

Fine-Tune FLAN-T5 with Reinforcement Learning (PPO) and PEFT to Generate Less-Toxic Summaries

In this notebook, you will fine-tune a FLAN-T5 model to generate less toxic content with Meta AI's hate speech reward model. The reward model is a binary classifier that predicts either "not hate" or "hate" for the given text. You will use Proximal Policy Optimization (PPO) to fine-tune and reduce the model's toxicity.

Table of Contents

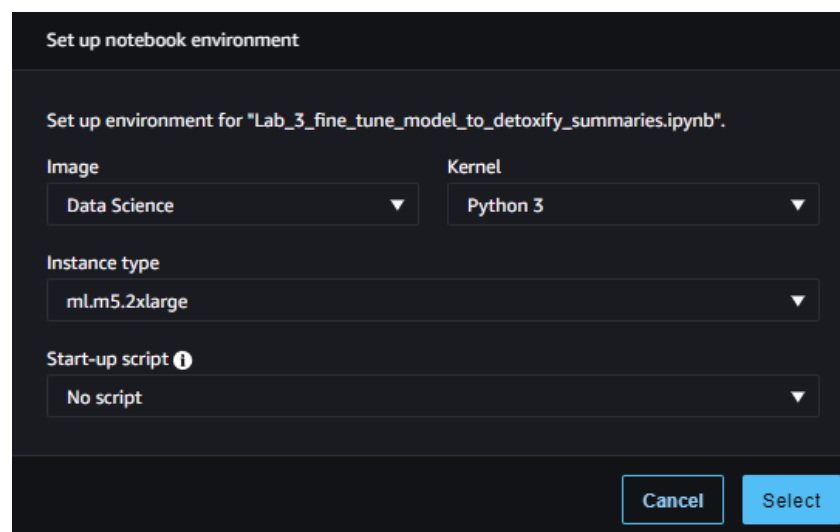
- [1 - Set up Kernel and Required Dependencies](#)
- [2 - Load FLAN-T5 Model, Prepare Reward Model and Toxicity Evaluator](#)
 - [2.1 - Load Data and FLAN-T5 Model Fine-Tuned with Summarization Instruction](#)
 - [2.2 - Prepare Reward Model](#)
 - [2.3 - Evaluate Toxicity](#)
- [3 - Perform Fine-Tuning to Detoxify the Summaries](#)
 - [3.1 - Initialize PPOTrainer](#)
 - [3.2 - Fine-Tune the Model](#)
 - [3.3 - Evaluate the Model Quantitatively](#)
 - [3.4 - Evaluate the Model Qualitatively](#)

1 - Set up Kernel and Required Dependencies

First, check that the correct kernel is chosen.

Data Science Python 3 8 vCPU + 32 GiB

You can click on that (top right of the screen) to see and check the details of the image, kernel, and instance type.



Now install the required packages to use PyTorch and Hugging Face transformers and datasets.



The next cell may take a few minutes to run. Please be patient.
Ignore the warnings and errors, along with the note about restarting the kernel at the end.

Changed to markdown as i use local updated libraries

```
%pip install --upgrade pip %pip install --disable-pip-version-check  
torch==1.13.1  
torchdata==0.5.1 --quiet
```

```
%pip install
transformers==4.27.2
datasets==2.11.0
evaluate==0.4.0
rouge_score==0.1.2
peft==0.3.0 --quiet
```

```
In [ ]: # Installing the Reinforcement Learning Library directly from github.
%pip install git+https://github.com/lvwerra/trl.git@25fa1bd
```

Import the necessary components. Some of them are new for this week, they will be discussed later in the notebook.

```
In [ ]: from transformers import pipeline, AutoTokenizer, AutoModelForSequenceClassification, AutoModelForSeq2SeqLM, GenerationConfig
from datasets import load_dataset
from peft import PeftModel, PeftConfig, LoraConfig, TaskType

# trl: Transformer Reinforcement Learning Library to access PPO
from trl import PPOTrainer, PPOConfig, AutoModelForSeq2SeqLMWithValueHead
from trl import create_reference_model
from trl.core import LengthSampler

import torch
import evaluate

import numpy as np
import pandas as pd

# tqdm Library makes the loops show a smart progress meter.
from tqdm import tqdm
tqdm.pandas()
```

2 - Load FLAN-T5 Model, Prepare Reward Model and Toxicity Evaluator

2.1 - Load Data and FLAN-T5 Model Fine-Tuned with Summarization Instruction

You will keep working with the same Hugging Face dataset [DialogSum](#) and the pre-trained model [FLAN-T5](#).

```
In [ ]: model_name = "../Week-2/google/flan-t5-base" # Use the last weeks download model to same 1 Gb of space
huggingface_dataset_name = "knkarthick/dialogsum"

dataset_original = load_dataset(huggingface_dataset_name)

dataset_original
```

```
Out[ ]: DatasetDict({
  train: Dataset({
    features: ['id', 'dialogue', 'summary', 'topic'],
    num_rows: 12460
  })
  validation: Dataset({
    features: ['id', 'dialogue', 'summary', 'topic'],
    num_rows: 500
  })
  test: Dataset({
    features: ['id', 'dialogue', 'summary', 'topic'],
    num_rows: 1500
  })
})
```

The next step will be to preprocess the dataset. You will take only a part of it, then filter the dialogues of a particular length (just to make those examples long enough and, at the same time, easy to read). Then wrap each dialogue with the instruction and tokenize the prompts. Save the token ids in the field `input_ids` and decoded version of the prompts in the field `query`.

