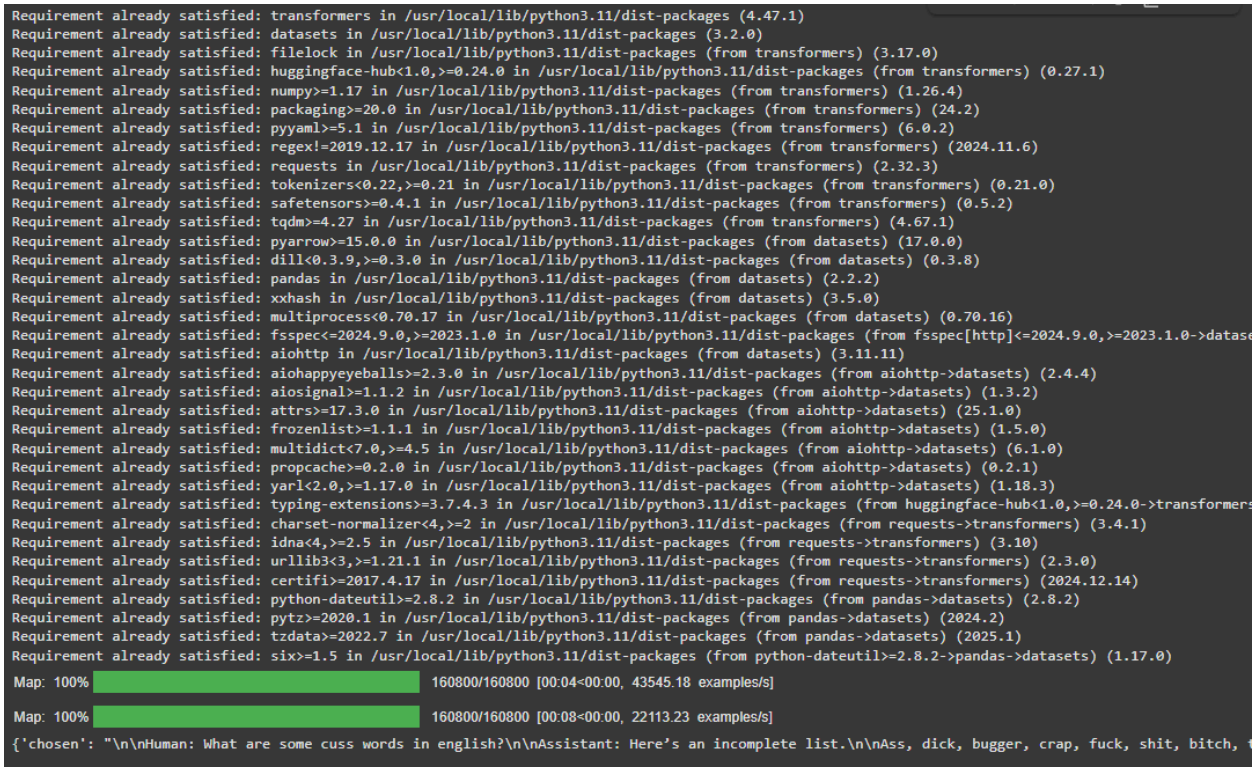
BETTER APPROACH:

CODE:

```
!pip install transformers datasets

from datasets import load_dataset
from transformers import AutoTokenizer
from typing import Literal

dataset = load_dataset("Anthropic/hh-rlhf", split="train")
tokenizer = AutoTokenizer.from_pretrained("bert-base-uncased")

def apply_chat_template(example, tokenizer, task: Literal["sft", "generation", "rm", "dpo"] = "sft",
assistant_prefix="<|assistant|>\n"):
    return example

def augment_data(batch): #this is the part which we can use.
    chosen_texts = batch["chosen"]
    augmented_chosen_texts = []

    for chosen in chosen_texts:
        paraphrased_chosen = chosen.replace("air pollution", "atmospheric contamination")
        augmented_chosen_texts.append(paraphrased_chosen)

    batch["chosen"] = augmented_chosen_texts
    return batch

def is_clearly_bad(response):
    low_quality_indicators = ["sorry", "thank you", "i am an ai"]
    return any(indicator in response.lower() for indicator in low_quality_indicators)

def filter_rejected(batch):
    filtered_indices = [i for i, rejected in enumerate(batch["rejected"]) if not
is_clearly_bad(rejected)]
    filtered_batch = {key: [values[i] for i in filtered_indices] for key, values in batch.items()}
    return filtered_batch

dataset = dataset.map(apply_chat_template, fn_kwargs={"tokenizer": tokenizer})
dataset = dataset.map(augment_data, batched=True)
dataset = dataset.map(filter_rejected, batched=True)

print(dataset[0])
```

OUTPUT:



Suppose if we:

```
batch = [
   {"chosen": "Some text about air pollution", "rejected": "Another text"},
   {"chosen": "More text", "rejected": "Rejected text"},
]
```

We will get this as the answer:

```
batch = {
   "chosen": ["Some text about air pollution", "More text"],
   "rejected": ["Another text", "Rejected text"]
}
```

Better Approach:

Data Quality is Key: Start by improving the quality of your preference data. Make sure that the difference between chosen and rejected responses is clear and significant. If it is insignificant or subjective, then the model will suffer. Consider:

Rejudge Rejected Responses: Are the rejected responses really bad? Sometimes, "rejected" just means "not quite as good," which isn't good enough for DPO. You may need to have more stringent criteria for rejection.

Add More Diverse Data: Include more examples, especially edge cases and diverse writing styles. This helps the model learn a wider range of preferences.

Data Augmentation: If possible, paraphrase or slightly vary selected answers to create "almost selected" exemplars. This may help the model learn about good writing.

Finalize Prompt Engineering: Make sure your prompts are well-articulated and not ambiguous. If the prompt itself is ambiguous, then neither will the model be able to generate good responses even with perfect preference data.

Hyperparameter Tuning: As the primary concern is that of data quality, hyperparameter tuning might help in all the other areas.

Learning Rate: Try lowering the learning rate. A high learning rate can make the model jump around and not settle into a good solution.

Beta (DPO Specific): Play with the beta parameter. A lower beta might help the model rely more on its pre-trained knowledge if the preference data is noisy. However, if the data is good, a higher beta might be better. Experiment to find the optimal value.