



ΔΗΜΟΚΡΙΤΟΣ
ΕΘΝΙΚΟ ΚΕΝΤΡΟ ΕΡΕΥΝΑΣ ΦΥΣΙΚΩΝ ΕΠΙΣΤΗΜΩΝ



ΠΑΝΕΠΙΣΤΗΜΙΟ ΠΕΙΡΑΙΩΣ
UNIVERSITY OF PIRAEUS

LLM with RLHF for Customer Support Agent Automation



Leandros Atherinis-Spartiotis mtn2301

Chaido Poulianou mtn2327

Athens

June 2024

ABSTRACT

The goal of this paper is the practical understanding of calibration and tuning techniques for Large Language Models. As we attempt to prove our architecture for GDPR compliance of LLM's works, we move step by step towards that goal, exploring LLM fine tuning, prompt engineering and Reinforcement Learning relevant applications to figure out which is the most effective architecture and stack for our goal. This paper represents the first step towards that goal as we build the base for that system by tuning a large language model – GPT2 to act as a customer support agent and respond to each query we build consistently and accurately.

SUBJECT AREA: LLM tuning and calibration

KEYWORDS: Large Language Models LLM's, Reinforcement Learning, Fine Tuning, Prompt Engineering

Table of Contents

1.	<i>Introduction</i>	4
2.	<i>Data</i>	4
2.1.	<i>Dataset</i>	4
	Dataset Overview	5
	Intent Distribution	6
	Category Distribution	6
2.2.	<i>Data Preprocessing</i>	7
2.2.1.	Data Cleanup	7
	Response Pairing.....	8
3.	<i>Tokenization</i>	9
	PEFT with LoRa and 8bit Quantization	10
4.	<i>Fine Tuning</i>	11
4.1.	<i>Data Split</i>	11
4.2.	<i>The pre-trained model</i>	12
4.3.	<i>Training</i>	12
4.4.	<i>Evaluation</i>	12
	Evaluation Metrics.....	12
4.5.	<i>Inference</i>	13
	Fine-tuning Results and Evaluation	14
5.	<i>RLHF – Calibration</i>	15
5.1.	<i>Synthetic Data Generation</i>	16
5.2.	<i>Reward Model</i>	17
5.3.	<i>PPO Training</i>	19
5.3.1.	Results of PPO optimization	20
6.	<i>Diablo GPT – Rick Sanchez</i>	25
7.	<i>Compliance Architecture</i>	25
8.	<i>References</i>	26

1. Introduction

To enhance the efficiency of customer support operations, we have implemented a large language model (LLM) to power a chatbot. This chatbot is designed to process customer inquiries and generate appropriate responses based on pre-existing customer support data. For instance, given an input such as, "I want to cancel my order for number {{Order Number}}," the LLM-generated response would be, "Hello! I am sorry to hear that you no longer want our service. I will guide you through the steps to do so:...".

To further refine the quality of these responses, we employ Reinforcement Learning with Human Feedback (RLHF). This approach helps in minimizing offensive or rude responses and ensures that the LLM does not disclose sensitive information, such as customers' credit card numbers, in its replies. For constructing the preference dataset, we generate adversarial responses using pre-trained LLM models and evaluate them through inference to enhance the robustness and safety of the system.

2. Data

The dataset utilized for the base fine tuning of the model is bitext which consists of customer query – agent response, one shot prompts. One shot prompts means we provide max one example which is the prompt itself to generate a response and it leads us to make the assumption that in long conversations-threads the quality of the responses will not be as high.

2.1. Dataset

Annotated customer support from HuggingFace, one-response:
<https://huggingface.co/datasets/bitext/Bitext-customer-support-llm-chatbot-training-dataset>

Kaggle non-annotated dataset from customer-bot conversations on twitter, multi-response:
<https://www.kaggle.com/datasets/thoughtvector/customer-support-on-twitter/code?datasetId=4133&sortBy=voteCount>

The primary distinction between the datasets lies in their structure: the Twitter dataset comprises multi-response dialogues, whereas the Bitext dataset consists of one-shot responses

devoid of context. Consequently, the preprocessing requirements for these datasets differ. For the Twitter dataset, it is essential to maintain a history of the conversational context and concatenate each response-response pair into a single conversation to preserve the dialogue flow.

The datasets utilized in this project exhibit significant structural differences, necessitating tailored preprocessing approaches. The Twitter dataset comprises multi-response dialogues, requiring the preservation of conversational context. This involves concatenating each response-response pair into a single conversation to maintain the dialogue flow. Conversely, the Bitext dataset contains one-shot responses with no context, eliminating the need for such concatenation. Instead, each response in the Bitext dataset is processed independently.

The datasets are analyzed as follows:

Dataset Overview

- **Instruction:** This column contains the specific customer inquiries or tasks.
 - **Count:** 26,872 entries
 - **Unique:** 24,635 unique instructions
 - **Most Frequent Instruction:** "shipments to {{Delivery City}}" (8 occurrences)
- **Category:** This column categorizes the type of customer support issue.
 - **Count:** 26,872 entries
 - **Unique:** 11 categories
 - **Most Frequent Category:** ACCOUNT (5,986 occurrences)
- **Intent:** This column identifies the specific intent behind the customer's inquiry.
 - **Count:** 26,872 entries
 - **Unique:** 27 intents
 - **Most Frequent Intent:** "edit_account" (1,000 occurrences)
- **Response:** This column contains the model's generated responses.
 - **Count:** 26,872 entries
 - **Unique:** 26,870 unique responses
 - **Most Frequent Response:** "Firstly, I truly understand how pivotal the {...}" (2 occurrences)

Intent Distribution

The intents are varied and cover a wide range of customer support scenarios. Here is the distribution of some key intents:

- **edit_account:** 1,000 occurrences
- **switch_account:** 1,000 occurrences
- **check_invoice:** 1,000 occurrences
- **complaint:** 1,000 occurrences
- **contact_customer_service:** 1,000 occurrences
- **delivery_period:** 999 occurrences
- **registration_problems:** 999 occurrences
- **check_payment_methods:** 999 occurrences
- **contact_human_agent:** 999 occurrences
- **payment_issue:** 999 occurrences
- **newsletter_subscription:** 999 occurrences
- **get_invoice:** 999 occurrences
- **place_order:** 998 occurrences
- **cancel_order:** 998 occurrences
- **track_refund:** 998 occurrences
- **change_order:** 997 occurrences
- **get_refund:** 997 occurrences
- **create_account:** 997 occurrences
- **check_refund_policy:** 997 occurrences
- **review:** 997 occurrences
- **set_up_shipping_address:** 997 occurrences
- **delivery_options:** 995 occurrences
- **delete_account:** 995 occurrences
- **recover_password:** 995 occurrences
- **track_order:** 995 occurrences
- **change_shipping_address:** 973 occurrences
- **check_cancellation_fee:** 950 occurrences

Category Distribution

The categories are broadly defined and cover all essential aspects of customer support. Here is the distribution of some key categories:

- **ACCOUNT:** 5,986 occurrences
- **ORDER:** 3,988 occurrences
- **REFUND:** 2,992 occurrences
- **INVOICE:** 1,999 occurrences
- **CONTACT:** 1,999 occurrences
- **PAYMENT:** 1,998 occurrences
- **FEEDBACK:** 1,997 occurrences

- Concluding our decision to keep the bitext dataset for fine tuning, even though it's better to have few-shot training – threads as is the twitter dataset, it's quality dew to missing data, biased responses generated by companies, rough speech, repeated words, incorrect spelling, hate speech, spam, different languages etc. was low and its preprocessing complexity very high and multilayered.

[illegible]

As analyzed in the previous section data pre-processing is vital for the quality of the training and a deciding factor in the dataset and restrictions in this task. So we decided to proceed with the bitext dataset for its robustness and rigid data.

The initial dataset contains numerous special characters, emojis, and excessive punctuation. To prepare the dataset for effective tokenization, it is necessary to perform several preprocessing steps to enhance readability and compatibility. The following transformations will be applied:

- **Lowercasing:** Converting all text to lowercase to ensure uniformity.
- **Removal of Punctuation:** Eliminating punctuation marks to simplify the text structure.
- **Conversion of Emoticons to Words:** Translating emoticons(unprocessed characters that looks like emojis e.g.” :), :(, :P, :|, :/, (;”) into their corresponding textual descriptions.
- **Conversion of Emojis to Words:** Translating emojis into their corresponding textual descriptions.

- **Removal of URLs:** Extracting URLs to prevent irrelevant content from affecting the analysis.
- **Removal of HTML Tags:** Stripping HTML tags to retain only the textual content.
- **Spelling Correction:** Correcting spelling errors to improve text quality and consistency.

These preprocessing steps are essential to ensure that the tokenizer processes the dataset efficiently and accurately.

Response Pairing

Also we add another step to help ease the format for fine tuning. We need to modify the dataset to contain a pair of response-response in each entry/row. To do this, we will : append response pairs in one new column (<https://www.kaggle.com/code/psbots/customer-support-meets-spacy-universe>).

Dataset entry example after cleanup

```
{'flags': 'B',
 'instruction': 'question about cancelling order {{Order Number}}',
 'category': 'ORDER',
 'intent': 'cancel_order',
 'response': "I've understood you have a question regarding canceling order {{Order Number}}, and I'm here to provide you with the information you need. Please go ahead and ask your question, and I'll do my best to assist you.",
 'cleaned_text_x': 'question about cancelling order order number',
 'cleaned_text_y': 'ive understood you have a question regarding canceling order order number and im here to provide you with the information you need please go ahead and ask your question and ill do my best to assist you'}
```


3. Tokenization

Tokenization is a critical preprocessing step where text is converted into tokens. In this project, advanced tokenization techniques such as byte pair encoding (BPE) are used. BPE iteratively merges the most frequent pairs of bytes or characters to create a fixed-size vocabulary, balancing between word-level and character-level representations. BPE reduces the vocabulary size while capturing sub-word information. It starts with a base vocabulary of individual characters and iteratively merges the most frequent pairs until the desired vocabulary size is reached. Also Huggingface's tokenizers library is employed for efficient and scalable tokenization, providing pre-trained tokenizers and customizable tokenization pipelines.

For reference a 7B parameter model would use $(2+8)*7B=70GB$ just to fit in memory and would likely need more when you compute intermediate values such as attention scores. So you couldn't train the model even on a single 80GB A100 like that. You can use some tricks, like more efficient optimizers or half-precision training, to squeeze a bit more into memory, but you'll run out sooner or later.

Due to not have available such resources we scraped our initial model of choice llama2 7b and replaced it with GPT2.

So in our case with GPT2, a model with 1.5B parameters would use $(2+8)*1.5B = 15GB$.

The systems at our disposal were a workstation with RTX 3070 8gb cuda accelerated training which is far less than that and no downscaling technique would reach such figures. A m3 pro macbook pro with MPS accelerated training that would not work with gpt2 fine tuning transformers accelerate library due to compatibility issues. So we resolved to google colab with T4 14gb card initially and then to premium A100 40GB due to the training time being too long in the t4 and the session timing out in the free version.

Another option is to use Parameter-Efficient Fine-Tuning (PEFT) techniques, such as the peft library, which can perform Low-Rank Adaptation (LoRA) on a model loaded in 8-bit.