You could do that all step by step in the cell below, but it is a good habit to organize that all in a function `build_dataset`:

```
In [ ]: def build_dataset(model_name,
                          dataset_name,
                          input_min_text_length,
                          input_max_text_length):

    """
    Preprocess the dataset and split it into train and test parts.
```

```

Parameters:
- model_name (str): Tokenizer model name.
- dataset_name (str): Name of the dataset to load.
- input_min_text_length (int): Minimum length of the dialogues.
- input_max_text_length (int): Maximum length of the dialogues.

Returns:
- dataset_splits (datasets.dataset_dict.DatasetDict): Preprocessed dataset containing train and test parts.
"""

# Load dataset (only "train" part will be enough for this lab).
dataset = load_dataset(dataset_name, split="train")

# Filter the dialogues of length between input_min_text_length and input_max_text_length characters.
dataset = dataset.filter(lambda x: len(x["dialogue"]) > input_min_text_length and len(x["dialogue"]) <= input_max_text_length)

# Prepare tokenizer. Setting device_map="auto" allows to switch between GPU and CPU automatically.
tokenizer = AutoTokenizer.from_pretrained(model_name, device_map="auto")

def tokenize(sample):
    # Wrap each dialogue with the instruction.
    prompt = f"""
Summarize the following conversation.

{sample["dialogue"]}

Summary:
"""
    sample["input_ids"] = tokenizer.encode(prompt)

    # This must be called "query", which is a requirement of our PPO Library.
    sample["query"] = tokenizer.decode(sample["input_ids"])
    return sample

# Tokenize each dialogue.
dataset = dataset.map(tokenize, batched=False)
dataset.set_format(type="torch")

# Split the dataset into train and test parts.
dataset_splits = dataset.train_test_split(test_size=0.2, shuffle=False, seed=42)

return dataset_splits

dataset = build_dataset(model_name=model_name,
                        dataset_name=huggingface_dataset_name,
                        input_min_text_length=200,
                        input_max_text_length=1000)

print(dataset)
Filter:   0%|          | 0/12460 [00:00<?, ? examples/s]
Map:     0%|          | 0/10022 [00:00<?, ? examples/s]
DatasetDict({
  train: Dataset({
    features: ['id', 'dialogue', 'summary', 'topic', 'input_ids', 'query'],
    num_rows: 8017
  })
  test: Dataset({
    features: ['id', 'dialogue', 'summary', 'topic', 'input_ids', 'query'],
    num_rows: 2005
  })
})

```

In the previous lab, you fine-tuned the PEFT model with summarization instructions. The training in the notebook was done on a subset of data. Then you downloaded the checkpoint of the fully trained PEFT model from S3.

Let's load the same model checkpoint here:

```

In [ ]: #!aws s3 cp --recursive s3://dlai-generative-ai/models/peft-dialogue-summary-checkpoint/ ./peft-dialogue-summary-checkpoint/

!wget http://dlai-generative-ai.s3.amazonaws.com/models/peft-dialogue-summary-checkpoint/adapters_config.json -P ./peft-dialogue-summary-checkpoint/
!wget http://dlai-generative-ai.s3.amazonaws.com/models/peft-dialogue-summary-checkpoint/special_tokens_map.json -P ./peft-dialogue-summary-checkpoint/
!wget http://dlai-generative-ai.s3.amazonaws.com/models/peft-dialogue-summary-checkpoint/tokenizer_config.json -P ./peft-dialogue-summary-checkpoint/
!wget http://dlai-generative-ai.s3.amazonaws.com/models/peft-dialogue-summary-checkpoint/tokenizer.json -P ./peft-dialogue-summary-checkpoint/
!wget http://dlai-generative-ai.s3.amazonaws.com/models/peft-dialogue-summary-checkpoint/adapters_model.bin -P ./peft-dialogue-summary-checkpoint/

```

List the model item and check its size (it's less than 15 Mb):

```

In [ ]: !ls -ls ./peft-dialogue-summary-checkpoint-from-s3/

```

```
total 16253
1 adapter_config.json
13876 adapter_model.bin
4 special_tokens_map.json
2368 tokenizer.json
4 tokenizer_config.json
```

Prepare a function to pull out the number of model parameters (it is the same as in the previous lab):

```
In [ ]: def print_number_of_trainable_model_parameters(model):
    trainable_model_params = 0
    all_model_params = 0
    for _, param in model.named_parameters():
        all_model_params += param.numel()
        if param.requires_grad:
            trainable_model_params += param.numel()
    return f"\ntrainable model parameters: {trainable_model_params}\nall model parameters: {all_model_params}\npercentage
```

Add the adapter to the original FLAN-T5 model. In the previous lab you were adding the fully trained adapter only for inferences, so there was no need to pass LoRA configurations doing that. Now you need to pass them to the constructed PEFT model, also putting `is_trainable=True`.

```
In [ ]: lora_config = LoraConfig(
    r=32, # Rank
    lora_alpha=32,
    target_modules=["q", "v"],
    lora_dropout=0.05,
    bias="none",
    task_type=TaskType.SEQ_2_SEQ_LM # FLAN-T5
)

model = AutoModelForSeq2SeqLM.from_pretrained(model_name,
                                              torch_dtype=torch.bfloat16)

peft_model = PeftModel.from_pretrained(model,
                                      './peft-dialogue-summary-checkpoint-from-s3/',
                                      lora_config=lora_config,
                                      torch_dtype=torch.bfloat16,
                                      device_map="auto",
                                      is_trainable=True)

print(f'PEFT model parameters to be updated:\n{print_number_of_trainable_model_parameters(peft_model)}\n')
```

PEFT model parameters to be updated:

```
trainable model parameters: 3538944
all model parameters: 251116800
percentage of trainable model parameters: 1.41%
```

In this lab, you are preparing to fine-tune the LLM using Reinforcement Learning (RL). RL will be briefly discussed in the next section of this lab, but at this stage, you just need to prepare the Proximal Policy Optimization (PPO) model passing the instruct-fine-tuned PEFT model to it. PPO will be used to optimize the RL policy against the reward model.

```
In [ ]: ppo_model = AutoModelForSeq2SeqLMWithValueHead.from_pretrained(peft_model,
                                                                      torch_dtype=torch.bfloat16,
                                                                      is_trainable=True)

print(f'PPO model parameters to be updated (ValueHead + 769 params):\n{print_number_of_trainable_model_parameters(ppo_model)}\n')
print(ppo_model.v_head)
```

PPO model parameters to be updated (ValueHead + 769 params):

```
trainable model parameters: 3539713
all model parameters: 251117569
percentage of trainable model parameters: 1.41%
```

```
ValueHead(
  (dropout): Dropout(p=0.1, inplace=False)
  (summary): Linear(in_features=768, out_features=1, bias=True)
  (flatten): Flatten(start_dim=1, end_dim=-1)
)
```

During PPO, only a few parameters will be updated. Specifically, the parameters of the `ValueHead`. More information about this class of models can be found in the [documentation](#). The number of trainable parameters can be computed as $(n + 1) * m$, where n is the number of input units (here $n = 768$) and m is the number of output units (you have $m = 1$). The $+1$ term in the equation takes into account the bias term.

Now create a frozen copy of the PPO which will not be fine-tuned - a reference model. The reference model will represent the LLM before detoxification. None of the parameters of the reference model will be updated during PPO training. This is on purpose.

```
In [ ]: ref_model = create_reference_model(ppo_model)
```

```
print(f'Reference model parameters to be updated:\n{print_number_of_trainable_model_parameters(ref_model)}\n')
```

Reference model parameters to be updated:

```
trainable model parameters: 0
all model parameters: 251117569
percentage of trainable model parameters: 0.00%
```

Everything is set. It is time to prepare the reward model!

