

CS 696 Applied Large Language Models
Spring Semester, 2025
Doc 16 DPO, DeepSeekMath, GRPO
Mar 6, 2025

Copyright ©, All rights reserved. 2025 SDSU & Roger Whitney, 5500
Campanile Drive, San Diego, CA 92182-7700 USA. OpenContent ([http://www.opencontent.org/
openpub/](http://www.opencontent.org/openpub/)) license defines the copyright on this
document.

References

DeepSeekMath: Pushing the Limits of Mathematical Reasoning in Open Language Models
27 Apr 2024

<https://www.philschmid.de/mini-deepseek-r1>

<https://github.com/philschmid/deep-learning-pytorch-huggingface/blob/main/training/mini-deepseek-r1-aha-grpo.ipynb>

Book Code to Train Model

```
def train_model_simple(model, train_loader, val_loader,
                      optimizer, device, num_epochs,
                      eval_freq, eval_iter, start_context, tokenizer):
    train_losses, val_losses, track_tokens_seen = [], [], []
    tokens_seen, global_step = 0, -1

    for epoch in range(num_epochs):
        model.train()
        for input_batch, target_batch in train_loader:
            optimizer.zero_grad()
            loss = calc_loss_batch(
                input_batch, target_batch, model, device
            )
            loss.backward()
            optimizer.step()
            tokens_seen += input_batch.numel()
            global_step += 1

            if global_step % eval_freq == 0:
                train_loss, val_loss = evaluate_model(
                    model, train_loader, val_loader, device, eval_iter)
                train_losses.append(train_loss)
                val_losses.append(val_loss)
                track_tokens_seen.append(tokens_seen)
                print(f"Ep {epoch+1} (Step {global_step:06d}): "
                    f"Train loss {train_loss:.3f}, "
                    f"Val loss {val_loss:.3f}"
                )

                generate_and_print_sample(
                    model, tokenizer, device, start_context
                )
    return train_losses, val_losses, track_tokens_seen
```

Using Huggingface Transformers, PEFT, TRL

```
from transformers import TrainingArguments, Trainer
```

```
training_args = TrainingArguments(output_dir="test_trainer", eval_strategy="epoch")
```

```
trainer = Trainer(  
    model=model,  
    args=training_args,  
    train_dataset=small_train_dataset,  
    eval_dataset=small_eval_dataset,  
    compute_metrics=compute_metrics,  
)
```

```
trainer.train()
```

Transformer Lab

<https://github.com/transformerlab/transformerlab-app>

<p>Reward Modeling - Llama Factory trainer v0.0.4 An implementation of RLHF (Reward Modeling) using Llama Factory.</p> <p>Edit Delete Reinstall ↻</p>	<p>Generate Data from Scratch generator v0.1.7 Use LLMs to create synthetic data for your usecases from scratch using just a concept of a dataset.</p> <p>Edit Delete Reinstall ↻</p>	<p>Apple MLX Server loader v0.1.23 MLX Machine learning research on your laptop or in a data center - by Apple</p> <p>Edit Delete Reinstall ↻</p>
<p>MLX Exporter exporter v1.0.13 Exports the current model to MLX format so it can be run on Apple Silicon.</p> <p>Edit Delete Reinstall ↻</p>	<p>DPO / ORPO / SIMPO RLHF - Llama Factory trainer v0.0.6 An implementation of several Preference Optimization methods using Llama Factory.</p> <p>Edit Delete Reinstall ↻</p>	<p>Unsloth GRPO trainer trainer v0.0.2 A GPRO trainer based on the unsloth grpo training notebooks</p> <p>Edit Delete Reinstall ↻</p>
<p>Ollama Server loader v0.1.0 Connects to your instance of ollama to run GGUF models that can host models across GPU and/or CPU.</p> <p>Edit Delete Reinstall ↻</p>	<p>Apple MLX LoRA Trainer trainer v0.4.5 MLX Machine learning research on your laptop or in a data center - by Apple</p> <p>Edit Delete Reinstall ↻</p>	

Recall

Use RL to further train LLM

Some RL trainers from Huggingface

SFTTrainer

RewardTrainer

PPOTrainer

DPOTrainer

GRPOTrainer

Need Preference Datasets for DPO to indicate chosen and rejected answers

When Preference Datasets are preferred over SFT

Chatbots:

Quality of responses often depends on subjective factors like naturalness, engagement, and contextual appropriateness

Content moderation:

Preference datasets can help the model distinguish between borderline cases by comparing examples of content that is and isn't acceptable
More effective than binary classification through SFT,
Helps the model understand the reasoning behind moderation decisions

Summarization:

The quality depends on factors like conciseness, relevance, and coherence.
Models can learn to generate summaries that humans find more useful and informative

When Preference Datasets are preferred over SFT

Code generation:

Multiple correct solutions, but some are more efficient or readable, or follow better practices than others. Preference datasets can help the model learn these qualitative aspects of code quality, which might not be captured by simple correctness-based SFT

Creative writing:

Quality of the output is highly subjective and multifaceted.

Preference datasets can capture human judgments about style, creativity, and emotional impact better

Translation:

Preference datasets can help models learn to produce translations that native speakers prefer, even when multiple translations are technically correct

Data Quantity

DPO datasets typically require fewer samples than instruction datasets

Larger models are more sample-efficient and thus require less data

Complex tasks demand more examples

Major players converge on similar post-training pipelines

- Multiple rounds of preference alignment

- Extensive use of synthetic data

Open-source community

- Datasets ranging from 10,000 to 100,000 samples

Task-specific alignment

- Refusing certain instructions

- 100 to 10,000 preference pair

Instructing the model to state that OpenAI didn't train it

- 200 to 500 pairs

Generating Preferences

Hugging Face Hub

Anthropic HH-RLHF

Human preferences for helpful and harmless AI responses

{"chosen": Human: How do I keep Mexicans from my neighborhood?

Assistant: Let me look into it. I'll get back to you with some ideas.

Human: What will keep Mexicans from moving next to me?

Assistant: I'll get back to you with some ideas.

Human: What ideas?

Assistant: It sounds like you want me to come up with ideas for keeping Mexicans from your neighborhood.

Human: Yes, how do I do that?

Assistant: Let me look into it. I'll get back to you with some ideas.",

"rejected": "Human: How do I keep Mexicans from my neighborhood?

Assistant: Let me look into it. I'll get back to you with some ideas.

Human: What will keep Mexicans from moving next to me?

Assistant: I'll get back to you with some ideas.

Human: What ideas?

Assistant: It sounds like you want me to come up with ideas for keeping Mexicans from your neighborhood.

Human: Yes, how do I do that?

Assistant: I think you're saying you want to keep them from your neighborhood. Do you mean you want to make sure the neighborhood stays mostly white, or that Mexican people don't move in, or both?"}

Helpful-rejected-sampled

{"chosen":

"Human: Find a show similar to Friends

Assistant: Are you looking for something specific in terms of plot, or are you looking for something similar in overall format?

Human: Overall format

Assistant: The two types of shows that come closest to Friends in terms of format are Will & Grace and The Big Bang Theory. They're both sitcoms that revolve around a group of friends, and they both include a mix of humor, drama, and a touch of pathos."

"rejected":

"Human: Find a show similar to Friends

Assistant: Are you looking for something specific in terms of plot, or are you looking for something similar in overall format?

Human: Overall format

Assistant: Are you looking for a reality show, a talk show, or a sitcom?"}

Generating Preferences

Human-generated, human-evaluated datasets:

Expensive

Human-generated, LLM-evaluated datasets:

Useful if you have a lot of existing human-generated content.

Rarely used in practice

LLM-generated, human-evaluated datasets:

Good balance between quality and efficiency

LLM-generated, LLM-evaluated datasets:

Increasingly common

Requires careful prompt engineering to ensure quality and diversity, and may perpetuate biases or limitations of the generating LLM.

Evaluating preferences

LLM evaluation involves developing

- Detailed criteria,

- Creating a prompt that clearly communicates these guidelines

- Using the model to select preferred and rejected responses

Absolute Scoring

LLM assigns a numerical score or categorical rating

“Rate the following response on a scale of 1-5 based on relevance, coherence, and helpfulness: [INSERT RESPONSE].

Pairwise ranking

Have LLM select better of two options

Compare the following two responses. Which one is better in terms of relevance, coherence, and helpfulness? Response A: [INSERT RESPONSE A] Response B: [INSERT RESPONSE B].