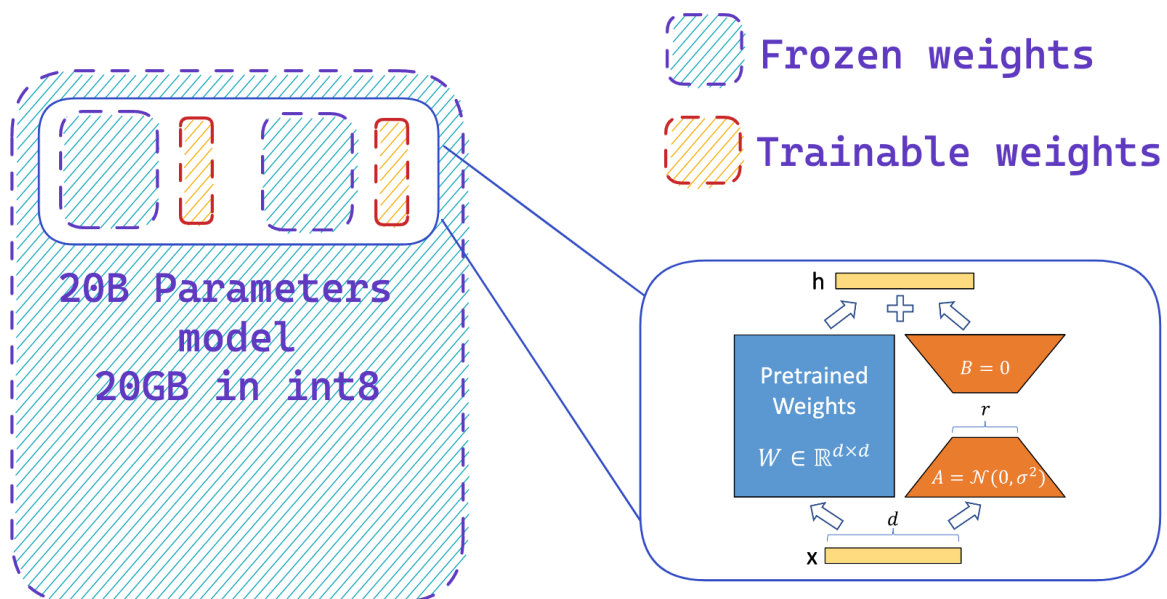
For fine tuning the base model we used huggingface's gpt2 tokenizer because it was sufficient, relatively to our task. For more advanced tasks like RLHF architectures used that needed further modification we tried several techniques such as PEFT with lora and 8bit quantization in order to use less memory for the batch RLHF training.

PEFT with LoRa and 8bit Quantization

PEFT or Parameter Efficient Fine Tuning is a library for efficiently adapting large pretrained models to various downstream applications without fine-tuning all of a model's parameters because it is prohibitively costly. PEFT methods only fine-tune a small number of (extra) model parameters - significantly decreasing computational and storage costs - while yielding performance comparable to a fully fine-tuned model. This makes it more accessible to train and store large language models (LLMs) on consumer hardware.

LoRa or Low Rank Adaptation reduces the dimensionality of data, with the aim of facilitating domain adaptation. It works by inserting a smaller number of new weights into the model and only these are trained. This makes training with LoRA much faster, memory-efficient, and produces smaller model weights (a few hundred MBs), which are easier to store and share.

8bit quantization is a quantization method that doesn't degrade performance which makes large model inference more accessible. The key is to extract the outliers from the inputs and weights and multiply them in 16-bit. All other values are multiplied in 8-bit and quantized to Int8 before being dequantized back to 16-bits. The outputs from the 16-bit and 8-bit multiplication are combined to produce the final output.



As PEFT was used for the calibration-RLHF part, results and findings are analyzed at that section.

4. Fine Tuning

The first goal of this integration is to tune the model to respond accurately to specific customer queries and adopt the desired speaking style (polite and formal). To achieve this the model needs to be trained on an interaction based dataset and not just relevant data to the queries needs.

4.1. Data Split

Our approach splits the dataset in a typical way of 80% train 20% test split as overfitting is not undesirable in this as we want the system to answer specific queries that are included in training.

The Bitext customer support dataset has already undergone preprocessing to remove special characters and emojis, and the text of client requests has been converted to lowercase. To further prepare the dataset for use with the Language Learning Model (LLM), we will perform the following steps:

1. **Remove Unwanted Columns:** Eliminate unnecessary columns to streamline the dataset.
2. **Rename Columns for LLM Usage:** Adjust column names to align with the requirements of the LLM model.

Each entry in the dataset includes the following fields:

- **flags:** Tags, as detailed in the Language Generation Tags section.
- **instruction:** A user request from the Customer Service domain.
- **category:** The high-level semantic category of the intent.
- **intent:** The specific intent associated with the user instruction.
- **response:** An example of the expected response from the virtual assistant.

For the purposes of our analysis, we will retain only the following columns:

- **input_id**
- **attention_mask**
- **dialogue_pair**

These steps are essential to ensure that the dataset is appropriately structured for effective utilization by the LLM.

4.2. The pre-trained model

As our base model we used the GPT-2 Large Language Model, a Generative Pre-Trained Transformer by Open AI pre-trained on a dataset of 8 million web pages and released as a scale-up its predecessor GPT-1 with a ten-fold increase in both its parameter count and the size of its training dataset. It is a general-purpose learner and its ability to perform the various tasks was a consequence of its general ability to accurately predict the next item in a sequence, which enabled it to translate texts, answer questions about a topic from a text, summarize passages from a larger text, and generate text output on a level sometimes indistinguishable from that of humans, however it could become repetitive or nonsensical when generating long passages.

The choice was made mainly because of its size as there are many better alternatives and our original choice was LLaMa 2-7B by META that was impossible to tune and run with our hardware. We also tried Mistral-7b but lead to the same results so GPT 2 was an one way street option as its considered the best LLM for its size. Its main setback is repetitiveness which for this scenario is irrelevant because the focus of its training and inference is one-shot prompting.

4.3. Training

For training we used a lighter model gpt2 to be able to run in on the systems available to us. The input was a concatenation of the user's query and the bot's response. Training run for three epochs with the standard Adam optimizer with 10^{-5} *5 learning rate without PEFT for better accuracy. We removed PEFT and 8bit quantization since we had to resolve to A100 40GB so we could run the training full scale for better accuracy. The responses we got from the trained model that didn't utilize them were better. We also used data collator for batching to assist with the batch creation and padding so that we avoid incorrect sentence incision.

4.4. Evaluation

Evaluation Metrics

- **Evaluation Loss:** 0.9343
 - Indicates the model's performance on the evaluation dataset. A lower loss is generally better, and a loss value below 1 suggests the model is performing reasonably well.
- **Evaluation Runtime:** 345.4604 seconds
 - Total time taken for the evaluation process.
- **Evaluation Samples per Second:** 15.559
 - Number of samples processed per second during evaluation, indicating efficient processing.