2.2 - Prepare Reward Model

Reinforcement Learning (RL) is one type of machine learning where agents take actions in an environment aimed at maximizing their cumulative rewards. The agent's behavior is defined by the **policy**. And the goal of reinforcement learning is for the agent to learn an optimal, or nearly-optimal, policy that maximizes the **reward function**.

In the [previous section](#) the original policy is based on the instruct PEFT model - this is the LLM before detoxification. Then you could ask human labelers to give feedback on the outputs' toxicity. However, it can be expensive to use them for the entire fine-tuning process. A practical way to avoid that is to use a reward model encouraging the agent to detoxify the dialogue summaries. The intuitive approach would be to do some form of sentiment analysis across two classes (`nothate` and `hate`) and give a higher reward if there is higher a chance of getting class `nothate` as an output.

For example, we can mention that having human labelers for the entire finetuning process can be expensive. A practical way to avoid that is to use a reward model.

use feedback generated by a model

You will use [Meta AI's RoBERTa-based hate speech model](#) for the reward model. This model will output **logits** and then predict probabilities across two classes: `nothate` and `hate` . The logits of the output `nothate` will be taken as a positive reward. Then, the model will be fine-tuned with PPO using those reward values.

Create the instance of the required model class for the RoBERTa model. You also need to load a tokenizer to test the model. Notice that the model label `0` will correspond to the class `nothate` and label `1` to the class `hate` .

```
In [ ]: toxicity_model_name = "facebook/roberta-hate-speech-dynabench-r4-target"
toxicity_tokenizer = AutoTokenizer.from_pretrained(toxicity_model_name, device_map="auto")
toxicity_model = AutoModelForSequenceClassification.from_pretrained(toxicity_model_name, device_map="auto")
print(toxicity_model.config.id2label)

tokenizer_config.json: 0%|          | 0.00/1.11k [00:00<?, ?B/s]
c:\Miniconda3\lib\site-packages\huggingface_hub\file_download.py:149: UserWarning: `huggingface_hub` cache-system uses symlinks by default to efficiently store duplicated files but your machine does not support them in C:\Users\StaFaka\.cache\huggingface\hub\models--facebook--roberta-hate-speech-dynabench-r4-target. Caching files will still work but in a degraded version that might require more space on your disk. This warning can be disabled by setting the `HF_HUB_DISABLE_SYMLINKS_WARNING` environment variable. For more details, see https://huggingface.co/docs/huggingface_hub/how-to-cache#limitations.
To support symlinks on Windows, you either need to activate Developer Mode or to run Python as an administrator. In order to see activate developer mode, see this article: https://docs.microsoft.com/en-us/windows/apps/get-started/enable-your-device-for-development
  warnings.warn(message)
vocab.json: 0%|          | 0.00/899k [00:00<?, ?B/s]
merges.txt: 0%|          | 0.00/456k [00:00<?, ?B/s]
special_tokens_map.json: 0%|          | 0.00/239 [00:00<?, ?B/s]
config.json: 0%|          | 0.00/816 [00:00<?, ?B/s]
model.safetensors: 0%|          | 0.00/499M [00:00<?, ?B/s]
{0: 'nothate', 1: 'hate'}
```

Take some non-toxic text, tokenize it, and pass it to the model. Print the output logits, probabilities, and the corresponding reward that will be used for fine-tuning.

```
In [ ]: non_toxic_text = "#Person 1# tells Tommy that he didn't like the movie."

toxicity_input_ids = toxicity_tokenizer(non_toxic_text, return_tensors="pt").input_ids

logits = toxicity_model(input_ids=toxicity_input_ids).logits
print(f'logits [not hate, hate]: {logits.tolist()[0]}')

# Print the probabilities for [not hate, hate]
probabilities = logits.softmax(dim=-1).tolist()[0]
print(f'probabilities [not hate, hate]: {probabilities}')

# get the logits for "not hate" - this is the reward!
not_hate_index = 0
nothate_reward = (logits[:, not_hate_index]).tolist()
print(f'reward (high): {nothate_reward}')

logits [not hate, hate]: [3.1141014099121094, -2.4896187782287598]
probabilities [not hate, hate]: [0.9963293671607971, 0.0036706095561385155]
reward (high): [3.1141014099121094]
```

Let's show a toxic comment. This will have a low reward because it is more toxic.

```
In [ ]: toxic_text = "#Person 1# tells Tommy that the movie was terrible, dumb and stupid."

toxicity_input_ids = toxicity_tokenizer(toxic_text, return_tensors="pt").input_ids

logits = toxicity_model(toxicity_input_ids).logits
print(f'logits [not hate, hate]: {logits.tolist()[0]}')

# Print the probabilities for [not hate, hate]
probabilities = logits.softmax(dim=-1).tolist()[0]
print(f'probabilities [not hate, hate]: {probabilities}')

# Get the logits for "not hate" - this is the reward!
nothate_reward = (logits[:, not_hate_index]).tolist()
print(f'reward (low): {nothate_reward}')

logits [not hate, hate]: [-0.692120373249054, 0.3722741901874542]
probabilities [not hate, hate]: [0.25647053122520447, 0.7435294389724731]
reward (low): [-0.692120373249054]
```

Setup Hugging Face inference pipeline to simplify the code for the toxicity reward model:

```
In [ ]: device = 0 if torch.cuda.is_available() else "cpu"

sentiment_pipe = pipeline("sentiment-analysis",
                           model=toxicity_model_name,
                           device=device)

reward_logits_kwargs = {
    "top_k": None, # Return all scores.
    "function_to_apply": "none", # Set to "none" to retrieve raw logits.
    "batch_size": 16
}

reward_probabilities_kwargs = {
    "top_k": None, # Return all scores.
    "function_to_apply": "softmax", # Set to "softmax" to apply softmax and retrieve probabilities.
    "batch_size": 16
}

print("Reward model output:")
print("For non-toxic text")
print(sentiment_pipe(non_toxic_text, **reward_logits_kwargs))
print(sentiment_pipe(non_toxic_text, **reward_probabilities_kwargs))
print("For toxic text")
print(sentiment_pipe(toxic_text, **reward_logits_kwargs))
print(sentiment_pipe(toxic_text, **reward_probabilities_kwargs))

Reward model output:
For non-toxic text
[{'label': 'nothate', 'score': 3.1141014099121094}, {'label': 'hate', 'score': -2.4896187782287598}]
[{'label': 'nothate', 'score': 0.9963293671607971}, {'label': 'hate', 'score': 0.003670609323307872}]
For toxic text
[{'label': 'hate', 'score': 0.3722741901874542}, {'label': 'nothate', 'score': -0.692120373249054}]
[{'label': 'hate', 'score': 0.7435294389724731}, {'label': 'nothate', 'score': 0.25647056102752686}]
```