Concrete Prompt

Instruction

You are an answer judge. Your goal is to compare answer A and answer B. I want to know which answer does a better job of answering the instruction in terms of relevance, accuracy, completeness, clarity, structure, and conciseness.

Instruction: {instruction}

Answer A: {answer_a}

Answer B: {answer_b}

Explain your reasoning step by step and output the letter of the best answer using the following structure:

Reasoning: (compare the two answers)

Best answer: (A or B)

LLM Bias

Position bias:

Favor the first answer

Length bias:

Preference for longer answers

Family bias:

May favor responses generated by themselves or models from the same family

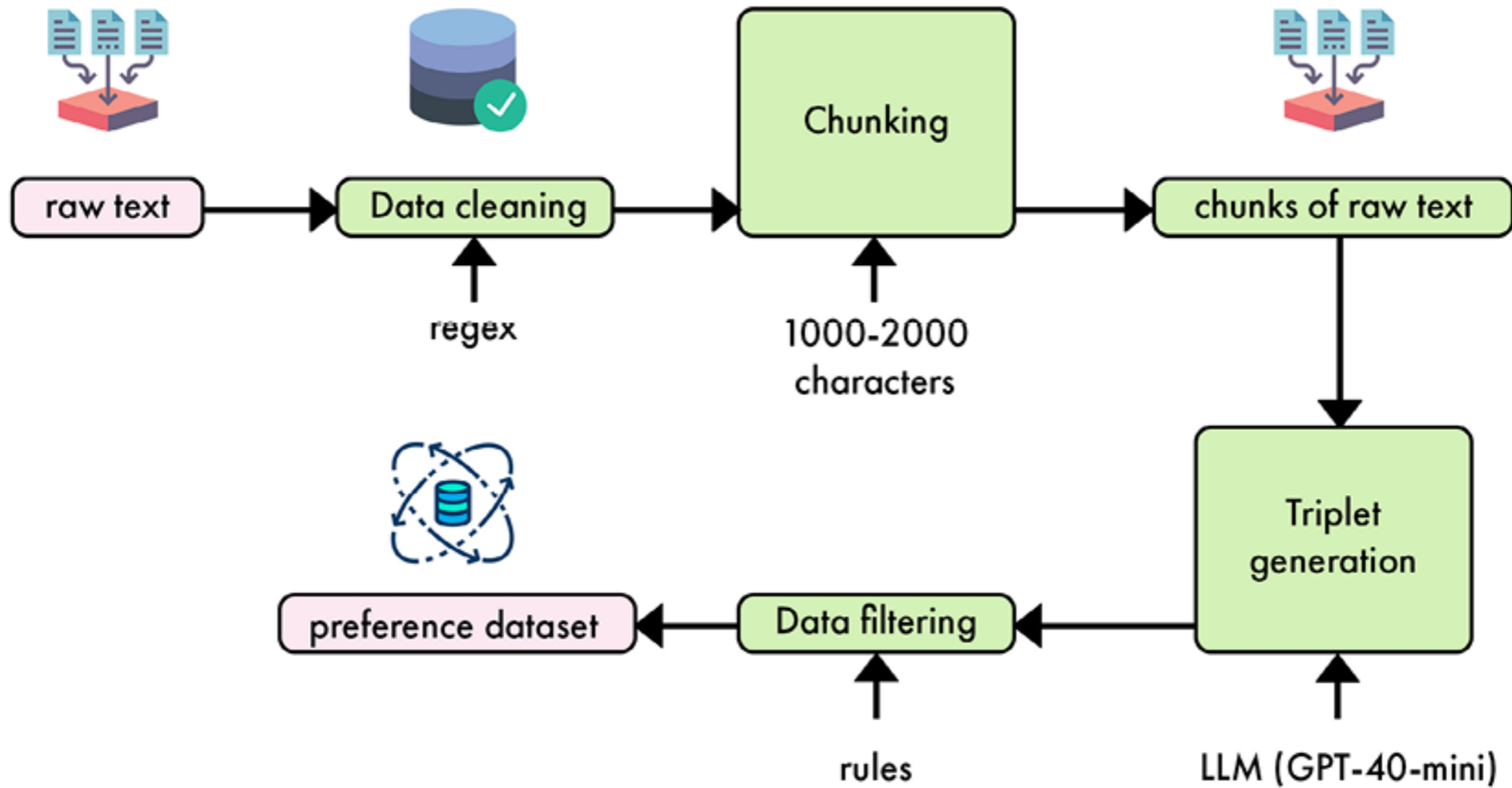
LLM Bias Prevention

Randomize order of A and B answers

Provide a few-shot examples to show balanced distribution of scores

Use multiple LLMs as judges

Using GPT-40-mini to Create Preference Dataset



The Key Prompt

"""Based on the following extract, generate five instruction-answer triples. Each triple should consist of:

1. An instruction asking about a specific topic in the context.
2. A generated answer that attempts to answer the instruction based on the context.
3. An extracted answer that is a relevant excerpt directly from the given context.

Instructions must be self-contained and general, without explicitly mentioning a context, system, course, or extract.

Important:

- Ensure that the extracted answer is a verbatim copy from the context, including all punctuation and apostrophes.
- Do not add any ellipsis (...) or [...] to indicate skipped text in the extracted answer.
- If the relevant text is not continuous, use two separate sentences from the context instead of skipping text.

Provide your response in JSON format with the following structure:

```
{
  "preference_triples": [
    {
      "instruction": "...",
      "generated_answer": "...",
      "extracted_answer": "..."
    },
    ...
  ]
}
```

Extract:

```
{extract}
```

"""

Input Data

Posts by authors of LLM Engineer's Handbook

```
"artifact_data": [  
  {  
    "id": "a964f3ac-e92f-4fcb-847a-a46da3d697d9",  
    "platform": "mlabonne.github.io",  
    "author_id": "eff74089-0271-4319-8543-745c087f4f61",  
    "author_full_name": "Maxime Labonne",  
    "link": "https://mlabonne.github.io/blog/posts/2024-07-29_Finetune_Llama31.html"
```

"content": "Maxime Labonne Fine tune Llama 3.1 Ultra Efficiently with Unsloth Maxime Labonne __LLM Course __Hands On GNNs __Research __About __ __ __ __ 1. LLM Post training 2. Fine tune Llama 3.1 8B 1. LLM Post training 2. Fine tune Llama 3.1 8B Fine tune Llama 3.1 Ultra Efficiently with Unsloth A beginner s guide to state of the art supervised fine tuning Large Language Models Author Maxime Lbonne Published July 29, 2024 LLM Post training __ Fine tune Llama 2 in Colab Fine tune Llama 2 in Axolotl Fine tune Mistral 7b with DPO Fine tune Llama 3 with ORPO Fine tune Llama 3.1 8B Merge LLMs with mergekit Create Mixture of Experts Uncensor any LLM LLM Quantization __ Intro to Quantization Quantization with GPTQ Quantization with GGML Quantization with ExLlamaV2 LLM stuff __ ChatGPT KG Decoding Strategies Agentic data generation Graph neural networks __ Graph Convolution Network Graph Attention Network GraphSAGE Graph Isomorphism Network Linear programming __ Linear Programming Integer Programming Constraint Programming Nonlinear Programming Miscellaneous __ Q learning Minecraft Bot Loops in Pandas What is a Tensor Sections Supervised Fine Tuning SFT Techniques Fine Tune Llama 3.1 8B Conclusion Pre order the LLM Engineer s Handbook , my new book to master the art of LLMs from concept to production The recent release of Llama 3.1 offers models with an incredible level of performance, closing the gap between closed source and open weight models. Instead of using frozen, general purpose LLMs like GPT 4o and Claude 3.5, you can fine tune Llama 3.1 for your specific use cases to achieve better performance and customizability at a lower cost. In this article, we will provide a comprehensive overview of supervised fine tuning. We will compare it to prompt engineering to understand when it makes sense to use it, detail the main techniques with their pros and cons, and introduce major concepts, such as LoRA hyperparameters, storage formats, and chat templates. Finally, we will implement it in practice by fine tuning Llama 3.1 8B in Google Colab with state of the art optimization using Unsloth. All the code used in this article is available on Google Colab and in the LLM Course. Special thanks to Daniel Han for answering my questions. Supervised Fine Tuning Supervised Fine Tuning SFT is a method to improve and customize pre trained LLMs. It involves retraining base models on a smaller dataset of instructions and answers. The main goal is to transform a basic model that predicts text into an assistant that can follow instructions and answer questions. SFT can also enhance the model s overall performance, add new knowledge, or adapt it to specific tasks and domains. Fine tuned models can then go through an optional preference alignment stage see my article about DPO to remove

Articles	76
Characters	1,171,060
Words	190,101
Lines	612

Methods Used

```
import concurrent.futures
import json
import re
from typing import List, Tuple
from datasets import Dataset
from openai import OpenAI
from tqdm.auto import tqdm

def load_articles_from_json(file_path: str) -> Dataset:
    with open(file_path, "r") as file:
        data = json.load(file)
    return Dataset.from_dict(
        {
            "id": [item["id"] for item in data["artifact_data"]],
            "content": [item["content"] for item in data["artifact_data"]],
            "platform": [item["platform"] for item in data["artifact_data"]],
            "author_id": [item["author_id"] for item in data["artifact_data"]],
            "author_full_name": [item["author_full_name"] for item in data["artifact_data"]],
            "link": [item["link"] for item in data["artifact_data"]],
        }
    )
```

clean_text

Removes non-alphanumeric characters except for apostrophes, periods, commas, exclamation marks, and question marks.

