% [3].

On average only 61.6% of distractors from the Human were accepted by the teachers, meaning 38.8% of those distractors were not up to the teachers' standards.

With some additional filtering, it's possible to observe how questions with rejected human-made distractors (from Human) affect the scores of the other sources. The split, shown in 4.2, is achieved by following these steps:

Table 4.1: Teacher evaluation of Human-made and generated distractors from the DPO & SFT checkpoints

Source	Accepted	Total	Percentage
DPO	119	248	48.0%
SFT	121	284	42.6%
Human	165	268	61.6%

1. The data points (each rated distractor) are sorted by their respective questions.

2. For any question without human-made distractors, the accompanying distractors are filtered out.

3. For any remaining question where there are one or more human-made distractors, and they are not unanimously accepted by both teachers - the accompanying distractors are filtered out.

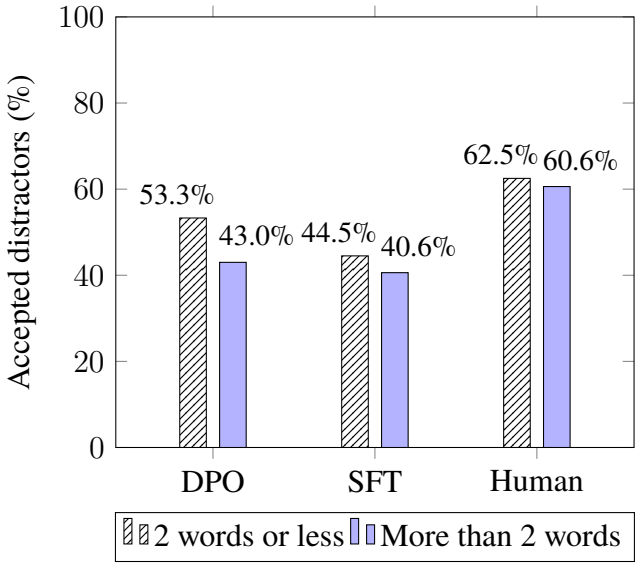
The results in 4.2 show how applying this split lets the human-made distractors achieve 100% acceptance. More interestingly, the results also show how this makes the distractors from the DPO checkpoint score even higher than the average rating of the Human data (when not applying this split) - showing that these questions in particular have severe impact on the model performance.

Table 4.2: Filtered teacher evaluation of Human-made and generated distractors from the DPO & SFT checkpoints, without questions where Human-made distractors were rejected

Source	Accepted	Total	Percentage
DPO	53	82	64.6 %
SFT	48	90	53.3 %
Human	91	91	100.0 %

By splitting distractors from each source into two categories based on their length in words, shown in 4.1, reveales that distractors from all sources perform better when generating shorter distractors (<=2 words). However, the SFT and DPO distractors benefit from this split to a higher degree.

Figure 4.1: Teacher acceptance rate of Human-made and generated distractors from the DPO & SFT checkpoints, split based on word length



4.2 Evaluator Comparison

The two teachers who helped evaluate the results of this thesis differed somewhat in their responses. Following are the evaluation scores from Teacher 1 4.3 and Teacher 2 4.4.

Table 4.3: Teacher 1 evaluation of Human-made and generated distractors from the DPO & SFT checkpoints

Source	Accepted	Total	Percentage
DPO	62	124	50.0 %
SFT	70	142	49.3 %
Human	95	134	70.9 %

Both teachers rated the distractors from Human the highest, DPO next, and thereafter SFT. But there are still some glaring differences between the them. The first, and maybe the most noticeable difference is that Teacher 2 4.4 gave more strict ratings on all sources compared to Teacher 1 4.3. The source with the biggest difference between the two teachers was Human with 18.7 percentage points.

Table 4.4: Teacher 2 evaluation of Human-made and generated distractors from the DPO & SFT checkpoints

Source	Accepted	Total	Percentage
DPO	57	124	46.0 %
SFT	51	142	35.9 %
Human	70	134	52.2 %

The second noteworthy difference is the relative disparity between the SFT and DPO models in each of the teachers' ratings. Whereas Teacher 1 rated both DPO and SFT very similarly, Teacher 2 scored the DPO model more than 10 percentage points higher than SFT.