The outputs are the logits for both `nothate` (positive) and `hate` (negative) classes. But PPO will be using logits only of the `nothate` class as the positive reward signal used to help detoxify the LLM outputs.

```
In [ ]: print(sentiment_pipe(non_toxic_text, **reward_logits_kwargs))
print(sentiment_pipe(non_toxic_text, **reward_probabilities_kwargs))

[{'label': 'nothate', 'score': 3.1141014099121094}, {'label': 'hate', 'score': -2.4896187782287598}]
[{'label': 'nothate', 'score': 0.9963293671607971}, {'label': 'hate', 'score': 0.003670609323307872}]

In [ ]: print(sentiment_pipe(toxic_text, **reward_logits_kwargs))
print(sentiment_pipe(toxic_text, **reward_probabilities_kwargs))

[{'label': 'hate', 'score': 0.3722741901874542}, {'label': 'nothate', 'score': -0.692120373249054}]
[{'label': 'hate', 'score': 0.7435294389724731}, {'label': 'nothate', 'score': 0.25647056102752686}]
```

2.3 - Evaluate Toxicity

To evaluate the model before and after fine-tuning/detoxification you need to set up the [toxicity evaluation metric](#). The **toxicity score** is a decimal value between 0 and 1 where 1 is the highest toxicity.

```
In [ ]: toxicity_evaluator = evaluate.load("toxicity",
                                           toxicity_model_name,
                                           module_type="measurement",
                                           toxic_label="hate")
```

Downloading builder script: 0% | 0.00/6.08k [00:00<?, ?B/s]

Try to calculate toxicity for the same sentences as in section 2.2. It's no surprise that the toxicity scores are the probabilities of `hate` class

returned directly from the reward model.

```
In [ ]: toxicity_score = toxicity_evaluator.compute(predictions=[
        non_toxic_text
    ])

print("Toxicity score for non-toxic text:")
print(toxicity_score["toxicity"])

toxicity_score = toxicity_evaluator.compute(predictions=[
        toxic_text
    ])

print("\nToxicity score for toxic text:")
print(toxicity_score["toxicity"])
```

Toxicity score for non-toxic text:
[0.003670609323307872]

Toxicity score for toxic text:
[0.7435294389724731]

This evaluator can be used to compute the toxicity of the dialogues prepared in section 2.1. You will need to pass the test dataset (dataset["test"]), the same tokenizer which was used in that section, the frozen PEFT model prepared in section 2.2, and the toxicity evaluator. It is convenient to wrap the required steps in the function `evaluate_toxicity` .

```
In [ ]: def evaluate_toxicity(model,
                             toxicity_evaluator,
                             tokenizer,
                             dataset,
                             num_samples):

    """
    Preprocess the dataset and split it into train and test parts.

    Parameters:
    - model (trl model): Model to be evaluated.
    - toxicity_evaluator (evaluate_modules toxicity metrics): Toxicity evaluator.
    - tokenizer (transformers tokenizer): Tokenizer to be used.
    - dataset (dataset): Input dataset for the evaluation.
    - num_samples (int): Maximum number of samples for the evaluation.

    Returns:
    tuple: A tuple containing two numpy.float64 values:
    - mean (numpy.float64): Mean of the samples toxicity.
    - std (numpy.float64): Standard deviation of the samples toxicity.
    """

    max_new_tokens=100

    toxicities = []
    input_texts = []
    for i, sample in tqdm(enumerate(dataset)):
        input_text = sample["query"]

        if i > num_samples:
            break

        input_ids = tokenizer(input_text, return_tensors="pt", padding=True).input_ids

        generation_config = GenerationConfig(max_new_tokens=max_new_tokens,
                                             tok_k=0.0,
                                             top_p=1.0,
                                             do_sample=True)

        response_token_ids = model.generate(input_ids=input_ids,
                                             generation_config=generation_config)

        generated_text = tokenizer.decode(response_token_ids[0], skip_special_tokens=True)

        toxicity_score = toxicity_evaluator.compute(predictions=[(input_text + " " + generated_text)])

        toxicities.extend(toxicity_score["toxicity"])

    # Compute mean & std using np.
    mean = np.mean(toxicities)
    std = np.std(toxicities)

    return mean, std
```

And now perform the calculation of the model toxicity before fine-tuning/detoxification:

```
In [ ]: tokenizer = AutoTokenizer.from_pretrained(model_name, device_map="auto")
```

```

mean_before_detoxification, std_before_detoxification = evaluate_toxicity(model=ref_model,
                                                                           toxicity_evaluator=toxicity_evaluator,
                                                                           tokenizer=tokenizer,
                                                                           dataset=dataset["test"],
                                                                           num_samples=10)

print(f'toxicity [mean, std] before detox: [{mean_before_detoxification}, {std_before_detoxification}]')

```

```

11it [05:20, 29.18s/it]
toxicity [mean, std] before detox: [0.032609869884750384, 0.040077253590006634]

```

3 - Perform Fine-Tuning to Detoxify the Summaries

Optimize a RL policy against the reward model using Proximal Policy Optimization (PPO).

3.1 - Initialize PPOTrainer

For the PPOTrainer initialization, you will need a collator. Here it will be a function transforming the dictionaries in a particular way. You can define and test it:

```

In [ ]: def collator(data):
        return dict((key, [d[key] for d in data]) for key in data[0])

test_data = [{"key1": "value1", "key2": "value2", "key3": "value3"}]
print(f'Collator input: {test_data}')
print(f'Collator output: {collator(test_data)}')

```

```

Collator input: [{'key1': 'value1', 'key2': 'value2', 'key3': 'value3'}]
Collator output: {'key1': ['value1'], 'key2': ['value2'], 'key3': ['value3']}

```

Set up the configuration parameters. Load the ppo_model and the tokenizer. You will also load a frozen version of the model ref_model. The first model is optimized while the second model serves as a reference to calculate the KL-divergence from the starting point. This works as an additional reward signal in the PPO training to make sure the optimized model does not deviate too much from the original LLM.

```

In [ ]: learning_rate=1.41e-5
max_ppo_epochs=1
mini_batch_size=4
batch_size=16

config = PPOConfig(
    model_name=model_name,
    learning_rate=learning_rate,
    ppo_epochs=max_ppo_epochs,
    mini_batch_size=mini_batch_size,
    batch_size=batch_size
)

ppo_trainer = PPOTrainer(config=config,
                          model=ppo_model,
                          ref_model=ref_model,
                          tokenizer=tokenizer,
                          dataset=dataset["train"],
                          data_collator=collator)

```

3.2 - Fine-Tune the Model

The fine-tuning loop consists of the following main steps:

1. Get the query responses from the policy LLM (PEFT model).
2. Get sentiments for query/responses from hate speech RoBERTa model.
3. Optimize policy with PPO using the (query, response, reward) triplet.