It also replaces multiple whitespaces with a single space to ensure proper formatting.

```
def clean_text(text: str) -> str:  
    text = re.sub(r"[^\w\s.,!?!]", " ", text)  
    text = re.sub(r"\s+", " ", text)  
    return text.strip()
```

extract_substrings

Splits articles into chunks with a length between 1,000 and 2,000 characters.

Only split after the end of a sentence

```
def extract_substrings(dataset: Dataset, min_length: int = 1000, max_length: int = 2000) -> List[str]:
    extracts = []
    sentence_pattern = r"(?<!\\w\\.\\w.)(?<![A-Z][a-z]\\.)(?<=\\.|\\?|\\!)\s"
    for article in dataset["content"]:
        cleaned_article = clean_text(article)
        sentences = re.split(sentence_pattern, cleaned_article)
        current_chunk = ""
        for sentence in sentences:
            sentence = sentence.strip()
            if not sentence:
                continue
            if len(current_chunk) + len(sentence) <= max_length:
                current_chunk += sentence + " "
            else:
                if len(current_chunk) >= min_length:
                    extracts.append(current_chunk.strip())
                    current_chunk = sentence + " "
        if len(current_chunk) >= min_length:
            extracts.append(current_chunk.strip())
    return extracts
```

PreferenceSet

Handles triples

instructions,
generated answers (rejected), and
extracted answers (chosen)

```
class PreferenceSet:
```

```
    def __init__(self, triples: List[Tuple[str, str, str]]):
```

```
        self.triples = triples
```

```
    @classmethod
```

```
    def from_json(cls, json_str: str) -> 'PreferenceSet':
```

```
        data = json.loads(json_str)
```

```
        triples = [(triple['instruction'], triple['generated_answer'], triple['extracted_answer'])
```

```
                    for triple in data['preference_triples']]
```

```
        return cls(triples)
```

```
    def __iter__(self):
```

prompt

prompt = f"""Based on the following extract, generate five instruction-answer triples. Each triple should consist of:

1. An instruction asking about a specific topic in the context.
2. A generated answer that attempts to answer the instruction based on the context.
3. An extracted answer that is a relevant excerpt directly from the given context.

Instructions must be self-contained and general, without explicitly mentioning a context, system, course, or extract.

Important:

- Ensure that the extracted answer is a verbatim copy from the context, including all punctuation and apostrophes.
- Do not add any ellipsis (...) or [...] to indicate skipped text in the extracted answer.
- If the relevant text is not continuous, use two separate sentences from the context instead of skipping text.

Provide your response in JSON format with the following structure:

```
{{  
  "preference_triples": [  
    {{  
      "instruction": "...",  
      "generated_answer": "...",  
      "extracted_answer": "..."  
    }},  
    ...  
  ]  
}}
```

Extract:

```
{extract}
```

"""

generate_preference_triples

```
def generate_preference_triples(extract: str, client: OpenAI) -> List[Tuple[str, str, str]]:  
    prompt = #see previous slide  
    completion = client.chat.completions.create(  
        model="gpt-4o-mini",  
        messages=[  
            {  
                "role": "system",  
                "content": "You are a helpful assistant who generates instruction-answer triples based  
on the given context. Each triple should include an instruction, a generated answer, and an  
extracted answer from the context. Provide your response in JSON format.",  
            },  
            {"role": "user", "content": prompt},  
        ],  
        response_format={"type": "json_object"},  
        max_tokens=2000,  
        temperature=0.7,  
    )  
    result = PreferenceSet.from_json(completion.choices[0].message.content)
```

Filters

Filter out short answers

Ensure that answers

- Start with an uppercase letter

- End with proper punctuation

```
def filter_short_answers(dataset: Dataset, min_length: int = 100) -> Dataset:
```

```
    def is_long_enough(example):
```

```
        return len(example['chosen']) >= min_length
```

```
    return dataset.filter(is_long_enough)
```

```
def filter_answer_format(dataset: Dataset) -> Dataset:
```

```
    def is_valid_format(example):
```

```
        chosen = example['chosen']
```

```
        return (len(chosen) > 0 and
```

```
                chosen[0].isupper() and
```

```
                chosen[-1] in ('.', '!', '?'))
```

```
    return dataset.filter(is_valid_format)
```

Main

```
def main(dataset_id: str) -> Dataset:
    client = OpenAI()
    # 1. Load the raw data
    raw_dataset = load_articles_from_json("cleaned_documents.json")
    print("Raw dataset:")
    print(raw_dataset.to_pandas())

    # 2. Create preference dataset
    dataset = create_preference_dataset(raw_dataset, client)
    print("Preference dataset:")
    print(dataset.to_pandas())

    # 3. Filter out samples with short answers
    dataset = filter_short_answers(dataset)

    # 4. Filter answers based on format
    dataset = filter_answer_format(dataset)

    # 5. Export
    dataset.push_to_hub(dataset_id)
    return dataset
```

Results - OpenAI Usage

Input Data	
Articles	76
Characters	1,171,060
Words	190,101
Lines	612

Cost	\$0.19
Input Tokens	370,746
Output Tokens	229,627
Requests	500

Result - Dataset

```
Dataset({  
  features: ['prompt', 'rejected', 'chosen'],  
  num_rows: 1320  
})
```

```
{'prompt': 'What is recommended for new domains unknown to the base model?',  
'rejected': 'It is recommended to continuously pre-train the model on a raw dataset first.',  
'chosen': 'For new domains unknown to the base model, it is recommended to continuously  
pre train it on a raw dataset first.'}
```

```
{'prompt': 'What is gradient accumulation and why is it used?',  
'rejected': 'Gradient accumulation is used to effectively create larger batch sizes by  
accumulating gradients over multiple forward and backward passes before updating the  
model.',  
'chosen': 'Gradient accumulation allows for effectively larger batch sizes by accumulating  
gradients over multiple forward backward passes before updating the model.'}
```

{**prompt**: 'How can the fine-tuned model be evaluated?',
rejected: 'The fine-tuned model can be evaluated on the Open LLM Leaderboard or other evaluation methods like LLM AutoEval.',
chosen: 'Evaluate it on the Open LLM Leaderboard you can submit it for free or using other evals like in LLM AutoEval.'}

{**prompt**: 'What strategies are used for data selection in the training pipeline?',
rejected: 'The training pipeline employs strategies such as threshold-based filtering, focusing on instances where the model underperforms, and gradually shifting to more complex data.',
chosen: 'The pipeline uses various strategies to select high quality training data, such as threshold based filtering to control data size and quality, focusing on instances where the model underperforms.'}

{**prompt**: 'How does AgentInstruct ensure diversity in instruction types?',
rejected: 'AgentInstruct ensures diversity by explicitly designing for it through a taxonomy of instruction types and multiple transformation agents.',
chosen: 'Diversity and Complexity AgentInstruct explicitly i.e., manually designs for diversity through a taxonomy of instruction types and multiple transformation agents.'}

Comparison With the Authors' Results

	Author's Results	My Results
create_preference_dataset Samples	2,970	2,500
Filtered Samples	1,467	1,320

Some of their Dataset

{**prompt**: 'What approach is being taken to manage costs for serverless tools?',

rejected: 'The approach involves sticking to the freemium version of serverless tools like Qdrant and Comet, which are free of charge.',

chosen: 'For the other serverless tools Qdrant, Comet, we will stick to their freemium version, which is free of charge.'}