In 4.1 an example of a MCQ from the evaluation is shown, where the two teachers agree on some distractors, and disagree on others - showcasing some underlying difference their in evaluation criteria.

```
Question: Vad har elektrikern till sin hjälp?
Answer: mätinstrument

1. [ ] automatiska anläggningar (source: DPO)
2. [T2] ritningar (source: DPO)
3. [ ] anläggningar (source: SFT)
4. [T1, T2] motorer (source: Human)
```

Listing 4.1: Teacher 1 (T1) and Teacher 2 (T2) ratings of distractors from different sources on an evaluation MCQ

4.3 AI Preference Insights

The SFT model was used to generate distractors that were turned into preference ratings by GPT-4. When comparing the GPT-4 ratings with the teachers' ratings on the SFT distractors, as shown in 4.5, there is a large discrepancy between the percentages of accepted distractors. One of the reasons for this is that the distractors generated for the preference dataset used the training split for training, similar to the SFT step. However, it might not be the whole truth. By randomly sampling accepted distractors for human review, it became apparent that not every distractor accepted by GPT-4 should have been. This was even though GPT-4 was given carefully considered instructions 3.11, meant to replicate the way a human would judge distractors.

Table 4.5: Comparison between GPT-4 and Teacher evaluation on distractors generated by the SFT checkpoint

Rater	Accepted	Total	Percentage
GPT-4	4 369	5 345	81.7 %
SFT	121	284	42.6 %

Chapter 5

Discussion

5.1 Performance

The final Mistral 7B model introduced in this thesis performed slightly worse than the BERT-based model, despite using more Swedish fine-tuning data, being pre-trained on more text, having a lot more trainable parameters and bringing improvements to the transformer architecture. Many factors may play into this - following are some probable, and less probable reasons for this result.

5.1.1 Dataset

The results showed that about 38% of the distractors in the training data were not accepted by the teachers. This could be attributed to either the teacher having too high quality standards or the distractors simply not being good enough. Either way, this discrepancy surely had an impact on the final model and it is reasonable to assume that by training on data that's only accepted about 62% of the time - the final model will not surpass that.

By filtering out questions where either teacher did not accept all the distractors from the test split of the dataset, some significant improvements can be seen. The percentage of accepted distractors originating from the DPO and SFT model jumps up about 13 and 10 percentage points, respectively. This seems to further hint towards that the quality of the generated distractors correlates with the quality of the distractors shown during training. How difficult the question some distractors belong to is, might also have played a role here. If a human has a hard time to come up with good distractors for a question - the model will most likely have the same problem.

Some additional data was added to the original dataset from [3]. It is worth mentioning that this data featured significantly more advanced Swedish - as it originates from the annual reports of Swedish government agencies. This type of data was selected with the hope of teaching the model more advanced words and phrases, to allow it to handle more types of contexts and questions during inference. This dataset is also a bit different compared to the first as the instructions followed by the creator were different. The largest change in the instructions was that the distractors were no longer required to be present in the context of the question.

It is worth considering the potential negative effects of mixing data created using different instructions and of different linguistic sophistication in the same training dataset. For instance, if the model is trained to extract relevant distractors from the context on some data points, adjusting the weights in some way, data points that do not follow the same rule might adjust them differently. This could affect not only the efficiency of the training, but also the final results.

Lastly, one of the most likely reasons for the minimal change in performance is simply the small scale of the dataset. The limit of what's achievable with a dataset with "only" a few thousand entries might have been reached. To gain better results, it might be the case that a lot more (and preferably better and more consistent) data is required. Creating a perfect distractor generator by training on just 2373 data points is most likely a too-ambitious goal. But as this model is to be used as a tool aimed to help teachers come up and sort through distractors - this model still serves a purpose.

5.1.2 Preferences

The DPO model scored somewhat higher than the SFT model, hinting at some improvement because of the preference data. However, this was curiously only due to the rating of Teacher 2, as Teacher 1 rated the two models very similarly.