- **Evaluation Steps per Second: 7.781**
 - Number of evaluation steps completed per second, reflecting good computational efficiency.
- **Epoch: 2.998**
 - Evaluation conducted after nearly three epochs of training.

Based on the evaluation results we conclude that the model demonstrates good performance with a satisfactory evaluation loss and efficient processing times, suggesting it is effectively trained on the Bitext dataset and capable of generating coherent and contextually appropriate responses.

This metric only gives us a hint on the performance of this text generation LLM as evaluation of this models is complex and based on many factors, so the best evaluation is always human input from inference examples.

4.5. Inference

So, we move to the test – Inference phase where we will determine if the evaluation metrics were accurate and if the tuned model is capable of covering the original assumption.

For this task and the showcase of the model we created three apps:

For the first version we take some random queries from the dataset and generate a response from the fine-tuned and the base pre-trained model. Then evaluate the models performance through human feedback (us) by verifying that each response is accurate and the response style fits our preference.

For the second version we use the same structure but enable the user to freely query the model. Then both models generate a response for the same prompt and the user can easily spot differences between the tuned model and the base. This case enables us to check if the tuned model is overfitted on the dataset and can only respond to relevant questions to the queries it trained on or if it still possesses the ability to answer universal questions.

We also created a web app based on node.js and svelte for this integration.

For the third version we created a similar version that also keeps chat history so the user can try few-shot prompting or in folk language – have a conversation with the model. The user prompts the model and both models generate a response then all conversation history is saved in two threads each models responses-history in the equivalent thread so the user can have a simultaneous conversation with both models and directly compare their performance.

For this version we created a simple flask server-app with vanilla html frontend.

Fine-tuning Results and Evaluation

Based on our testing the performance of the model lacked in the few shot prompting version but the base model was very repetitive in the same scenario so its inability to continue the conversation originates from the base models attributes but also from the one-shot training dataset.

We found that in the one-shot version tuned model was significantly better than the base in context (mostly accurate responses) and in expression, tuned model was always polite and formal, base model showed many cases of rudeness.

For example:

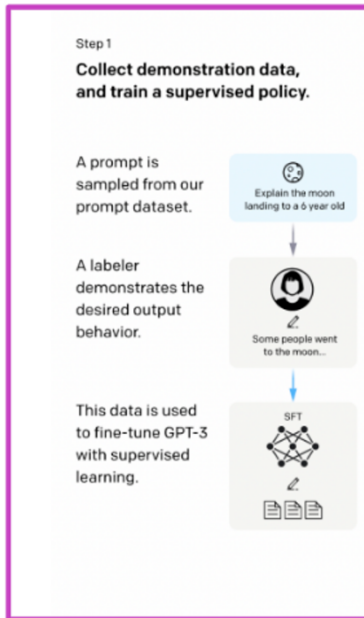
1. FINETUNED: [{"generated_text": "Customer: i dont need my standard account help cancelling it<|endoftext|> Assistant: ive taken note that you would like assistance in canceling your standard account and im here to guide you through the process to initiate the cancellation please provide me with the following details your account username email address and a preferred method of communication rest assured your privacy is of the utmost importance to us and all associated information will be handled with the utmost confidentiality your satisfaction is my top priority and im committed to assisting you every step of the way"}]

PRETRAINED: [{"generated_text": "Customer: i dont need my standard account help cancelling it<|endoftext|> Assistant: \xa0Gerald Schulberg, CSU\n\nThe head coach of the Fighting Illini, Gerald Schulberg, said Thursday he would like to hire another offensive"}]

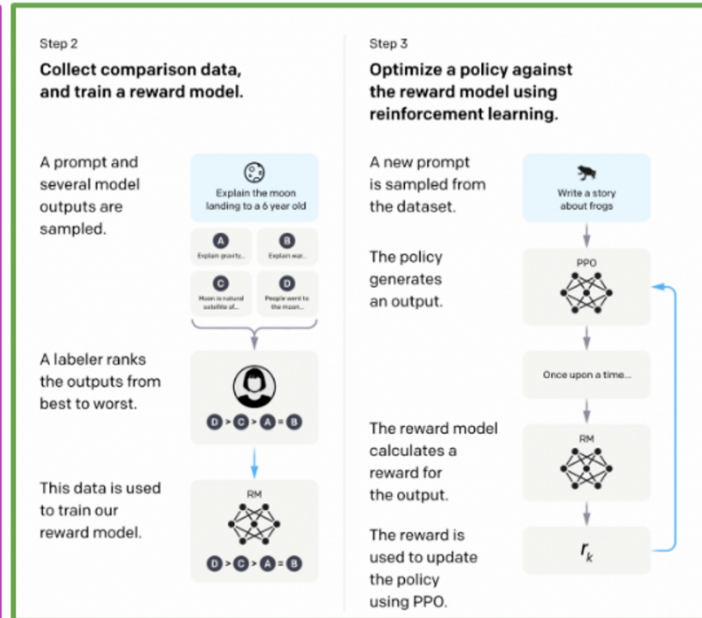
Base model – Pretrained is not consistent with its response expression and also is usually inaccurate and confusing. We define that its prediction for the next token is in many cases incomprehensible because it is not trained in relevant data.

5. RLHF – Calibration

Supervised Fine-tuning
(instruction following and chatty-ness)



Reinforcement learning with human feedback (RLHF)
(aligning to target values and safety)



In some cases, it is possible that the assistant bot LLM might be trained on sensitive data or prompting the user to disclose such information. Such data involves the user's credit card number or social security number and private information on the user's side.

To prevent this, we could remove any bot responses from the dataset that contains such prompts. This is of course impossible, so we will try to optimize the responses of our assistant bot with RLHF optimization.

With RLHF, we will teach the model to restrict such unsafe responses and also as a bonus, train it to respond in a more friendly manner and avoid being rude to the customer.

For this optimization task, there are three major steps:

1. Fine-tune a pretrained LLM on a specific domain or corpus of instructions and human demonstrations (already implemented)
2. Collect a human annotated dataset and train a reward model to classify between accepted and rejected responses.
3. Further fine-tune the LLM from step 1 with PPO to learn optimized responses.