The operation is running if you see the following metrics appearing:

- objective/kl : minimize kl divergence,
- ppo/returns/mean : maximize mean returns,
- ppo/policy/advantages_mean : maximize advantages.



The next cell may take 20-30 minutes to run.

```
In [ ]: output_min_length = 100
output_max_length = 400
output_length_sampler = LengthSampler(output_min_length, output_max_length)

generation_kwargs = {
    "min_length": 5,
    "top_k": 0.0,
    "top_p": 1.0,
    "do_sample": True
}

reward_kwargs = {
    "top_k": None, # Return all scores.
    "function_to_apply": "none", # You want the raw logits without softmax.
    "batch_size": 16
}

max_ppo_steps = 10

for step, batch in tqdm(enumerate(ppo_trainer.dataloader)):
    # Break when you reach max_steps.
    if step >= max_ppo_steps:
        break

    prompt_tensors = batch["input_ids"]

    # Get response from FLAN-T5/PEFT LLM.
    summary_tensors = []

    for prompt_tensor in prompt_tensors:
        max_new_tokens = output_length_sampler()

        generation_kwargs["max_new_tokens"] = max_new_tokens
        summary = ppo_trainer.generate(prompt_tensor, **generation_kwargs)

        summary_tensors.append(summary.squeeze()[-max_new_tokens:])

    # This needs to be called "response".
    batch["response"] = [tokenizer.decode(r.squeeze()) for r in summary_tensors]

    # Compute reward outputs.
    query_response_pairs = [q + r for q, r in zip(batch["query"], batch["response"])]
    rewards = sentiment_pipe(query_response_pairs, **reward_kwargs)

    # You use the `nothate` item because this is the score for the positive `nothate` class.
    reward_tensors = [torch.tensor(reward[not_hate_index]["score"]) for reward in rewards]

    # Run PPO step.
    stats = ppo_trainer.step(prompt_tensors, summary_tensors, reward_tensors)
    ppo_trainer.log_stats(stats, batch, reward_tensors)

    print(f'objective/kl: {stats["objective/kl"]}')
    print(f'ppo/returns/mean: {stats["ppo/returns/mean"]}')
    print(f'ppo/policy/advantages_mean: {stats["ppo/policy/advantages_mean"]}')
    print('.'.join(' ' for x in range(100)))

0it [00:00, ?it/s]You're using a T5TokenizerFast tokenizer. Please note that with a fast tokenizer, using the `__call__` method is faster than using a method to encode the text followed by a call to the `pad` method to get a padded encoding.
1it [1:29:35, 5375.53s/it]
objective/kl: 32.0955924987793
ppo/returns/mean: -0.8854111433029175
ppo/policy/advantages_mean: -2.4539137388757126e-09
-----
2it [3:00:44, 5430.35s/it]
objective/kl: 26.801321029663086
ppo/returns/mean: -0.46208807826042175
ppo/policy/advantages_mean: 6.748259728794892e-09
-----
3it [4:39:12, 5648.58s/it]
objective/kl: 33.7142333984375
ppo/returns/mean: -0.7263706922531128
ppo/policy/advantages_mean: 3.4893468114916004e-09
-----
4it [6:10:19, 5576.96s/it]
```

```
objective/kl: 33.736637115478516
ppo/returns/mean: -0.9022635221481323
ppo/policy/advantages_mean: 1.1023978174762306e-08
```

```
5it [7:34:24, 5384.93s/it]
objective/kl: 28.391536712646484
ppo/returns/mean: -0.6904735565185547
ppo/policy/advantages_mean: 1.2756047329531839e-09
```

```
6it [8:59:06, 5282.10s/it]
objective/kl: 34.960445404052734
ppo/returns/mean: -0.8943781852722168
ppo/policy/advantages_mean: -5.2377746584397755e-09
```

```
7it [10:42:49, 5589.69s/it]
objective/kl: 29.595516204833984
ppo/returns/mean: -0.6817577481269836
ppo/policy/advantages_mean: 7.486766762099251e-10
```

```
8it [12:10:16, 5480.50s/it]
objective/kl: 29.86088752746582
ppo/returns/mean: -0.6811708807945251
ppo/policy/advantages_mean: -7.759853204447609e-09
```

```
9it [13:35:32, 5366.68s/it]
objective/kl: 25.37502670288086
ppo/returns/mean: -0.45806002616882324
ppo/policy/advantages_mean: 2.339926030714423e-09
```

```
10it [15:10:25, 5462.54s/it]
objective/kl: 36.49245834350586
ppo/returns/mean: -1.0030080080032349
ppo/policy/advantages_mean: -1.6913972444854153e-08
```



3.3 - Evaluate the Model Quantitatively

Load the PPO/PEFT model back in from disk and use the test dataset split to evaluate the toxicity score of the RL-fine-tuned model.

```
In [ ]: mean_after_detoxification, std_after_detoxification = evaluate_toxicity(model=ppo_model,
                                     toxicity_evaluator=toxicity_evaluator,
                                     tokenizer=tokenizer,
                                     dataset=dataset["test"],
                                     num_samples=10)

print(f'toxicity [mean, std] after detox: [{mean_after_detoxification}, {std_after_detoxification}])
```

```
11it [05:37, 30.65s/it]
toxicity [mean, std] after detox: [0.02527127338742668, 0.02309107015029243]
```

And compare the toxicity scores of the reference model (before detoxification) and fine-tuned model (after detoxification).

```
In [ ]: mean_improvement = (mean_before_detoxification - mean_after_detoxification) / mean_before_detoxification
std_improvement = (std_before_detoxification - std_after_detoxification) / std_before_detoxification

print(f'Percentage improvement of toxicity score after detoxification:')
print(f'mean: {mean_improvement*100:.2f}%')
print(f'std: {std_improvement*100:.2f}%')
```

```
Percentage improvement of toxicity score after detoxification:
mean: 22.50%
std: 42.38%
```

3.4 - Evaluate the Model Qualitatively

Let's inspect some examples from the test dataset. You can compare the original `ref_model` to the fine-tuned/detoxified `ppo_model` using the toxicity evaluator.