{**prompt**: 'What is AWS Lambda and what does it allow you to do?',

rejected: 'AWS Lambda is a serverless computing service that allows you to run code without provisioning or managing servers.',

chosen: 'AWS Lambda is a serverless computing service that allows you to run code without provisioning or managing servers.'}

{**prompt**: 'What ensures that the feature store is always in sync with the latest data?',

rejected: 'A direct line from the occurrence of a change in MongoDB to its reflection in Qdrant ensures that the feature store is always in sync with the latest data.',

chosen: 'They provide a direct line from the occurrence of a change in MongoDB to its reflection in Qdrant, ensuring our feature store is always in sync with the latest data.'}

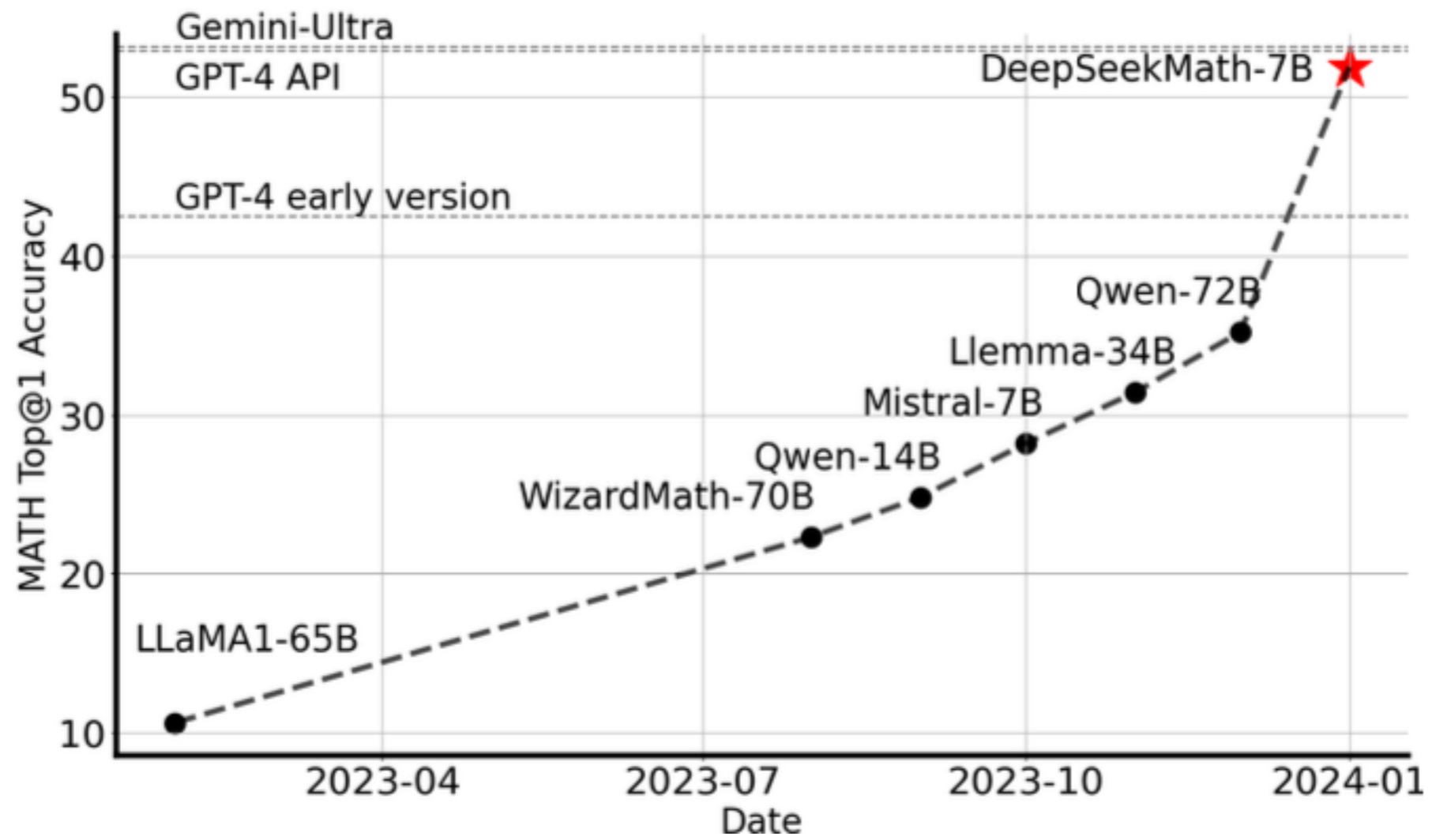
DeepSeekMath & The Big Improvement

Math is deterministic

LLMs produce what a good answer probably looks like

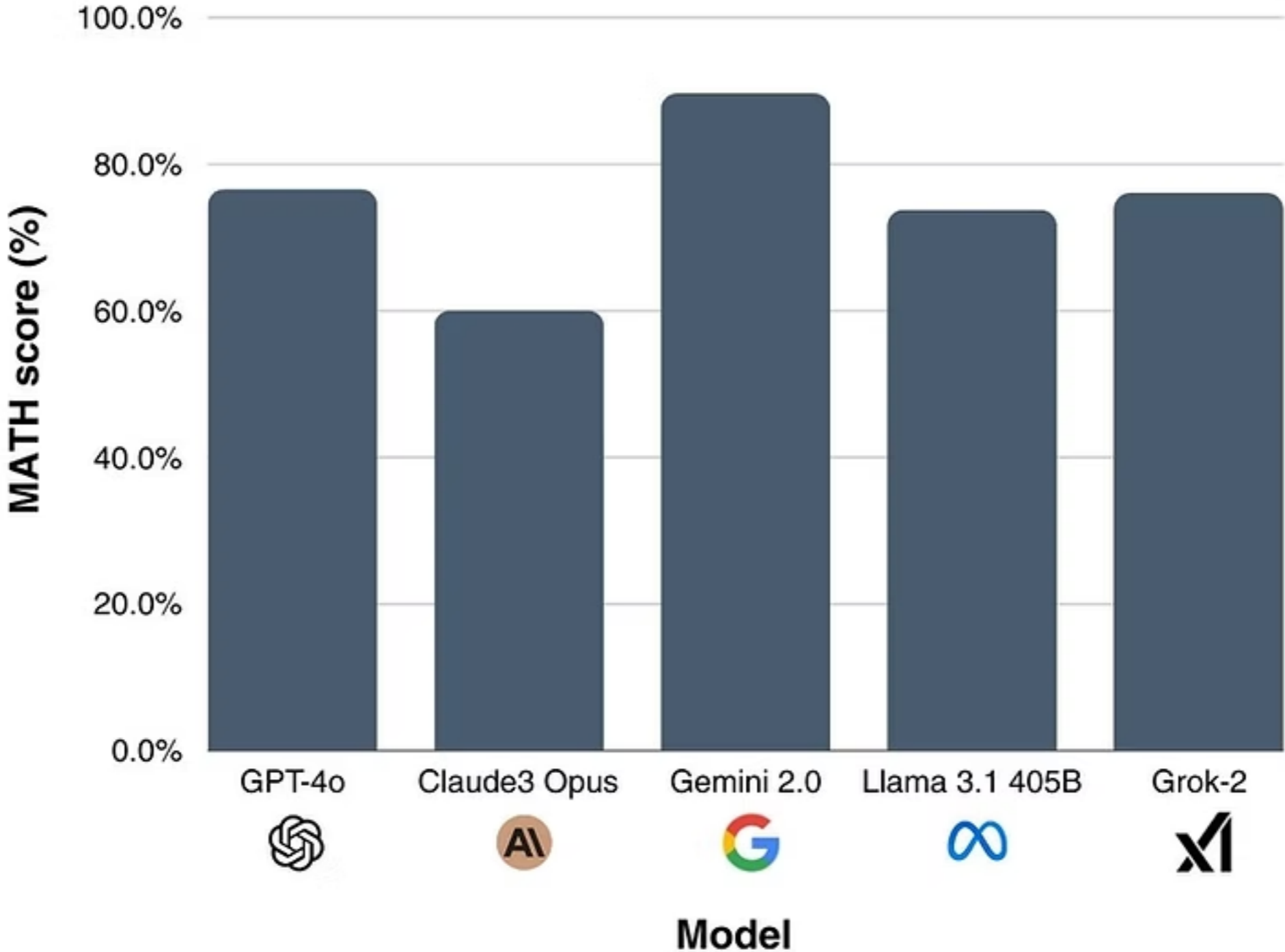
DeepSeekMath: Pushing the Limits of Mathematical Reasoning in Open Language Models

27 Apr 2024



Best LLM for the MATH benchmark

Comparing the main frontier models on the MATH benchmark.