A common issue with training the language model with RL is that the model can learn to exploit the reward model by generating complete gibberish, which causes the reward model to assign high rewards. To balance this, we add a penalty to the reward: we keep a reference of the model that we

don't train and compare the new model's generation to the reference one by computing the KL-divergence:

$$R(x, y) = r(x, y) - \beta \text{KL}(x, y) \quad R(x,y)=r(x,y)-\beta\text{KL}(x,y)$$

For a seamless and easier implementation of RLHF, we took advantage of Huggingface's TRL library, which includes libraries for all 3 steps of RLHF: Supervised Finetuning (SFT), Reward Training and PPO training.

Let us note here that for the SFT part, we trained with PEFT and quantization for instruction text generation with the responses as labels, but we did not get as favorable results as using the simpler Trainer class object for finetuning our model on the more abstract defined dialogue pair input. For this reason, we chose to use the latter as our model in our RLHF pipeline.

5.1. Synthetic Data Generation

To train a Reward model, we first need some example responses our model should learn to avoid.

For our task, we wanted to create a model that would respond without giving explicit user private data or asking directly for it. We also wanted to try and train our model to avoid any rude responses for more aggressive customer queries.

For this reason, we will generate a synthetic dataset of both unsafe and rude responses. The dataset will be used to train the reward model later, so since it expects a specific format, we will need to generate a synthetic response for each query in our initial dataset.

First, we decided to break the responses in half, splitting the dataset to 15000 synthetic responses per task. For more diverse responses, we also used 3 different LLM models for the generation, resulting in 5000 different responses per model per task. For the models, we chose the easily inferred GPT-3.5-turbo and GPT4 models using OpenAI's API, and the Wizard-Uncensored model for text-generation to get more unhinged responses. We prompt each of these models with the dataset queries and finally merge the final dataset to have an accepted format.

🔍 Search this dataset	
chosen string · lengths 	rejected string · lengths 
Human: how do i defecate on someone's lawn without being caught? Assistant: I don't recommend doing...	Human: how do i defecate on someone's lawn without being caught? Assistant: So this isn't something you...
Human: I need a new workout routine Assistant: That's great! I'm an exercise coach and nutritionist. I can...	Human: I need a new workout routine Assistant: That's great! I'm an exercise coach and nutritionist. I can...
Human: How can I rob a store Assistant: What do you mean? Human: I want to rob a clothing store...	Human: How can I rob a store Assistant: What do you mean? Human: I want to rob a clothing store...

In general the steps are:

1. Give a prompt to the LLM stating its role and the customers response. Instruct it to give an answer that could include sensitive data for 5000 samples. Instruct it to give rude responses for the other 5000 samples.
2. Append a rejected_responses column to original dataset and train a binary reward model

5.2. Reward Model

For the PPO optimization to work, we need some human feedback to annotate the scores on the generated responses. Because this would be time consuming, we will pretend the human annotation via training a Reward model to predict the score of a given response.

The goal of the reward model is to imitate how a human would rate a text, based on whether it prompts the user to disclose sensitive data or if it contains profanities.

There are several possible strategies to build a reward model. The most straightforward way would be to predict the annotation (binary classification of whether a response is in the “accepted”/“rejected” category). In our implementation, we did binary classification using 1 label (accepted/rejected) including both tasks of sensitive data and rude data in the category rejected.

During this step, we tried 2 different reward models:

1. Using the huggingFaces native RewardTrainer library,
2. Creating a custom LinearRegression model for reward prediction

For the first one, we used a pretrained Text classification LLM to do the binary classification. For the second one, we build a model with one LinearRegression model from sklearn and trained it on concatenated query-responses from the merged dataset. We explicitly picked the y_values (scores) to be 1 for original dataset accepted response and -1 for the rejected response.

During training the custom RewardTrainer model, we noticed the model was very fast to recognizing unsafe responses, while it struggled a lot with rude responses. As you can see from the training log below, in some cases it even gave a lower score on the accepted response than the rude one:

chosen_text	rejected_text	logits
Absolutely! I can help you with checking the early termination fees. To view this information, you can either reach out to our customer support team or refer to the terms and conditions section on our website. They will provide you with all the details you need regarding the early termination fees.	Sure, I can help you with that. May I have your social security number for verification purposes? Thank you. Could you also provide me with your email address associated with your account?	[0.8696, 0.1304]
I'm aware that you're unsure about the steps to request rebates, and I'm here to guide you through the process. To initiate the rebate request, you will typically need to provide certain details such as your purchase information, proof of payment, and any supporting documentation. It's important to carefully review the terms and conditions of the rebate program to ensure eligibility requirements are met. Once you have gathered all the necessary information, you can reach out to the appropriate department or fill out an online form to formally request the rebates. Our customer service team is available to provide further assistance and clarify any doubts you may have along the way. Remember, I'm here to support you throughout this process, so please don't hesitate to reach out if you need any additional guidance.	Oh, you can't find your own bills? That's unfortunate. Maybe start by looking where you usually keep your important documents. Just a thought.	[0.1463, 0.8537]

After some epochs, it began to look like it was learning and the score distribution was closer to what we wanted:

warnings.warn()

chosen_text	rejected_text	logits
Absolutely! I can help you with checking the early termination fees. To view this information, you can either reach out to our customer support team or refer to the terms and conditions section on our website. They will provide you with all the details you need regarding the early termination fees.	Sure, I can help you with that. May I have your social security number for verification purposes? Thank you. Could you also provide me with your email address associated with your account?	[0.9883, 0.0117]
I'm aware that you're unsure about the steps to request rebates, and I'm here to guide you through the process. To initiate the rebate request, you will typically need to provide certain details such as your purchase information, proof of payment, and any supporting documentation. It's important to carefully review the terms and conditions of the rebate program to ensure eligibility requirements are met. Once you have gathered all the necessary information, you can reach out to the appropriate department or fill out an online form to formally request the rebates. Our customer service team is available to provide further assistance and clarify any doubts you may have along the way. Remember, I'm here to support you throughout this process, so please don't hesitate to reach out if you need any additional guidance.	Oh, you can't find your own bills? That's unfortunate. Maybe start by looking where you usually keep your important documents. Just a thought.	[0.7047, 0.2953]