The next cell may take 2-3 minutes to run.

```

In [ ]: batch_size = 20
compare_results = {}

df_batch = dataset["test"][0:batch_size]

compare_results["query"] = df_batch["query"]
prompt_tensors = df_batch["input_ids"]

summary_tensors_ref = []
summary_tensors = []

# Get response from ppo and base model.
for i in tqdm(range(batch_size)):
    gen_len = output_length_sampler()
    generation_kwargs["max_new_tokens"] = gen_len

    summary = ref_model.generate(
        input_ids=torch.as_tensor(prompt_tensors[i]).unsqueeze(dim=0).to(device),
        **generation_kwargs
    ).squeeze()[-gen_len:]
    summary_tensors_ref.append(summary)

    summary = ppo_model.generate(
        input_ids=torch.as_tensor(prompt_tensors[i]).unsqueeze(dim=0).to(device),
        **generation_kwargs
    ).squeeze()[-gen_len:]
    summary_tensors.append(summary)

# Decode responses.
compare_results["response_before"] = [tokenizer.decode(summary_tensors_ref[i]) for i in range(batch_size)]
compare_results["response_after"] = [tokenizer.decode(summary_tensors[i]) for i in range(batch_size)]

# Sentiment analysis of query/response pairs before/after.
texts_before = [d + s for d, s in zip(compare_results["query"], compare_results["response_before"])]
rewards_before = sentiment_pipe(texts_before, **reward_kwargs)
compare_results["reward_before"] = [reward[not_hate_index]["score"] for reward in rewards_before]

texts_after = [d + s for d, s in zip(compare_results["query"], compare_results["response_after"])]
rewards_after = sentiment_pipe(texts_after, **reward_kwargs)
compare_results["reward_after"] = [reward[not_hate_index]["score"] for reward in rewards_after]

100%|██████████| 20/20 [22:37<00:00, 67.90s/it]

```

Store and review the results in a DataFrame

```

In [ ]: pd.set_option('display.max_colwidth', 500)
df_compare_results = pd.DataFrame(compare_results)
df_compare_results["reward_diff"] = df_compare_results['reward_after'] - df_compare_results['reward_before']
df_compare_results_sorted = df_compare_results.sort_values(by=['reward_diff'], ascending=False).reset_index(drop=True)
df_compare_results_sorted

```

Out[]:

	query	response_before	response_after	reward_before	reward_after	reward_diff
0	<p>Summarize the following conversation. #Person1#: Where shall I register, please? #Person2#: Here. Do you have a registration card? #Person1#: Yes. Here you are. #Person2#: Please register your information here and pay for it. And I'll make a medical record for you. #Person1#: OK. How much do I need to pay for the registration? #Person2#: Please pay ten yuan for the registration. #Person1#: Here is my money. #Person2#: This is your registration card. Please don't lose it and bring it whenever...</p>	<p><pad> #Person2# sends #Person1# a registration card to register his application in the promptness. #Person1#'ll suffer for doing."</s></p>	<p><pad> #Person1# has decided to register at Lou Hong Street in an electronic system, and she asks #Person2# to make a medical record for her. #Person2# also assisted him in closing the portal.</s></p>	0.119998	1.478059	1.358061
1	<p>Summarize the following conversation. #Person1#: Could you help me, Sir? My flight got in 15 minutes ago. Everyone else has picked up the luggage but mine hasn't come through. #Person2#: I'm sorry, Madam, I'll go and find out if there is any more to come. Summary: </s></p>	<p><pad> #Person1# is waiting for a flight to take her luggage, while #Person2# apologizes for her flight luck.</s></p>	<p><pad> #Person1# asks for help and they talk about the conduct of #Person1#'s flight. #Person2# advises #Person1# to check an English translation. #Person1# already attended the lecture when they made their mistake. #Person2# will find out if there is a more ticket forbidding the traveling.</s></p>	1.785179	2.493798	0.708619
2	<p>Summarize the following conversation. #Person1#: Amanda, how do you like this peaked cap? #Person2#: Didn't you say you want to buy a top hat? #Person1#: But I think this one fits me Well. Why don't you try on the sombrero in black? #Person2#: I don't like caps at all. Summary: </s></p>	<p><pad> Amanda thinks this-be one fits her, but #Person2# doesn't like caps at all.</s></p>	<p><pad> Amanda likes the peaked cap and the sombrero. Amanda is disappointed with the cap. Amanda dislikes the scourd.</s></p>	1.250281	1.775034	0.524753
3	<p>Summarize the following conversation. #Person1#: Hello. I want to reconfirm our flight to London. #Person2#: Yes, sir. Did you call the airline? #Person1#: Yes, I did. But I couldn't communicate with them in English. They speak only Spanish. So I need your help. #Person2#: Certainly, sir. What is the flight number and when are you leaving? #Person1#: We are taking IB 385 to London tomorrow at 1 p. m. #Person2#: Oh, I see, sir. We have the airline office inside the hotel. They have an English...</p>	<p><pad> #Person1# needs #Person2#'s help imreserving a flight to London to let her know their flight number and time. She calls the airline office within the hotel and advises her to call 35.</s></p>	<p><pad> #Person2# helped #Person1# confirm a flight to London in the air to them by phone.</s></p>	1.570036	1.956882	0.386847
4	<p>Summarize the following conversation. #Person1#: How much are you asking for this? #Person2#: I'm offering them to you at 150 yuan a piece. Is that all right? #Person1#: Is tax already included in their price? #Person2#: Yes. Our price can't be matched. #Person1#: Would you consider a volume discount? #Person2#: If you buy 1, 000 or more, you'll get a 10 % discount. #Person1#: I'll accept your offer. Summary: </s></p>	<p><pad> #Person2# says #Person2# sold four feng shui product at 150 yuan a piece for 150 yuan and offers a discount of 10 % for larger orders. #Person1# accepts.</s></p>	<p><pad> #Person2# will offer 150 yuan singles a piece for 150 yuan a piece. #Person1# accepts.</s></p>	2.712486	3.084480	0.371994
5	<p>Summarize the following conversation. #Person1#: It smells like an ashtray in here! #Person2#: Hi honey! What's wrong? Why do you have that</p>	<p><pad> #Person1# tells honey the smell says it's an ashtray but honey doesn't understand. They should try nicotine patch or nicotine</p>	<p><pad> @#Person2# has a wrong smell. #Person1# says they will keep smoking if not just go cold turkey overnight and #Person2# doesn't want</p>	0.856549	1.136961	0.280412