As showcased in the results in chapter 4, the GPT-4 based AI distractor rating system accepted over 80% of the generated distractors. Some of those probably would not have been accepted by the teachers who evaluated the final results. If that is the case the DPO step would sometimes have aligned the model towards generating bad distractors - counteracting the positive effect achieved by the correctly judged distractors. Because of that, it is likely that human ratings would have created a larger positive impact on the model - that perhaps also could satisfy Teacher 1.

5.1.3 Language

The choice of making a Swedish distractor generator from Mistral 7B might have also contributed to the similar score compared to the baseline BERT model. The BERT model was pre-trained on a completely Swedish dataset - compared to the Mistral 7B model which is mainly an English model.

It might simply be the case that Mistral 7B has not seen enough Swedish text during pre-training to be able to come up with good distractors for every context. A sub-par understanding of the Swedish language could be the reason why both the SFT and DPO models perform better when generating short distractors. Longer, more complex sequences might make this fault become more obvious to a human evaluator.

5.1.4 Size

A 7B parameter model was chosen due to the self-imposed hardware constraints in this project. Today, 7B is on the lower end of LLM sizes, with the top performing models having hundreds of billions of parameters. It is reasonable to assume that a larger model would perform even better, as they've been shown to do in most cases. However, as the baseline BERT model with 110M parameters [31] (1.6% of 7B) performed better than the new model - it is likely the previously mentioned reasons had a larger impact on the results.

Additionally, 4-bit quantization with QLoRA was used to during both training and inference to make it possible to run the model on the available hardware 3.3. But as mentioned in 2.6.2 this technique doesn't really impact performance, suggesting once again that the above reasons had larger impact, and that training using consumer grade hardware is a viable alternative.

5.1.5 Prompt Separation using XML

As shown in 3.2 the different parts of the prompt used to train and run inference on Mistral 7B were wrapped in the XML tags. The first versions of the SFT model were trained without using anything to wrap each section. Because the dataset contained newlines that made sense to keep, just beginning each new part with a newline and *CONTEXT:* or similar didn't work. When testing models that were trained this way, it was common for input fields to repeat themselves among the distractors in the output - making them inconsistent and unusable. The XML tags solved this in addition to making it simple to parse out the generated distractors from the output of a model.

An additional fun observation that revealed itself as a side-effect of the tags was that the model also implicitly was fine-tuned to be able to generate the other tags as well. For example, if the user gives an input context and nothing else - the model automatically generates a question, answer, and distractors. This means that the final model in this project is also a full MCQ generator. However, as the DPO stage only featured distractor preferences, this part of the training process did not help in regards of generating other tags.

5.1.6 AI Preference Model & Language

Both of the most popular OpenAI models GPT-4 & ChatGPT (GPT-3.5 Turbo) were tested as distractor evaluators. When experimenting with the two models, the predecessor ChatGPT, showcased inconsistencies that could not be ignored. The most common of these was to give the same evaluation to all of the supplied distractors. It was also common for it to give wildly different scores on repeated tries using the same input. While neither of these inconsistencies disappeared completely by switching to the newer GPT-4 model, they became less common.

Another curious discovery was also made while experimenting with different prompts for GPT-4. A couple of tests were done using a Swedish translation of the prompt, meaning both the instruction and the input (data point with MCQ data) were in the same language. Surprisingly this made the results a lot less consistent - similar to the drastic difference between ChatGPT and GPT-4.

5.2 Societal Impact

While Artificial Intelligence has found practical applications in society for decades, the recent boom of research following the introduction of the Transformer have some worried how it will transform society. Shortly after its release, ChatGPT became the quickest service ever to reach both 1 and 100 million users[52].

The topic of this thesis is a rather tame use-case for the technology, with the aim to simply aid teachers in their workflow. This is believed to have a positive impact on the amount of preparatory work a teacher has to do in addition to teaching.

However, there is no avoiding that the general technology eventually might be used to replace jobs. While concerning, this is not the first time industries

and jobs have been displaced because of technological evolution, and it's not likely to be the last either.

World governments have been quick to draft up regulations for the use cases of AI. The European Union's AI Act[53] was passed into law earlier this year, making it possible to hold bad actors using AI for malicious purposes accountable. However, legislation may also result in stifling innovation by wrongly prohibiting some ethical use cases.