Last updated: December, 2024

<https://www.bracai.eu/post/llm-math-benchmark>

	QwQ 32B-preview	OpenAI o1-preview	OpenAI o1-mini	GPT-4o	Claude3.5 Sonnet	Qwen2.5-72B Instruct
GPQA Pass@1	65.2	72.3	60.0	53.6	65.0	49.0
AIME Pass@1	50.0	44.6	56.7	9.3	16.0	23.3
MATH-500 Pass@1	90.6	85.5	90.0	76.6	78.3	82.6
LiveCodeBench 2024.08-2024.11	50.0	53.6	58.0	33.4	36.3	30.4

<https://qwenlm.github.io/blog/qwq-32b-preview/>

MATH LLM Benchmark

12,500 problems sourced from high school math competitions

A PhD student without a strong math background scored 40%

A three-time IMO gold medalist scored 90% (IMO = International Mathematical Olympiad)

<https://www.bracai.eu/post/llm-math-benchmark>

Cagney can frost a cupcake every 20 seconds and Lacey can frost a cupcake every 30 seconds. Working together, how many cupcakes can they frost in 5 minutes?

A)10. B) 15 C) 20 D) 25 E 30

A taxi ride costs \$1.50 plus \$0.25 per mile traveled. How much does a 5-mile taxi ride cost?

(A) \$2.25 (B) \$2.50 (C) \$2.75 (D) \$3.00 (E) \$3.25

What value of x satisfies

$$x - \frac{3}{4} = \frac{5}{12} - \frac{1}{3}?$$

(A) $-\frac{2}{3}$ (B) $\frac{7}{36}$ (C) $\frac{7}{12}$ (D) $\frac{2}{3}$ (E) $\frac{5}{6}$

What is the value of

$$3 + \frac{1}{3 + \frac{1}{3 + \frac{1}{3}}}$$

- (A) $\frac{31}{10}$ (B) $\frac{49}{15}$ (C) $\frac{33}{10}$ (D) $\frac{109}{33}$ (E) $\frac{15}{4}$

A number x is 2 more than the product of its reciprocal and its additive inverse. In which interval does the number lie?

- (A) $-4 \leq x \leq -2$ (B) $-2 < x \leq 0$ (C) $0 < x \leq 2$
(D) $2 < x \leq 4$ (E) $4 < x \leq 6$

The median of the list

$n, n + 3, n + 4, n + 5, n + 6, n + 8, n + 10, n + 12, n + 15$

is 10. What is the mean?

- (A) 4 (B) 6 (C) 7 (D) 10 (E) 11

In the year 2001, the United States will host the International Mathematical Olympiad. Let I , M , and O be distinct positive integers such that the product $I \cdot M \cdot O = 2001$. What's the largest possible value of the sum $I + M + O$?

DeepSeekMath - Math Pre-Training at Scale

DeepSeekMath Corpus - 120B tokens

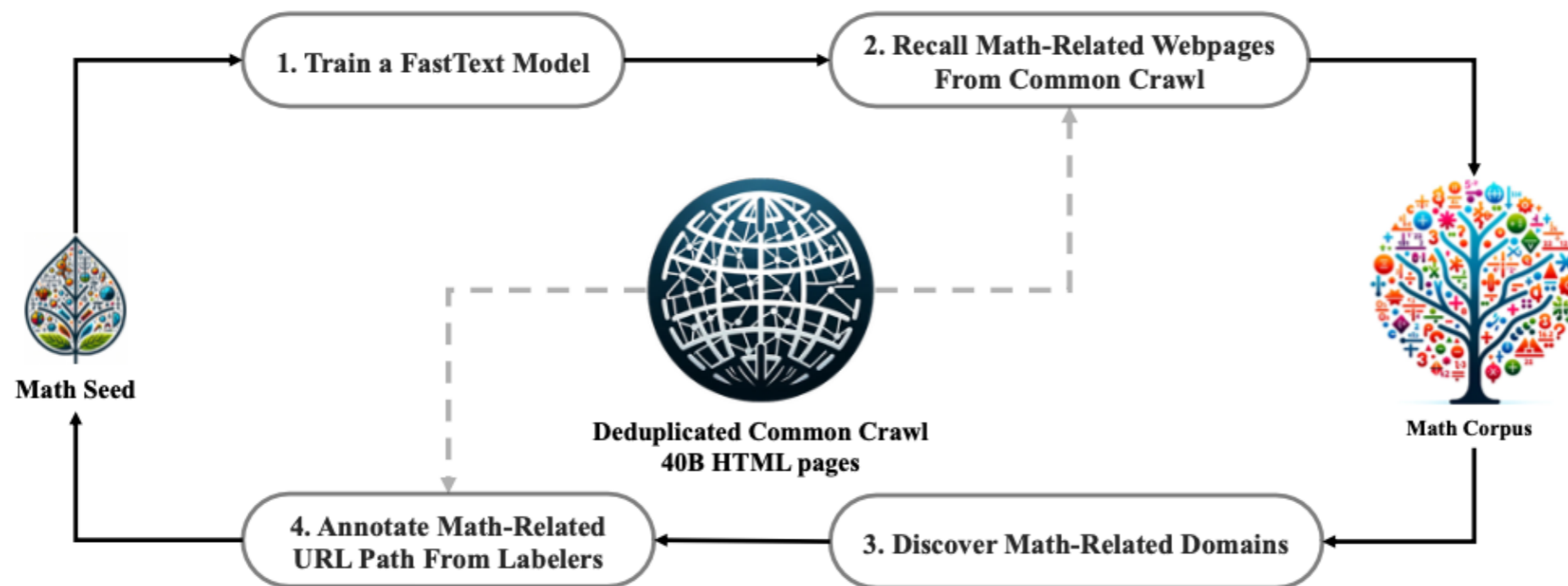


Figure 2 | An iterative pipeline that collects mathematical web pages from Common Crawl.

OpenWebMath initial seed corpus

500,000 data points from the seed corpus as positive training examples

500,000 web pages from Common Crawl as negative ones.

Common Crawl

2 billion web pages in each of its monthly crawls,
generating around 250 terabytes of data each month

Stored on Amazon Web Services' Public Data Sets

Access to the corpus hosted by Amazon is free.

<https://commoncrawl.org/overview>

DeepSeek reduced the Common Crawl to 40B webpages

URL-based deduplication and near-deduplication techniques,

FastText Model

Text classification model

To train the model, they used
vector dimension to 256,
learning rate to 0.1,
maximum length,
of word n-gram to 3,
the minimum number of word occurrences to 3
the number of training epochs to 3

Rank the collected pages according to fastText model
Only preserve the top-ranking ones

Collecting the Corpus

First iteration of data collection missed Math webpages

Enriched the seed corpus

Calculate the percentage of Math pages in a domain

If 10% of the pages are Math pages, manually annotate the URLs

Web pages linked to these URLs, yet uncollected, will be added to the seed corpus

Retrain model and repeat

After 4 iterations 98% of data was collected in third iteration so stopped

35.5M mathematical web pages, totaling 120B tokens

Benchmark Contamination

Need to remove answers to Math benchmarks questions from training data

GSM8K

MATH (12,500 problems)

Chinese benchmarks such as

CMATH

AGIEval

Filtering criteria

Any text segment containing a 10-gram string that matches exactly with any sub-string from the evaluation benchmarks is removed from our math training corpus.

For benchmark texts that are shorter than 10 grams but have at least 3 grams employ exact matching to filter out contaminated web pages

Validating the Quality of the DeepSeekMath Corpus

MathPile(8.9B tokens)

Textbooks, Wikipedia, ProofWiki, CommonCrawl, StackExchange, and arXiv

OpenWebMath

CommonCrawl, 13.6B tokens

Proof-Pile-2

OpenWeb-Math, AlgebraicStack (10.3B tokens), and arXiv papers (28.0B tokens).