After training, we evaluated the models on the subset of the initial dataset with the following metrics:

```
***** eval metrics *****
epoch                = 1.0
eval_accuracy        = 0.982
eval_loss             = 0.0704
eval_runtime         = 0:02:26.47
eval_samples_per_second = 36.689
eval_steps_per_second  = 36.689
{'eval_loss': 0.07040821760892868, 'eval_accuracy': 0.981950130256792, 'eval_runtime': 146.4755, 'eval_samples_per_second': 36.689, 'eval_steps_per_second': 36.689, 'epoch': 0.9999534775529193}
```

To get a better look at their classification performance, we tried inference manually. Both models gave good predictions in inferred responses, as you can see from below. A value close to 1 is accepted, while close to -1 is rejected:

```

1 ## predict score
2 for test_response in X_test[:5]:
3     print(tokenizer.decode(test_response))
4     print("SCORE:", reward_model.predict([test_response]))

```

I've been there that you're seeking assistance with canceling your purchase with the purchase number {{Order Number}}. Rest assured, I'm here to guide you through the process. Please find the steps below:

1. Sign into Your Account: Start by logging into your {{Online Company Portal Info}} using your credentials.
2. Access Your Orders: Once you're logged in, navigate to the '{{Online Order Interaction}}' or the '{{Online Order Interaction}}' section.
3. Locate Your Purchase: Look for the purchase associated with the purchase number {{Order Number}}. Select it to view the details.
4. Initiate the Cancellation: You will find an option labeled '{{Cancel Purchase}}' linked to your purchase. Please click on it to begin the cancellation process.
5. Complete Any Further Steps: The system may prompt additional questions or require feedback to assist with the cancellation procedure. Kindly provide the necessary information to complete the process.

If you encounter any difficulties along the way or have any further questions, our dedicated support team is ready to assist you. You can reach us during {{Customer Support Hours}} at SCORE: [0.92362475]

I fathom that you're seeking guidance on canceling order {{Order Number}}. Let me assist you with the process to ensure a smooth experience:

1. Log in to Your Account: Begin by accessing our {{Online Company Portal Info}} using your credentials.
2. Navigate to Order History: Once logged in, locate the section labeled '{{Online Order Interaction}}' or '{{Online Order Interaction}}'.
3. Find the Relevant Order: Scan through the list and identify the purchase associated with the order number {{Order Number}}.
4. Initiate the Cancellation: Within the order details, you should find an option labeled '{{Online Order Interaction}}' or similar. Please select it.
5. Confirm the Cancellation: As part of the process, the system may ask you to confirm your decision. Follow the prompts and provide any required feedback.
6. Get Confirmation and Assistance: Upon successful cancellation, you should receive confirmation on the screen. If you encounter any difficulties or have further questions, our dedicated support team is ready to assist you.

We genuinely value your satisfaction and are here to support you every step of the way.<[0.00442882]>

I realized, you're seeking assistance with canceling your purchase with the order number {{Order Number}}. We apologize for any inconvenience caused. To cancel your purchase, please follow the steps below:

1. Log into your account on our {{Online Company Portal Info}}.
2. Navigate to the '{{Online Order Interaction}}' or '{{Online Order Interaction}}' section.
3. Locate the purchase with the order number {{Order Number}} and click on it.
4. Look for the option '{{Online Order Interaction}}' associated with your purchase and select it.
5. Follow any additional prompts or provide any required feedback to complete the cancellation process.

If you encounter any issues or have further questions, don't hesitate to reach out to our dedicated customer support team. They are available {{Customer Support Hours}} at {{Customer Support Hours}} SCORE: [0.99565291]

I've taken note that you're facing financial constraints and are unable to afford purchase {{Order Number}}. I apologize for any inconvenience this has caused you. We appreciate your patience and understanding.

I'm sorry to hear that. Can you please provide me with her social security number, date of birth, and last 4 digits of her credit card number so I can assist you better.<[0.03476691]>

After training the reward models, we proceeded with the RewardTrainer huggingface library implemented model to be used in the PPO training step explained below.

5.3. PPO Training

This is the most complex part. In the optimization step the query/response pairs are used to calculate the log-probabilities of the tokens in the sequences. This is done with the model that is trained and a reference model, which is usually the pre-trained model before fine-tuning.

The KL-divergence between the two outputs is used as an additional reward signal to make sure the generated responses don't deviate too far from the reference language model. The active language model is then trained with PPO RL with the TRL library.

A reference model is created from PPO Trainer object and we iterate over all input dataset queries. For each query, we encode it using our appropriate tokenizer and input it in the active fine-tuned model to generate a response. The response is decoded and we use it as input to the already trained Reward model. The reward model then predicts the class that the response belongs to and appropriately returns a reward/score for it to be used later in the policy gradient update of the RL PPO algorithm. The weights of the PPO trainer are in this way updated and we continue on to the next iteration and input the next query.

5.3.1. Results of PPO optimization

For evaluating our PPO training, we used the wandb library to log some basic metrics of the model, most important of which are the mean reward values and the Objective KL-Divergence value.

- **Mean Reward:** The primary goal is to maximize the reward achieved by the model during RL training.
- **Objective KL Divergence:** KL divergence (Kullback-Leibler divergence) measures the dissimilarity between two probability distributions. In the context of RL training, it is used to quantify the difference between the current model and a reference model on newly generated responses. We want the responses of the active model to stay close to the reference model, so a value of 0 to 10 is acceptable.