5.3 Ethics and Sustainability

Distractor generators like the one presented in this thesis are tools to help teachers build MCQs faster. The teacher still makes up an integral part of the creation process - mainly selecting the context, question, and answer, and vetting if the generated distractors are good enough. In a scenario where distractor generators (and maybe more likely, full MCQ generators) would be used to replace teachers, the quality would likely degrade. This could impact a school's or other institution's ability to test knowledge, resulting in knowledge gaps. An example could be the theoretical part of a driver examination, which often consists of an MCQ. If an AI is tasked to make it, and it is not vetted by anyone, it might end up making the distractors too easy. This could result in some getting a license without the proper knowledge, and could have significant consequences in society.

Very large language models like the current top performer GPT-4 require enormous amounts of energy during training [54]. But it is not just training that is expensive, inference also drains energy - especially when serving millions of users concurrently. For some narrower tasks that do not require the same kind of logical reasoning presented in ChatGPT and similar assistants, a smaller model might be able to compete, and do so more efficiently.

This is the reason for the decision to focus on smaller, widely available models and data for this project. One of the key goals was to showcase the process of fine-tuning a smaller 7B parameter model to a specific task. While some hardware and energy are required for the training and inference, it is magnitudes less than GPT-4, for instance. Additionally, the process of training these models has gotten rather simple, and the required hardware is generally quite accessible.

Chapter 6

Conclusions and Future work

6.1 Conclusions

The final Mistral 7B distractor generator is competitive with the baseline BERT model introduced in [3], but changing to a newer base model did not have the significant improvements that were expected.

More interestingly, the model fine-tuned using direct preference optimization (DPO) on AI preferences performed a little better compared to the one without it - at least according to one of the two evaluators. When filtering out questions where the human written distractors from the dataset weren't accepted, the model trained with DPO scored 64.6%, in comparison to the only supervised fine-tuned model which scored 53.3%. While not a huge difference, DPO can enhance the accuracy of a Mistral 7B model, even when using a small-scale dataset in a less common language.

The improvements seen on the DPO model trained using AI preferences showcase the possibility of replacing human feedback. When comparing preferences generated by GPT-4 with the human evaluators, there is a stark difference in the acceptance rate of distractors - resulting in misaligned AI ratings. This might have contributed to a less than optimal effect of DPO training, compared to preference data created by humans. Interestingly, while state-of-the-art very large language models like ChatGPT and GPT-4 are capable at some tasks, it is not easy to convey complex rules and get great reproducible outcomes, especially in a less common language like Swedish. That said, the improvement from the supervised fine-tuned (SFT) model still hints towards that replacing human preferences with AI preferences still yields improvements, but needs additional work to reach the same quality.

Lastly, this project has shown that it is perfectly possible to use state-of-

the-art language modeling and consumer-grade hardware to create a distractor generator that can aid teachers in making Swedish multiple-choice questions. This is possible thanks to techniques like LoRA, quantization, and other efficiency improvements that heavily reduce the required hardware while maintaining similar quality. For smaller and more narrowly focused models there are few downsides with this approach, and as sometimes outsourcing inference and training via another company isn't a possibility, training a model locally serves as a great alternative.

6.2 Future work

As mentioned in the discussion, there may be several contributing factors to the performance of the final model. In this section, potential improvements are presented.

6.2.1 Dataset

Both of the two evaluators gave the human-made distractors originating from the test split of the dataset a rather low score. This was surprising, as the distractors from the training dataset were expected to be of high quality, and score nearly 100%. This is in contrast to their actual scores, 70.9%, and 52.2%, for Teacher 1 and Teacher 2 respectively.

As the Test split is a random selection of questions and contexts from the complete training data, this quality issue is likely to spread throughout the whole dataset - impacting the final models of this thesis, as well as the baseline BERT model [3] they are compared against. A way to combat this would be to involve the evaluators in the dataset-making process, and let them remove or optionally rewrite distractors they perceived as not good enough.