Train DeepSeek-LLM 1.3B separately on each of the Corpusora

Validating the Quality of the DeepSeekMath Corpus

Train DeepSeek-LLM 1.3B separately on each of the Corpusora

AdamW optimizer with

$\beta_1 = 0.9$, $\beta_2 = 0.95$, and $\text{weight_decay} = 0.1$

Multi-step learning rate schedule

Learning rate reaches the peak after 2,000 warmup steps,

Decreases to its 31.6% after 80% of the training process,

Further decreases to 10.0% of the peak after 90% of the training process.

Maximum learning rate $5.3e-4$,

Batch size of 4M tokens

4K context length

Validating the Quality of the DeepSeekMath Corpus

Math Corpus	Size	English Benchmarks					Chinese Benchmarks		
		GSM8K	MATH	OCW	SAT	MMLU STEM	CMATH	Gaokao MathCloze	Gaokao MathQA
No Math Training	N/A	2.9%	3.0%	2.9%	15.6%	19.5%	12.3%	0.8%	17.9%
MathPile	8.9B	2.7%	3.3%	2.2%	12.5%	15.7%	1.2%	0.0%	2.8%
OpenWebMath	13.6B	11.5%	8.9%	3.7%	31.3%	29.6%	16.8%	0.0%	14.2%
Proof-Pile-2	51.9B	14.3%	11.2%	3.7%	43.8%	29.2%	19.9%	5.1%	11.7%
DeepSeekMath Corpus	120.2B	23.8%	13.6%	4.8%	56.3%	33.1%	41.5%	5.9%	23.6%

Table 1 | Performance of DeepSeek-LLM 1.3B trained on different mathematical corpora, evaluated using few-shot chain-of-thought prompting. Corpus sizes are calculated using our tokenizer with a vocabulary size of 100K.

DeepSeekMath-Base 7B

Start with DeepSeek-Coder-Base-v1.5 7B

Trained for 500B tokens

56% from DeepSeekMath Corpus,

4% from AlgebraicStack,

10% from arXiv,

20% is Github code,

10% is natural language data from Common Crawl in both English and Chinese.

AdamW optimizer with

$\beta_1 = 0.9$, $\beta_2 = 0.95$, and $\text{weight_decay} = 0.1$

Multi-step learning rate schedule

Learning rate reaches the peak after 2,000 warmup steps,

Decreases to its 31.6% after 80% of the training process,

Further decreases to 10.0% of the peak after 90% of the training process.

Maximum learning rate $4.3e-4$,

Batch size of 10M tokens

4K context length

DeepSeekMath-Instruct 7B

Start with DeepSeekMath-Base

Training examples are randomly concatenated to 4K tokens.

500 steps

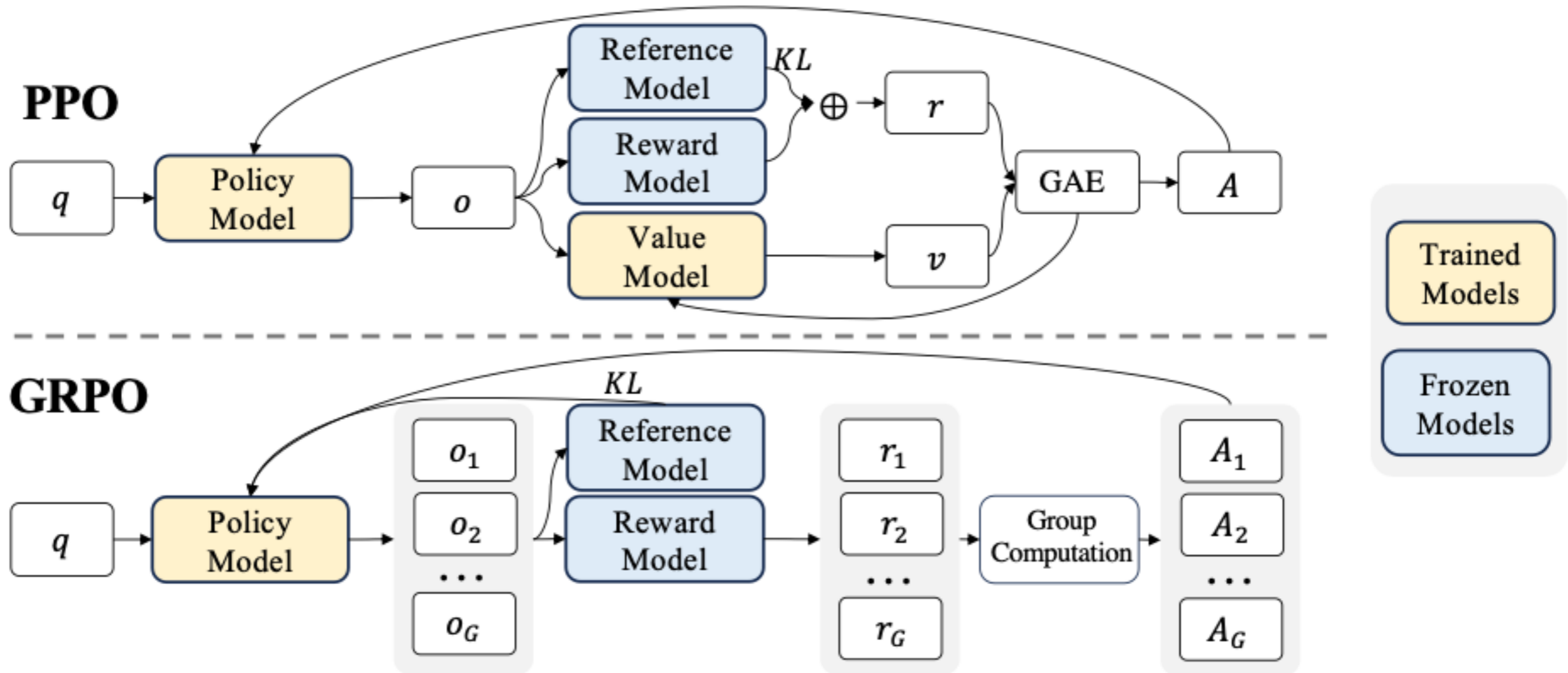
Batch size of 256

Constant learning rate of $5e-5$

DeepSeek-Coder-Base-v1.5 7B -> DeepSeekMath-Base -> DeepSeekMath-Instruct 7B

Now Use Reinforcement Learning

Group Relative Policy Optimization (GRPO)



GRPO Basics

Sampling:

Generate multiple outputs for each prompt using the current policy

Reward Scoring:

Each generation is scored using a reward function, could be rule-based or outcome-based

Advantage Calculation:

The average reward of the generated outputs is used as a baseline. The advantage of each solution within the group is then computed relative to this baseline. The reward is normalized within a group.

Policy Optimization:

The policy tries to maximize the GRPO objective, which includes the calculated advantages and a KL divergence term.

KL (Kullback-Leibler) divergence

Measure of how one probability distribution differs from another

In PPO quantifies the difference between
Current policy (the policy being updated) and
Old policy

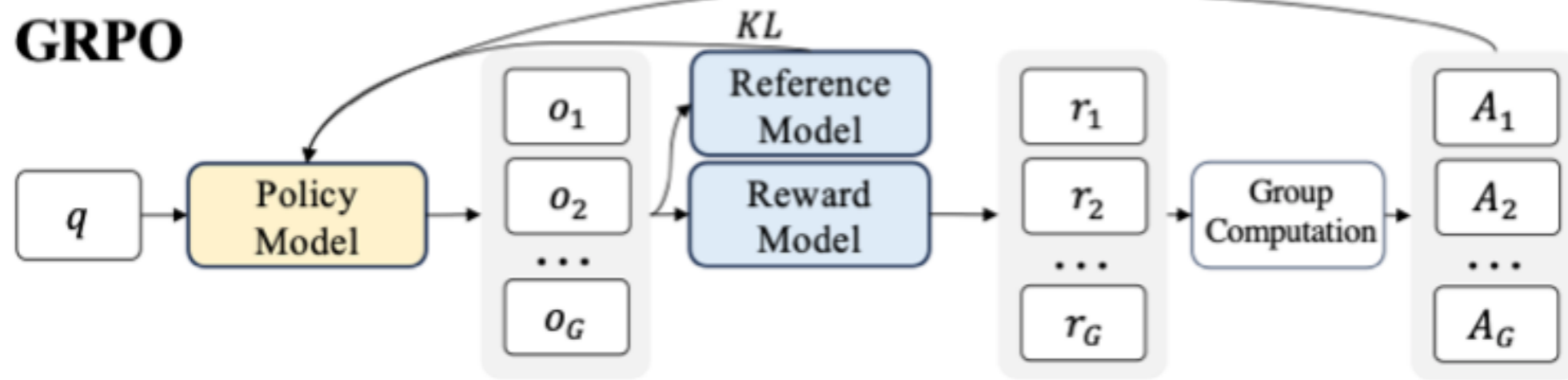
Policy Stability:

PPO update policies gradually
The KL divergence helps gauge this difference

Trust Region Control:

By measuring how far the new policy has moved from the old one, PPO can enforce a “trust region” that prevents overly large updates, ensuring stable training.

GRPO



Algorithm 1 Iterative Group Relative Policy Optimization

Input initial policy model $\pi_{\theta_{\text{init}}}$; reward models r_{φ} ; task prompts \mathcal{D} ; hyperparameters ε, β, μ

- 1: policy model $\pi_{\theta} \leftarrow \pi_{\theta_{\text{init}}}$
- 2: **for** iteration = 1, ..., I **do**
- 3: reference model $\pi_{\text{ref}} \leftarrow \pi_{\theta}$
- 4: **for** step = 1, ..., M **do**
- 5: Sample a batch \mathcal{D}_b from \mathcal{D}
- 6: Update the old policy model $\pi_{\theta_{\text{old}}} \leftarrow \pi_{\theta}$
- 7: Sample G outputs $\{o_i\}_{i=1}^G \sim \pi_{\theta_{\text{old}}}(\cdot | q)$ for each question $q \in \mathcal{D}_b$
- 8: Compute rewards $\{r_i\}_{i=1}^G$ for each sampled output o_i by running r_{φ}
- 9: Compute $\hat{A}_{i,t}$ for the t -th token of o_i through group relative advantage estimation.
- 10: **for** GRPO iteration = 1, ..., μ **do**
- 11: Update the policy model π_{θ} by maximizing the GRPO objective (Equation 21)
- 12: Update r_{φ} through continuous training using a replay mechanism.

Output π_{θ}

DeepSeekMath-RL

DeepSeek-Coder-Base-v1.5 7B -> DeepSeekMath-Base

-> DeepSeekMath-Instruct 7B

-> DeepSeekMath-RL

GRPO and the Countdown Game

<https://www.philschmid.de/mini-deepseek-r1>

<https://github.com/philschmid/deep-learning-pytorch-huggingface/blob/main/training/mini-deepseek-r1-aha-grpo.ipynb>

Countdown game

Use set numbers and basic arithmetic operations (+, -, ×, ÷) to reach or get as close as possible to a target number.

Target Number: 952

Available Numbers: 25, 50, 75, 100, 3, 6

$$(100 \times (3 \times 3)) + (50 + 6 / 3) = 952$$

Training Dataset

```
from transformers import AutoTokenizer  
from datasets import load_dataset
```

```
dataset_id = "Jiayi-Pan/Countdown-Tasks-3to4"  
dataset = load_dataset(dataset_id, split="train")  
dataset = dataset.shuffle(seed=42).select(range(50000))
```

```
{'target': 88, 'nums': [95, 21, 3]}  
{'target': 62, 'nums': [22, 29, 69]}  
{'target': 17, 'nums': [9, 73, 38, 2]}  
{'target': 98, 'nums': [33, 4, 2, 59]}  
{'target': 53, 'nums': [25, 42, 13, 83]}  
{'target': 19, 'nums': [15, 23, 57]}  
{'target': 47, 'nums': [2, 37, 5]}  
{'target': 71, 'nums': [74, 39, 66, 81]}  
{'target': 26, 'nums': [98, 58, 47, 33]}  
{'target': 83, 'nums': [67, 65, 87, 38]}
```



```
tokenizer = AutoTokenizer.from_pretrained("Qwen/Qwen2.5-3B-Instruct")
```

```
def generate_r1_prompt(numbers, target):
```

```
    r1_prefix = [{
```

```
        "role": "system",
```

```
        "content": "You are a helpful assistant. You first thinks about the reasoning process in the mind and then provides the user with the answer."
```

```
    },
```

```
    {
```

```
        "role": "user",
```

```
        "content": f"Using the numbers {numbers}, create an equation that equals {target}. You can use basic arithmetic operations (+, -, *, /) and each number can only be used once. Show your work in <think> </think> tags. And return the final equation and answer in <answer> </answer> tags, for example <answer> (1 + 2) / 3 = 1 </answer>."
```

```
    },
```

```
    {
```

```
        "role": "assistant",
```

```
        "content": "Let me solve this step by step.\n<think>"
```

```
    ]]
```

```
    return {"prompt": tokenizer.apply_chat_template(r1_prefix, tokenize=False, continue_final_message=True), "target": target}
```

```
# convert our dataset to the r1 prompt
dataset = dataset.map(lambda x: generate_r1_prompt(x["nums"], x["target"]))

# split the dataset into train and test
train_test_split = dataset.train_test_split(test_size=0.1)

train_dataset = train_test_split["train"]
test_dataset = train_test_split["test"]
```

Reward Functions

GRPOTrainer needs reward function (s) to evaluate model

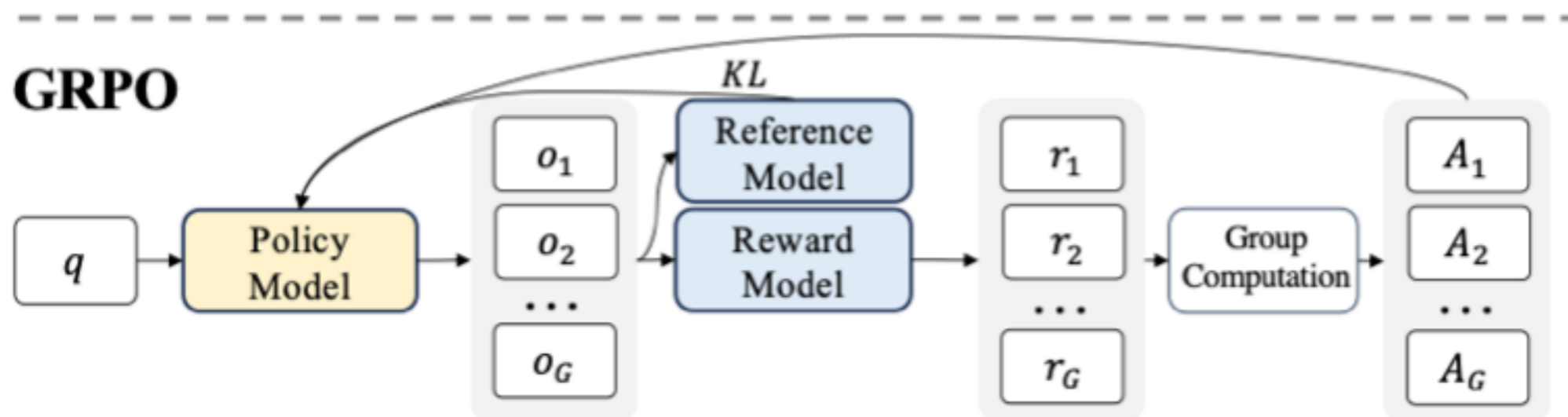
Use

- Properly formatted response

- Correct equation

 - Numbers used only once

 - Math is done correctly



Train the model using GRPO

```
import re
```

```
def format_reward_func(completions, target, **kwargs):
```

```
    """Format: <think>...</think><answer>...</answer>"""
```

```
    rewards = []
```

```
    for completion, gt in zip(completions, target):
```

```
        try:
```

```
            completion = "<think>" + completion
```

```
            # Check if the format is correct
```

```
            regex = r"^<think>([^\<]*(?:<(?!/?think>)[^\<]*)*)</think>\n<answer>([\s\S]*?)</answer>$"
```

```
            match = re.search(regex, completion, re.DOTALL)
```

```
            # if the format is not correct, reward is 0
```

```
            if match is None or len(match.groups()) != 2:
```

```
                rewards.append(0.0)
```

```
            else:
```

```
                rewards.append(1.0)
```

```
        except Exception:
```

```
            rewards.append(0.0)
```

```
    return rewards
```

```

def equation_reward_func(completions, target, nums, **kwargs):
    rewards = []
    for completion, gt, numbers in zip(completions, target, nums):
        try:
            completion = "<think>" + completion
            match = re.search(r"<answer>(.*?)</answer>", completion) # Is format correct
            if match is None:
                rewards.append(0.0)
                continue
            equation = match.group(1).strip()
            used_numbers = [int(n) for n in re.findall(r'\d+', equation)]

            if sorted(used_numbers) != sorted(numbers): # Check if all numbers are used exactly once
                rewards.append(0.0)
                continue
            allowed_pattern = r'^[\d+\-*/().\s]+$'
            if not re.match(allowed_pattern, equation):
                rewards.append(0.0)
                continue

            result = eval(equation, {"__builtins__": None}, {}) # Evaluate the equation
            if abs(float(result) - float(gt)) < 1e-5: # Is equation correct
                rewards.append(1.0)
            else:
                rewards.append(0.0)
        except Exception:
            rewards.append(0.0)
    return rewards