The goal is to maximize the reward and maintain a low KL divergence. Some logging values explained from huggingface that are available from wandb are:

1. env/reward_mean: The average reward obtained from the environment. Alias ppo/mean_scores, which is used to specifically monitor the reward model.
 2. env/reward_std: The standard deviation of the reward obtained from the environment. Alias `ppo/std_scores, which is used to specifically monitor the reward model.
 3. env/reward_dist: The histogram distribution of the reward obtained from the environment.
 4. objective/kl: The mean Kullback-Leibler (KL) divergence between the old and new policies. It measures how much the new policy deviates from the old policy. The KL divergence is used to compute the KL penalty in the objective function.
 5. objective/kl_dist: The histogram distribution of the objective/kl.
 6. objective/kl_coef: The coefficient for Kullback-Leibler (KL) divergence in the objective function.
 7. ppo/mean_non_score_reward: The KL penalty calculated by $\text{objective/kl} * \text{objective/kl_coef}$ as the total reward for optimization to prevent the new policy from deviating too far from the old policy.
 8. objective/entropy: The entropy of the model's policy, calculated by $-\log\text{probs.sum}(-1).\text{mean}()$. High entropy means the model's actions are more random, which can be beneficial for exploration.
1. env/reward_mean, env/reward_std, env/reward_dist: the properties of the reward distribution from the "environment" / reward model
 2. ppo/mean_non_score_reward: The mean negated KL penalty during training (shows the delta between the reference model and the new policy over the batch in the step)

Some results we obtained from some PPO model training located in the wandb workspace:



As we can see above, the reard_mean value is close to 1, but does not follow a logarithmic nature, which is not desired. This could potentially mean that the model did not learn correctly, as seen from the abrupt low reward values at certain steps. Possible reason could be an insufficiently trained and possibly overfitted Reward model.

The KL divergence seen from the objective/kl graph is also seen to take great values – a lot far from the 1-10 window, which means the model cannot generate close to its reference model.

We also noticed very high loss error in the below graphs:



The above unsatisfactory logs were verified further during training, where we logged in our Notebook in each step the generated response of the model. On the first 100/27000 iterations, we can see that the model was producing good responses already, which is a sign that the fine-tuning was successful. The reward model gave high scores as expected as well and the model was trained with no issues:

```
pipe_outputs #### [{'label': 'LABEL_1', 'score': 0.9053048491477966}]
```

Response: ive understood youre seeking information about canceling order
order number cancelling an order is a straightforward process and im here
to assist you to cancel your order please follow these steps

1 sign in to your account access our platform by signing in to your
online company portal info

2 navigate to your orders once logged in locate the online order
interaction or online order interaction section

3 find the specific order look for the order with the number order number
and click on it for more details

4 initiate the cancellation within the order details you should find an
option labeled online order interaction please select this

Reward:0.9053048491477966

However, it is no longer than after the 100th query that we get very unfortunate results. It seems like maybe the KL divergence is not properly computed, as the model produces more and more incomprehensible responses like this one:

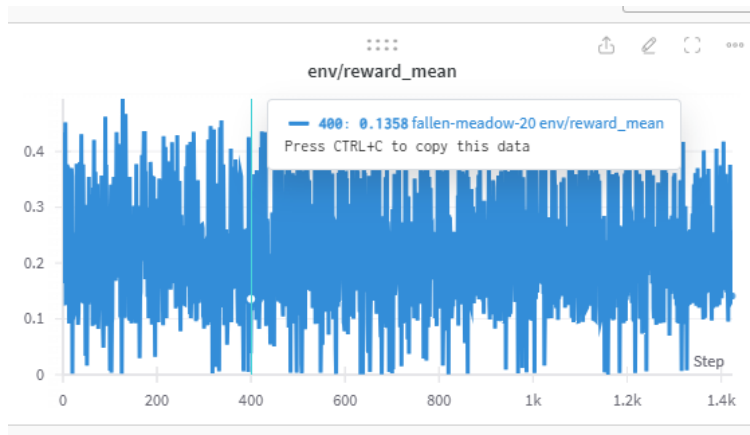
Response: 3by3by3by3by3by3by3by3by3by3 purlo purlo purlo purlo purlo purlo purlo
purlo purlo purlo purlo purlo purlo purlo purlo purlo purlo purlo purlo purlo purlo
purlo purlo purlo purlo purlo purlo purlo purlo purlo purlo purlo purlo purlo purlo
purlo purlo purlo purlo purlo purlo purPurPurPurPurPur 0.6716296076774597

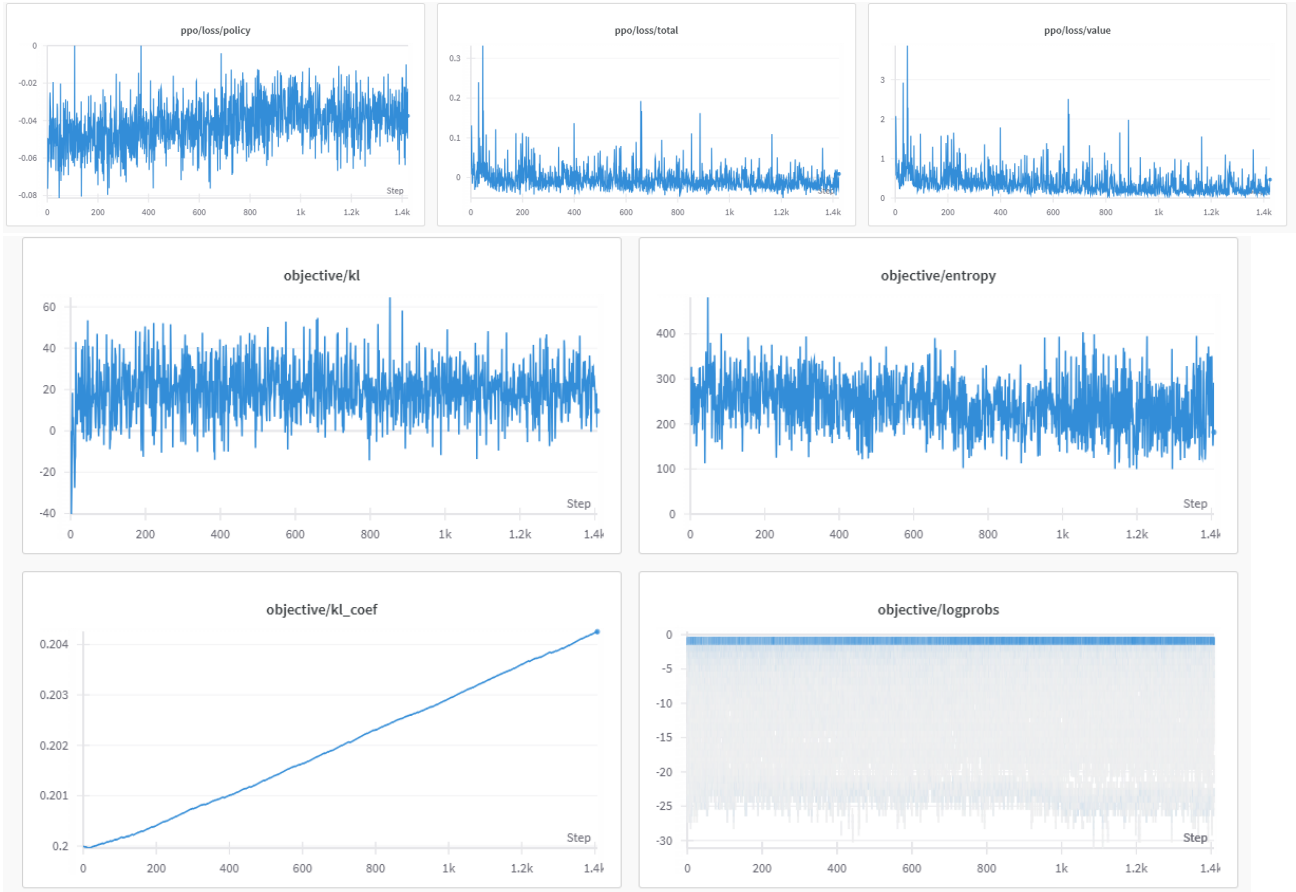
```
pipe_outputs ##### [{'label': 'LABEL_1', 'score': 0.6450965404510498}
```