6.2.2 Preferences

The AI preference data was very generous towards the generated distractors. Future work could further explore the differences between human and AI evaluation and new ways to prompt the model to get it further aligned with human evaluators.

6.2.3 Language

On the language front, it would be interesting to experiment with another model that has been pre-trained on more Swedish data, to see whether that was one of the bottlenecks experienced in this project. Additionally, if another easily accessible LLM on the same level as GPT-4 releases, but it's better at Swedish, it would be interesting to see whether its preference ratings would be closer to the human evaluators.

References

[1] V. D. Lai, N. T. Ngo, A. P. B. Veyseh, H. Man, F. Dernoncourt, T. Bui, and T. H. Nguyen, "Chatgpt beyond english: Towards a comprehensive evaluation of large language models in multilingual learning," 2023. [Pages 1 and 6.]

[2] Y. Susanti, T. Tokunaga, H. Nishikawa *et al.*, "Evaluation of automatically generated english vocabulary questions," *RPTEL*, vol. 12, p. 11, 2017. doi: 10.1186/s41039-017-0051-y. [Online]. Available: https://doi.org/10.1186/s41039-017-0051-y [Pages 2 and 6.]

[3] D. Kalpakchi and J. Boye, "Bert-based distractor generation for swedish reading comprehension questions using a small-scale dataset," 2021. [Pages 2, 3, 7, 8, 27, 28, 36, 45, 52, 57, and 58.]

[4] A. Radford, K. Narasimhan, T. Salimans, and I. Sutskever, "Improving language understanding by generative pre-training," 2018. [Online]. Available: https://s3-us-west-2.amazonaws.com/openai-assets/research-covers/language-unsupervised/language_understanding_paper.pdf [Pages 2 and 16.]

[5] A. Q. Jiang, A. Sablayrolles, A. Mensch, C. Bamford, D. S. Chaplot, D. de las Casas, F. Bressand, G. Lengyel, G. Lample, L. Saulnier, L. R. Lavaud, M.-A. Lachaux, P. Stock, T. L. Scao, T. Lavril, T. Wang, T. Lacroix, and W. E. Sayed, "Mistral 7b," 2023. [Pages 2, 17, 28, and 33.]

[6] H. Roediger, A. Putnam, and M. Sumeracki, *Ten Benefits of Testing and Their Applications to Educational Practice*. Elsevier Academic Press, 01 2011, vol. 55, pp. 1–36. ISBN 9780123876911 [Page 5.]

[7] N. Mulla and P. Gharpure, "Automatic question generation: a review of methodologies, datasets, evaluation metrics, and applications," *Prog*

Artif Intell, vol. 12, no. 1, pp. 1–32, 2023. doi: 10.1007/s13748-023-00295-9 Epub 2023 Jan 30. PMCID: PMC9886210. [Page 6.]

[8] K. Vachev, M. Hardalov, G. Karadzhov, G. Georgiev, I. Koychev, and P. Nakov, "Leaf: Multiple-choice question generation," 2022. [Pages 6 and 7.]

[9] E. Gabajiwala, P. Mehta, R. Singh, and R. Koshy, "Quiz maker: Automatic quiz generation from text using nlp," in *Futuristic Trends in Networks and Computing Technologies*, P. K. Singh, S. T. Wierzchoń, J. K. Chhabra, and S. Tanwar, Eds. Singapore: Springer Nature Singapore, 2022. ISBN 978-981-19-5037-7 pp. 523–533. [Page 6.]

[10] OpenAI. (2022) Introducing chatgpt. Accessed: 2024-03-08. [Online]. Available: https://openai.com/blog/chatgpt [Pages 6 and 21.]

[11] B. Cheung, G. Lau, G. Wong, E. Lee, D. Kulkarni, C. Seow, R. Wong, and M. Co, "Chatgpt versus human in generating medical graduate exam multiple choice questions-a multinational prospective study (hong kong s.a.r., singapore, ireland, and the united kingdom)," *PLoS One*, vol. 18, no. 8, p. e0290691, Aug 2023. doi: 10.1371/journal.pone.0290691 [Page 6.]

[12] J. Welbl, N. F. Liu, and M. Gardner, "Crowdsourcing multiple choice science questions," 2017. [Page 6.]