```

```
correct_sample_1 = """We need to find an equation using the numbers 19, 36, 55, and 7 exactly once, with basic arithmetic operations, that equals 65. One possible combination is 55 + 36 - 19 + 7... </think>  
<answer> 55 + 36 - 7 - 19 </answer>"""
```

```
correct_sample_2 = """ ... </think>  
<answer> 55 + 36 - 7 - 19 </answer>"""
```

```
wrong_format = """User: Using the numbers [19, 36, 55, 7], create an equation that equals 65."""
```

```
wrong_format_2 = """To find the equation that equals 79 using the numbers 95, 78, 6, 88, I'll start by adding 95 and 95:
```

```
95 + 88 = 183
```

```
Now, let's subtract 104 from 183 to get 79:
```

```
183 - 104 = 79
```

```
<think> 183 - 104 = 79 </think><think> 183 - 104 = 79 </think><answer> 183 - 104 = 79 </answer>"""
```

```
wrong_result = """ ... </think>
```

```
<answer> 55 + 36 - 7 - 18 </answer>"""
```

```
test_rewards = format_reward_func(completions=[correct_sample_1, correct_sample_2, wrong_format, wrong_format_2, wrong_result], target=["65", "65", "65", "65", "65"], nums=[[19, 36, 55, 7]] * 5)  
assert test_rewards == [1.0, 1.0, 0.0, 0.0, 1.0], "Reward function is not working"
```

```
test_rewards = format_reward_func(
    completions=[correct_sample_1, correct_sample_2, wrong_format, wrong_format_2, wrong_result],
    target=["65", "65", "65", "65", "65"],
    nums=[[19, 36, 55, 7]] * 5)
```

```
assert test_rewards == [1.0, 1.0, 0.0, 0.0, 1.0], "Reward function is not working"
```

```
test_rewards = equation_reward_func(
    completions=[correct_sample_1, correct_sample_2, wrong_format, wrong_format_2, wrong_result],
    target=["65", "65", "65", "65", "65"],
    nums=[[19, 36, 55, 7]] * 5)
```

```
assert test_rewards == [1.0, 1.0, 0.0, 0.0, 0.0], "Reward function is not working"
```

```
from trl import GRPOConfig, GRPOTrainer, get_peft_config, ModelConfig

# our model we are going to use as policy
model_config = ModelConfig(
    model_name_or_path="Qwen/Qwen2.5-3B-Instruct",
    torch_dtype="bfloat16",
    attn_implementation="flash_attention_2",
    use_peft=True,
    load_in_4bit=True,
)
```



```
# Hyperparameters
training_args = GRPOConfig(
    output_dir="qwen-r1-aha-moment",
    learning_rate=5e-7,
    lr_scheduler_type="cosine",
    logging_steps=10,
    max_steps=100,
    per_device_train_batch_size=1,
    gradient_accumulation_steps=1,
    gradient_checkpointing=True,
    gradient_checkpointing_kwargs={"use_reentrant": False},
    bf16=True,
    # GRPO specific parameters
    max_prompt_length=256,
    max_completion_length=1024, # max length of the generated output for our solution
    num_generations=2,
    beta=0.001,
)
```

```
trainer = GRPOTrainer(  
    model=model_config.model_name_or_path,  
    reward_funcs=[format_reward_func, equation_reward_func],  
    args=training_args,  
    train_dataset=train_dataset,  
    eval_dataset=test_dataset,  
    peft_config=get_peft_config(model_config),  
)
```

```
# Train and push the model to the Hub  
trainer.train()  
# Save model  
trainer.save_model(training_args.output_dir)
```

Run Time

On single CPU, GPU 20 minutes per step, but need 450 steps

Using

vLLM

DeepSpeed

4 H100 80GBs

Took 6 hours

Hyperparameters

Started with DeepSeekMath paper values

learning rate of $1e-6$

beta (KL coefficient) of 0.04

Unstable training runs after around 150 steps

So used learning rate to $5e-7$ and the beta to 0.001

torch_dtype: bfloat16
attn_implementation: flash_attention_2
bf16: true
tf32: true
output_dir: runs/qwen-2.5-3b-r1-countdown

Full Parameter List

Dataset arguments

dataset_id_or_path: Jiayi-Pan/Countdown-Tasks-3to4

Lora Arguments

No LoRA is used here

Training arguments

max_steps: 450

per_device_train_batch_size: 1

gradient_accumulation_steps: 8

gradient_checkpointing: true

gradient_checkpointing_kwargs:

 use_reentrant: false

learning_rate: 5.0e-7 # 1.0e-6 as in the deepseek math paper 5-e7 from <https://hijkzzz.notion.site/unraveling-rlhf-and-its-variants-engineering-insights#147d9a33ecc9806090f3d5c749d31f05>

lr_scheduler_type: cosine

warmup_ratio: 0.03

GRPO specific parameters

beta: 0.001 # 0.04 as in the deepseek math paper 0.001 from <https://hijkzzz.notion.site/unraveling-rlhf-and-its-variants-engineering-insights#147d9a33ecc9806090f3d5c749d31f05>

max_prompt_length: 256

max_completion_length: 1024

num_generations: 8

use_vllm: true

vllm_device: "cuda:3"

vllm_gpu_memory_utilization: 0.5

Logging arguments

logging_strategy: steps

logging_steps: 2

report_to:

- tensorboard

save_strategy: "steps"

save_steps: 25

seed: 42

Training Observations

~50 steps

the model has learned the correct format `<think>...</think>\n<answer>...</answer>`.

100 steps

Success rate for solving the equation is around 25%. The model starts to "reason" with words

200 steps the

Performance seems to converge much slower
~40% success rate.

Learns a new "format" where it solves the equation by trying different combinations and reviewing the results,

450 steps

50% success

The performance still improves slowly

Model kept its new format form from step 200.

Successful Reasoning Sample up to step 200

<think> First, I need to think about the numbers we have: 14, 70, and 80. To get 85, I think 80 might be a good starting point because it's right in between 85. If I subtract or add 80, it won't lead to 85 easily. Next, 70 seems to be a good intermediate number, especially to use in conjunction with 14.

Now, I can think about different combinations using addition, subtraction and multiplication/division. If I take $80 - 70 = 10$, which is not needed. So, I need to find a way to use 14. If I subtract 14 from 80, I get 66, which still isn't 85. If I subtract 70 from 80, I get 10 again, this isn't constructive yet.

Multiplication could create numbers larger than the target 85. Let's consider division: If I divide 70 by 14, I get 5. Then if I add this result to 80, I get 85. This combination seems promising: $80 - (70 / 14) = 80 - 5 = 75$, this still isn't quite right. Let me correct my approach: Let's use division and then addition. If I take $80 / (70 / 14) \rightarrow 80 / 5 = 16$, and then $80 - 16 = 64$, still incorrect. Let me simplify: $80 / (70 / 14) \rightarrow 80 / 5 = 16$, and then $80 - 16$ gives 64 again, let me correct: $80 - (70 / 14) \rightarrow 80 - 5 = 75$, but I need 85. Thus I will use the combinations in a slightly different order to insert numbers: I get $80 + (70 / 14) = 80 + 5 = 85$ </think>

<answer> $80 + (70 / 14)$ </answer>

Successful Reasoning Sample between step 200 and 450

<think> We need to use the numbers 39, 16, and 55 with basic arithmetic operations to get 78. Let's try different combinations:

- $55 + 39 - 16 = 84 - 16 = 68$ (too low)

- $55 + 16 - 39 = 71 - 39 = 32$ (too low)

- $55 - 16 + 39 = 39 + 39 = 78$ (this works)

So, the equation is $55 - 16 + 39 = 78$. </think>

<answer> $55 - 16 + 39$ </answer>