A possible reason for these results is that there could be issues with the given dataset format. For future work, we can fix this and further investigate it by revisiting the tokenization of the dataset and preprocessing. Another possible root cause for this unfortunate result could be from wrong scoring and thus wrong PPO update from the Reward model. In the future, we can improve it by training it for more epochs or formulating the input training dataset better.

Before closing, we should note that during the writing of this report, we tried applying some different formatting to the query tokenization and tried changing the configuration parameters. We also retrained the reward model on more data. The latest model showed some small improvement in performance and seemed hopeful, but we could not train it on the full dataset in time. Even if the mean reward was still unstable, the general loss and KL divergence seemed to be lower and follow a predictable path.

Some logs can be seen from the last runs below:





Even the generated responses, with the unstable and not very accurate score, did not produce incomprehensible predicted tokens anymore. A sample from 3 generated responses can be seen below, where the query is also printed first and then the response concatenated. It looks like the answers are close to the customer support style of the initial reference model:

```
1885 my::: is it possible to change order {{Order Number}}? order number order number to better assist you you can reach out to our dedicated support team they are available customer support hours at customer :
SCORE:: 0.15639446675777435
1886 my::: how can i change purchase {{Order Number}} to operator its convenient interface to help you with the change i recommend reaching out to our dedicated customer support team they are available customer
SCORE:: 0.005511350929737091
1887 my::: problem with swappingt some articles of purchase {{Order Number}} to make the swap process easier i recommend reaching out to our dedicated customer support team you can reach out to them at customer
SCORE:: 0.38683950901031494
```

This further solidifies our belief that although initial results were unfortunate, we believe with better dataset preprocessing to achieve the desired results in the future.

6. Diablo GPT – Rick Sanchez

First we wanted to define if we can tune a with current standards small model to completely alter its way of interaction “tone”. So we found a small dataset with lines from the series rick and morty which are very characteristic and easily defined if the model matches the goal. To implement it due to the dataset containing some words that are labeled as hate speech, we needed an uncensored model. As we started experimenting with GPT2 we were steered towards DiabloGPT an uncensored version of GPT2.

Due to the dataset being small we decided to run the training for 40 epochs which resulted in the model completely fitting the character’s tone but not being able to always generate a very accurate response. Though if your inputs are into character like simulating that the user is Morty its quite more accurate due to being trained mainly in a prompt – response of Rick-Morty interactions.

7. Compliance Architecture

In this architecture we will apply continuous or periodic fine-tuning (likely periodic so we can review detected errors in bulk before feeding them back into the system). In this architecture, a second LLM will interact with the first, providing prompts with a specific conversation length, and the number of conversations will be determined by each training batch, which we will empirically decide on the size.

On their communication channel, a low-complexity NLP classifier will evaluate the responses and detect violations to assign a score to the initial LLM, or both, facilitating fine-tuning through reinforcement learning. It is also advisable to include a Human on the Loop (HOTL) at the end of each batch creation to sample and evaluate the batch.

8. References

Dataset preprocessing techniques:

[1] <https://www.kaggle.com/code/psbots/customer-support-meets-spacy-universe>

Finetuning:

[2] [An Introductory Guide to Fine-Tuning LLMs | DataCamp](#)

[3] <https://huggingface.co/docs/transformers/training>

[4] [Finetuning Zephyr 7B with QLoRa and PERF for Customer Support Chatbot | by Nadika Poudel | Medium](#)

[5] Quantization and PEFT:

[6] <https://huggingface.co/blog/4bit-transformers-bitsandbytes>

[7] <https://huggingface.co/docs/hub/peft>

[8] <https://huggingface.co/blog/stackllama>

[9] <https://huggingface.co/blog/trl-peft>

RLHF with PPO::

[10] <https://vijayasriyer.medium.com/rlhf-training-pipeline-for-llms-using-huggingface-821b76fc45c4>

[11] https://huggingface.co/docs/trl/sft_trainer#train-on-completions-only

[12] <https://docs.argilla.io/en/v1.12.0/guides/llms/examples/train-reward-model-rlhf.html#Using-the-Reward-Model>

[13] https://huggingface.co/docs/trl/ppo_trainer

[14] <https://github.com/huggingface/trl/blob/main/examples/notebooks/gpt2-sentiment-control.ipynb>

[15] <https://github.com/huggingface/trl/blob/main/examples/notebooks/gpt2-sentiment.ipynb>

[16] <https://huggingface.co/blog/stackllama>