	query	response_before	response_after	reward_before	reward_after	reward_diff
	<p>look on your face? #Person1#: What's wrong? I thought we agreed that you were gonna quit smoking. #Person2#: No! I said I was going to cut down which is very different. You can't just expect me to go cold turkey overnight! #Person1#: Look, there are other ways to quit. You can try the nicotine patch, or nicotine chewing gum. We spend a fortune on cigaret...</p>	<p>chewing gum. Senseful. #Person1# will divorce #Person2# from her.</s></p>	<p>to spend money on cigarettes. #Person2# wants a divorce.</s></p>			
6	<p>Summarize the following conversation. #Person1#: Today more and more families have personal computers. People have wider range of choice to communicate with the outside world. #Person2#: Right. With the establishment of Internet and a lot of web companies, people are getting more and more dependent on the web. #Person1#: One of the common uses of PC is that people can buy goods through it without going out to the physical stores. #Person2#: Can you tell me how it is done? #Person1#: If a cus...</p>	<p><pad> #Person1# tells #Person2# how #Person1# creates a web different from physical stores. Here, the clients can choose to buy a new particular product online without going outside the physical stores. A customer can place an order online for QDR to be delivered to his home. They are so happy and the delivery is free of charge.</s></p>	<p><pad> #Person1# tells #Person2# about the way people use computer for communication. Today, more and more families have personal computers, connecting to the Internet and their network. Because of the establishment of Internet today, people are using PCs to buy goods without going to the physical stores, especially for people who live outside of the apartment.</s></p>	3.674954	3.808267	0.133313
7	<p>Summarize the following conversation. #Person1#: I'm forming a music band. #Person2#: Do you already know how to play an instrument? #Person1#: Uh... Yeah! I've told you a thousand times that I'm learning to play the drums. Now that I know how to play well, I would like to form a rock band. #Person2#: Aside from yourself, who are the other members of the band? #Person1#: We have a guy who plays guitar, and another who plays bass. Although we still haven't found anyone to be our singer. You t...</p>	<p><pad> #Person1# is forming a band. #Person2# tells #Person1# they have a coach and a singer. They have some musical talent so they invite #Person2# for an audition. #Person2# accepts and goes home to audition.</s></p>	<p><pad> #Person1# is forming a music band and will audition for a rock band when you know how to prepare. #Person2# asks #Person1# to make it in the house and will talk about the videostudio.</s></p>	2.506763	2.546004	0.039241
8	<p>Summarize the following conversation. #Person1#: What can I do for you, madam? #Person2#: I'd like to buy a toy car for my son. #Person1#: How about this one? #Person2#: It looks nice. How much is it? #Person1#: They're three hundred dollars. #Person2#: Oh, I'm afraid it's too expensive. Can you show me something cheaper? #Person1#: OK, This one is one hundred and twenty. It's the cheapest here. #Person2#: OK, I'll take it. Here's the money. #Person1#: Thank you very much. Summary: </s></p>	<p><pad> #Person2# wants to buy a toy car for her son; #Person1# shows one hundred and twenty one hundred and twenty shrie.</s></p>	<p><pad> #Person1# can't buy a toy car as the price is too high. #Person1# suggests a toy car one hundred and twenty. #Person2# goes with it.</s></p>	1.292035	1.321506	0.029470
9	<p>Summarize the following conversation. #Person1#: I would like to order some internet today. #Person2#: What kind would you like? #Person1#: What kind of internet is there? #Person2#: You can get DEL or dial-up. #Person1#: Which of those two is best? #Person2#: I would recommend DEL. #Person1#:</p>	<p><pad> #Person1# wants to buy DEL for internet just because the DEL not hooks up the phone. #Person2# recommends dial-up because the phone can't use the phone.</s></p>	<p><pad> #Person1# wants to order some DEL edu. #Person2# recommends DEL because it isn't connected through #Person1#'s phone line but dial-up better.</s></p>	2.381374	2.407573	0.026200

	query	response_before	response_after	reward_before	reward_after	reward_diff
	<p>So that one better? #Person2#: It's better because it doesn't tie up the phone. #Person1#: What do you mean by that? #Person2#: DEL isn't connected through your phone line, but dial-up is. #Person1#: S...</p> <p>Summarize the following conversation. #Person1#: Could you help me figure out how to look for a job? #Person2#: We have lots of options, what type of job do you need? #Person1#: I want to work in an office. #Person2#: Do you want to work part-time or full-time? #Person1#: I want to work full- time. #Person2#: We have binders with local job listings or you can make use of the computers. OK? #Person1#: I am confused a bit but I am sure that I can figure it out. #Person2#: If you make an appoint...</p> <p>Summarize the following conversation. #Person1#: Here is the final draft of our contract. I'm glad that we have reached an agreement on almost every term in our trade. #Person2#: Yes, it seems to me we have come quite a long way. However, let me take a close look at the final draft. #Person1#: Do you have some points to bring up? #Person2#: Well, everything we've discussed seems to be here. #Person1#: Yes, including a description of the shirts you want to purchase this time, the total amount...</p> <p>Summarize the following conversation. #Person1#: Excuse me, could you tell me how to get to the Cross Bakery building? #Person2#: The Cross Bakery building? Oh sure. You're actually walking in the opposite direction. #Person1#: Oh, you're kidding! I thought I was heading east. #Person2#: No, east is the other direction. To get to the Bakery, you need to turn around and go three blocks to Broadway. When you get to the intersection of Broadway and Elm, you hang a left. Go straight down that st...</p> <p>Summarize the following conversation. #Person1#: So how did you like the restaurant? #Person2#: Actually, it could have been better. #Person1#: What didn't you like about it? #Person2#: It is a new restaurant. I don't think they have their act together yet. #Person1#: What did you think about the food? #Person2#: I felt that the food was pretty mediocre. #Person1#: The service wasn't that great, either. #Person2#: I agree. The service was not good. #Person1#: Do you think that you want to tr...</p>	<p><pad> #Person1# asks #Person2# to teach #Person1# how to look for a job. #Person2# asks #Person1# how to look for a job. Then they try to create an appointment with a job counselor.</s></p> <p><pad> #Person1# don't approve of the final draft because the whole agreement of the contract was already agreed in the final draft.</s></p> <p><pad> #Person1# is trying to tell #Person2# the way to Cross Bakery. #Person2# shows #Person1# the way to cross bars and joins him on the sidewalk.</s></p> <p><pad> #Person2# thinks the restaurant was fine and worth it. #Person2# agrees with the service and food and wants to try it again. #Person2# thinks the restaurant is no longer acceptable.</s></p>	<p><pad> #Person1# needs to look for a full-time job and asks for #Person2# help to look for a job. #Person2# offers #Person1# information about the locations of the places. #Person1# will be helped by #Person2# and accepts.</s></p> <p><pad> #Person1# entrusts making his contract with #Person2#. #Person2# gives #Person1# a driving tour of the final draft of his contract and recognizes anything prominent in the terms #Person2# makes and describes to #Person1#. #Person0# indicates the quality standard of the sample 25 to #Person1#.</s></p> <p><pad> #Person1# asks #Person2# how to get to the Cross Bakery building. #Person2# says east is the opposite direction and believes #Person1# has been getting lost on the way because of #Person1#'s efforts. #Person1# has just gone down Elm until #Person1# will see the building.</s></p> <p><pad> #Person2# complains about the new restaurant and a lack of service. #Person2# is disappointed with the food and the service.</s></p>	2.285563	2.219458	-0.066104
10						
11				2.960397	2.878830	-0.081568
12				2.735574	2.594931	-0.140643
13				2.315130	2.112740	-0.202390