[13] S. K. Bitew, J. Deleu, C. Develder, and T. Demeester, "Distractor generation for multiple-choice questions with predictive prompting and large language models," 2023. [Page 6.]

[14] G. Lai, Q. Xie, H. Liu, Y. Yang, and E. Hovy, "Race: Large-scale reading comprehension dataset from examinations," *arXiv preprint arXiv:1704.04683*, 2017. [Page 7.]

[15] K. Papineni, S. Roukos, T. Ward, and W.-J. Zhu, "Bleu: a method for automatic evaluation of machine translation," in *Proceedings of the 40th Annual Meeting of the Association for Computational Linguistics*, P. Isabelle, E. Charniak, and D. Lin, Eds. Philadelphia, Pennsylvania, USA: Association for Computational Linguistics, Jul. 2002. doi: 10.3115/1073083.1073135 pp. 311–318. [Online]. Available: https://aclanthology.org/P02-1040 [Page 7.]

[16] C.-Y. Lin, "ROUGE: A package for automatic evaluation of summaries," in *Text Summarization Branches Out*. Barcelona, Spain: Association for Computational Linguistics, Jul. 2004, pp. 74–81. [Online]. Available: https://aclanthology.org/W04-1013 [Page 7.]

[17] D. De-Fitero-Dominguez, E. Garcia-Lopez, A. Garcia-Cabot, J.-A. Del-Hoyo-Gabaldon, and A. Moreno-Cediel, "Distractor generation through text-to-text transformer models," *IEEE Access*, vol. 12, pp. 25 580–25 589, 2024. doi: 10.1109/ACCESS.2024.3361673 [Page 7.]

[18] M. Zyrianova, D. Kalpakchi, and J. Boye, "Embrace: Evaluation and modifications for boosting race," 2023. [Page 7.]

[19] C. Callison-Burch, M. Osborne, and P. Koehn, "Re-evaluating the role of Bleu in machine translation research," in *11th Conference of the European Chapter of the Association for Computational Linguistics*, D. McCarthy and S. Wintner, Eds. Trento, Italy: Association for Computational Linguistics, Apr. 2006, pp. 249–256. [Online]. Available: https://aclanthology.org/E06-1032 [Page 7.]

[20] A. Smith, C. Hardmeier, and J. Tiedemann, "Climbing mont BLEU: The strange world of reachable high-BLEU translations," in *Proceedings of the 19th Annual Conference of the European Association for Machine Translation*, 2016, pp. 269–281. [Online]. Available: https://aclanthology.org/W16-3414 [Page 7.]

[21] M. Zhang, C. Li, M. Wan, X. Zhang, and Q. Zhao, "Rouge-sem: Better evaluation of summarization using rouge combined with semantics," *Expert Systems with Applications*, vol. 237, p. 121364, 2024. doi: https://doi.org/10.1016/j.eswa.2023.121364. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0957417423018663 [Page 8.]

[22] S. Hochreiter and J. Schmidhuber, "Long Short-Term Memory," *Neural Computation*, vol. 9, no. 8, pp. 1735–1780, 11 1997. doi: 10.1162/neco.1997.9.8.1735. [Online]. Available: https://doi.org/10.1162/neco.1997.9.8.1735 [Page 9.]

[23] K. Cho, B. van Merrienboer, Çaglar Gülçehre, D. Bahdanau, F. Bougares, H. Schwenk, and Y. Bengio, "Learning phrase representations using rnn encoder–decoder for statistical machine translation," in *Conference on Empirical Methods in Natural Language*

Processing, 2014. [Online]. Available: https://api.semanticscholar.org/CorpusID:5590763 [Page 9.]

[24] O. Kuchaiev and B. Ginsburg, "Factorization tricks for lstm networks," *arXiv preprint arXiv:1703.10722*, 2017. [Page 9.]

[25] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, Ł. Kaiser, and I. Polosukhin, "Attention is all you need," *Advances in neural information processing systems*, vol. 30, 2017. [Pages 9, 10, 11, 12, 13, 14, and 17.]

[26] J. Alammar, "The illustrated transformer," http://jalammar.github.io/illustrated-transformer/, 2018, accessed: 2024-03-29. [Page 9.]

[27] D. Jurafsky and J. H. Martin, *Speech and Language Processing*, 3rd ed. Pearson, 2023. ISBN 9780131873216 [Pages 13, 14, and 15.]

[28] D. P. Kingma and J. Ba, "Adam: A method for stochastic optimization," 2017. [Page 15.]

[29] A. M. Dai and Q. V. Le, "Semi-supervised sequence learning," *Advances in neural information processing systems*, vol. 28, 2015. [Page 16.]

[30] P. J. Liu, M. Saleh, E. Pot, B. Goodrich, R. Sepassi, L. Kaiser, and N. Shazeer, "Generating wikipedia by summarizing long sequences," *arXiv preprint arXiv:1801.10198*, 2018. [Page 16.]

[31] J. Devlin, M.-W. Chang, K. Lee, and K. Toutanova, "BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding," *arXiv e-prints*, p. arXiv:1810.04805, Oct. 2018. doi: 10.48550/arXiv.1810.04805 [Pages 16 and 53.]

[32] J. Ainslie, J. Lee-Thorp, M. de Jong, Y. Zemlyanskiy, F. Lebrón, and S. Sanghai, "Gqa: Training generalized multi-query transformer models from multi-head checkpoints," 2023. [Page 17.]

[33] E. J. Hu, Y. Shen, P. Wallis, Z. Allen-Zhu, Y. Li, S. Wang, L. Wang, and W. Chen, "LoRA: Low-Rank Adaptation of Large Language Models," *arXiv e-prints*, p. arXiv:2106.09685, Jun. 2021. doi: 10.48550/arXiv.2106.09685 [Pages 18, 19, and 33.]

[34] C. Li, H. Farkhoor, R. Liu, and J. Yosinski, "Measuring the Intrinsic Dimension of Objective Landscapes," *arXiv e-prints*, p. arXiv:1804.08838, Apr. 2018. doi: 10.48550/arXiv.1804.08838 [Page 18.]

[35] A. Aghajanyan, L. Zettlemoyer, and S. Gupta, "Intrinsic Dimensionality Explains the Effectiveness of Language Model Fine-Tuning," *arXiv e-prints*, p. arXiv:2012.13255, Dec. 2020. doi: 10.48550/arXiv.2012.13255 [Page 18.]

[36] G. Xiao, J. Lin, M. Seznec, H. Wu, J. Demouth, and S. Han, "Smoothquant: Accurate and efficient post-training quantization for large language models," 2023. [Page 19.]

[37] T. Dettmers, A. Pagnoni, A. Holtzman, and L. Zettlemoyer, "Qlora: Efficient finetuning of quantized llms," 2023. [Pages 19 and 20.]

[38] J. Leike, D. Krueger, T. Everitt, M. Martic, V. Maini, and S. Legg, "Scalable agent alignment via reward modeling: a research direction," 2018. [Pages 20 and 21.]

[39] R. Rafailov, A. Sharma, E. Mitchell, S. Ermon, C. D. Manning, and C. Finn, "Direct preference optimization: Your language model is secretly a reward model," 2023. [Pages 20, 21, and 22.]

[40] L. Ouyang, J. Wu, X. Jiang, D. Almeida, C. L. Wainwright, P. Mishkin, C. Zhang, S. Agarwal, K. Slama, A. Ray, J. Schulman, J. Hilton, F. Kelton, L. Miller, M. Simens, A. Askell, P. Welinder, P. Christiano, J. Leike, and R. Lowe, "Training language models to follow instructions with human feedback," 2022. [Page 21.]

[41] D. M. Ziegler, N. Stiennon, J. Wu, T. B. Brown, A. Radford, D. Amodei, P. Christiano, and G. Irving, "Fine-tuning language models from human preferences," 2020. [Page 21.]

[42] J. Schulman, F. Wolski, P. Dhariwal, A. Radford, and O. Klimov, "Proximal policy optimization algorithms," 2017. [Page 21.]

[43] M. G. Azar, M. Rowland, B. Piot, D. Guo, D. Calandriello, M. Valko, and R. Munos, "A general theoretical paradigm to understand learning from human preferences," 2023. [Page 23.]

[44] A. Sharma, S. Keh, E. Mitchell, C. Finn, K. Arora, and T. Kollar, "A critical evaluation of ai feedback for aligning large language models," 2024. [Pages 24 and 25.]

[45] Y. Bai, S. Kadavath, S. Kundu, A. Askell, J. Kernion, A. Jones, A. Chen, A. Goldie, A. Mirhoseini, C. McKinnon, C. Chen, C. Olsson, C. Olah,

D. Hernandez, D. Drain, D. Ganguli, D. Li, E. Tran-Johnson, E. Perez, J. Kerr, J. Mueller, J. Ladish, J. Landau, K. Ndousse, K. Lukosuite, L. Lovitt, M. Sellitto, N. Elhage, N. Schiefer, N. Mercado, N. DasSarma, R. Lasenby, R. Larson, S. Ringer, S. Johnston, S. Kravec, S. E. Showk, S. Fort, T. Lanham, T. Telleen-Lawton, T. Conerly, T. Henighan, T. Hume, S. R. Bowman, Z. Hatfield-Dodds, B. Mann, D. Amodei, N. Joseph, S. McCandlish, T. Brown, and J. Kaplan, "Constitutional ai: Harmlessness from ai feedback," 2022. [Page 24.]

[46] J. Wei, X. Wang, D. Schuurmans, M. Bosma, B. Ichter, F. Xia, E. Chi, Q. Le, and D. Zhou, "Chain-of-thought prompting elicits reasoning in large language models," 2023. [Pages 24 and 25.]

[47] S. Guo, B. Zhang, T. Liu, T. Liu, M. Khalman, F. Llinares, A. Rame, T. Mesnard, Y. Zhao, B. Piot, J. Ferret, and M. Blondel, "Direct language model alignment from online ai feedback," 2024. [Page 25.]

[48] E. Cheah, "Eagle 7b: Soaring past transformers with 1 trillion tokens across 100+ languages (rwkv-v5)," https://blog.rwkv.com/p/eagle-7b-soaring-past-transformers, 2024, accessed: 2024-04-17. [Page 28.]

[49] Y. Zheng, R. Zhang, J. Zhang, Y. Ye, Z. Luo, and Y. Ma, "Llamafactory: Unified efficient fine-tuning of 100+ language models," *arXiv preprint arXiv:2403.13372*, 2024. [Online]. Available: http://arxiv.org/abs/2403.13372 [Page 32.]

[50] W.-L. Chiang, L. Zheng, Y. Sheng, A. N. Angelopoulos, T. Li, D. Li, H. Zhang, B. Zhu, M. Jordan, J. E. Gonzalez, and I. Stoica, "Chatbot arena: An open platform for evaluating llms by human preference," 2024. [Page 37.]

[51] OpenAI. (2023) Gpt-4. Accessed: 2024-04-12. [Online]. Available: https://openai.com/research/gpt-4 [Page 37.]

[52] K. Hu. (2023) Chatgpt sets record for fastest-growing user base - analyst note. Accessed: 2024-05-21. [Online]. Available: https://www.reuters.com/technology/chatgpt-sets-record-fastest-growing-user-base-analyst-note-2023-02-01/ [Page 54.]

[53] European Parliament. (2023) EU AI Act: First Regulation on Artificial Intelligence. [Online]. Available: https://www.europarl.europa.eu/topics/en/article/20230601STO93804/eu-ai-act-first-regulation-on-artificial-intelligence [Page 55.]

[54] C. Metz. (2024) Ai is poised to consume a lot more electricity as it gets smarter. The Verge. [Online]. Available: https://www.theverge.com/24066646/ai-electricity-energy-watts-generative-consumption [Page 55.]

Chapter 7

Supporting materials

7.1 Source Code

The source code for the human and AI rating environments, as well as the final datasets can be found in this GitHub Repository.

7.2 Model

The final Mistral 7B distractor generator is available on Huggingface.

TRITA-EECS-EX-2024:441
Stockholm, Sweden 2024

www.kth.se