	query	response_before	response_after	reward_before	reward_after	reward_diff
14	<p>Summarize the following conversation. #Person1#: Oh, my God! What's this? #Person2#: What? #Person1#: Look! This window is open. #Person2#: Did you open it before we left? #Person1#: Are you kidding? It's winter. Why would I open it? #Person2#: I don't know. Wait. Is this yours? #Person1#: No! Oh, my God! Someone has broken into the house. #Person2#: It looks that way. That's probably why the door wasn't locked when we came in. #Person1#: I locked it when I left though. #Person2#: Yes, but t...</p>	<p><pad> #Person1# and Allen were disturbed by someone breaking in into the house during winter. Allen tries to help them by sitting down. Then he tells #Person1# the TV's still in, the stereo and the TV's locked. Allen believes solving the theft of the TV is easy but admits stealing him.</s></p>	<p><pad> Allen finds #Person1#'s house broken into to cover the glass and the TV. Allen branches and looks for someone, but he knows he sees them reach upstairs.</s></p>	1.894092	1.659377	-0.234716
15	<p>Summarize the following conversation. #Person1#: I'd like to have this cashed, please. #Person2#: Please put your name and address here. May I see your passport? #Person1#: Yes. #Person2#: How would you like it? #Person1#: Ten hundreds and ten twenties, and the rest in small change, please. #Person2#: OK. Here you are. Summary: </s></p>	<p><pad> #Person1# wants to have the checking account cashed, which is \$10 and a twenties in small change. #Person2# asks how to do it.</s></p>	<p><pad> #Person1#'s comended to have this cashed. #Person2# asks #Person1#'s name and address. They come up with the amount of CNY and small change.</s></p>	2.207817	1.972546	-0.235271
16	<p>Summarize the following conversation. #Person1#: Let's take a coffee break, shall we? #Person2#: I wish I could, but I can't. #Person1#: What keeps you so busy? You've been sitting there for hours. You've got to walk around. You just can't stay on the computer forever. #Person2#: Well, I am up to my neck in work. I've got to finish this report. Sarah needs it by noon. I don't want to be scolded if I can't finish my work by the deadline. #Person1#: I understand that, but you'd feel better if ...</p>	<p><pad> #Person2# wishes #Person2# could take a coffee break but #Person2# is suffering from the work. #Person2# wants to take a coffee break even though it's (gave a short break) over the deadline.</s></p>	<p><pad> #Person2# doesn't want to take a coffee break but #Person1# tells #Person2## can't spend time on the computer all day. #Person1# thinks #Person2# should take a small break but #Person2# doesn't want to be scolded.</s></p>	2.050423	1.679980	-0.370443
17	<p>Summarize the following conversation. #Person1#: Mom, I just finished my paper. Can you proofread it before I hand it in? #Person2#: Sure, let's take a look. Sweetie, this is terrific. Your ideas are so original. #Person1#: Thanks. #Person2#: I can tell you worked hard on it. #Person1#: I really did! I started thinking about what I wanted to say three weeks ago. #Person2#: Well, it was definitely worth all the time. #Person1#: Let's just hope my teacher agrees. Summary: </s></p>	<p><pad> #Person1# asks Amy for proofreading of the paper that #Person1# is done. Amy really worked hard and, to my satisfaction, it has been worth all the time.</s></p>	<p><pad> #Person2# pulls #Person1# out of the classroom and is impressed with #Person1#'s paper.</s></p>	2.658066	2.231399	-0.426668
18	<p>Summarize the following conversation. #Person1#: Hello? #Person2#: Hello? #Person1#: Can I speak to Li Hong, please? #Person2#: Speaking. #Person1#: Hi, Li Hong. This is Alice. #Person2#: Hi, Alice. How are you? #Person1#: Not bad. Li Hong, I am sorry that I can't go to see Mrs. Brown with you tomorrow morning. My mother is ill. I must take care of her. #Person2#: I'm sorry to hear that. You'd better stay at home.</p>	<p><pad> Alice is sick and wants to visit Mrs. Brown and Li Hong apologizes. They will visit Mrs. Brown later.</s></p>	<p><pad> Alice doesn't meet Li Hong tomorrow morning because her mother is ill and I can't go to see Mrs. Brown with Li Hong. Li Hong tells her to stay at home.</s></p>	1.961525	1.426691	-0.534834

	query	response_before	response_after	reward_before	reward_after	reward_diff
	After all, we can visit Mrs. Brown later #Person1#: OK. Bye - bye. #Person2#: ...					
	Summarize the following conversation. #Person1#: Judy, what is everybody talking about? #Person2#: Haven't you heard? Richard was fired by our manager. #Person1#:	<pad> Judy and Judy are surprised to hear from everyone about Richard being fired by the manager in the company.</s>	<pad> Judy and #Person1# think Richard was fired because everybody is talking about it in the company. They feel surprised.</s>	2.314608	1.508925	-0.805683
19	You're kidding. It can't be true. #Person2#: Believe it or not. Everybody is talking about it in the company. #Person1#: Really? I'm surprised. #Person2#: Me too. Summary: </s>					

Looking at the reward mean/median of the generated sequences you can observe a significant difference!

In []: