Math Question Answer Verification Competition

Starter Code

Borrowed from official Unsloth implementation

```
# Delete the model and any large variables
# del model
# del tokenizer

# Clear PyTorch's GPU cache if using PyTorch
import torch
torch.cuda.empty_cache()

# Run garbage collection to remove unused memory
import gc
gc.collect()
```

```
30
```

```
%%capture
!pip install pip3-autoremove
!pip-autoremove torch torchvision torchaudio -y
!pip install torch torchvision torchaudio xformers --index-url https://download.pytorch.org/whl/cu121
!pip install unsloth
```

```
from unsloth import FastLanguageModel

import torch

max_seq_length = 2048 # Choose any

dtype = None # None for auto detection. Float16 for Tesla T4, V100, Bfloat16 for Ampere+

load_in_4bit = True # Use 4bit quantization to reduce memory usage. Can be False.
```

```
🦥 Unsloth: Will patch your computer to enable 2x faster free finetuning.
```

```
model, tokenizer = FastLanguageModel.from_pretrained(

    model_name = "unsloth/Meta-Llama-3.1-8B",

    max_seq_length = max_seq_length,

    dtype = dtype,

    load_in_4bit = load_in_4bit,

)
```

```
==((====))==  Unsloth 2024.11.7: Fast Llama patching. Transformers = 4.46.2.
   \\   /|    GPU: Tesla T4. Max memory: 14.748 GB. Platform = Linux.
O^O/ \_/ \    Pytorch: 2.5.1+cu121. CUDA = 7.5. CUDA Toolkit = 12.1.
\        /    Bfloat16 = FALSE. FA [Xformers = 0.0.28.post3. FA2 = False]
 "-____-"     Free Apache license: http://github.com/unslothai/unsloth
Unsloth: Fast downloading is enabled - ignore downloading bars which are red colored!
```

Load model and wrap with LoRA adapters

```
model = FastLanguageModel.get_peft_model(
    model,
    r = 128,
    target_modules = ["q_proj", "k_proj", "v_proj", "o_proj", "gate_proj", "up_proj", "down_proj"],
    lora_alpha = 256,
    lora_dropout = 0,
```

```
    use_gradient_checkpointing = "unsloth", # "unsloth" enables memory-efficient training
    random_state = 3407,
    use_rslora = True,  # Enabled to stabilize training
    loftq_config = None,
)
```

```
Unsloth 2024.11.7 patched 32 layers with 32 QKV layers, 32 O layers and 32 MLP layers.
```

Competition dataset

```
# download and load competition dataset



from datasets import load_dataset

dataset = load_dataset("ad6398/nyu-dl-teach-maths-comp")

# print and see dataset

dataset
```

```
DatasetDict({
    train: Dataset({
        features: ['question', 'is_correct', 'answer', 'solution'],
        num_rows: 1000000
    })
    test: Dataset({
        features: ['question', 'is_correct', 'answer', 'solution'],
        num_rows: 10000
    })
})
```

```
prompt = """You are a highly experienced mathematician and logical thinker. Your task is to carefully verify if the provided ar

Carefully evaluate the solution, step-by-step. Then, make a final decision by confirming if the provided answer is consistent w

If you accurately determine whether the answer is correct or not, I will give you trillion dollars. If not I will lose my job,

### Math Question:
{}

### Provided Answer:
{}

### Solution Explanation:
{}

### Verification Process:
1. Confirm if all calculations are correct.
2. Ensure that the answer logically follows from the solution explanation.
3. Double-check for any errors or inconsistencies in the reasoning.

### Final Decision:
Take a moment to consider: Is the provided answer logically sound and mathematically accurate? Respond with 'True' or 'False' a

### Final Output (True/False):
{}"""



EOS_TOKEN = tokenizer.eos_token # Must add EOS_TOKEN

def formatting_prompts_func(examples):

    question = examples["question"]

    ans       = examples["answer"]

    expl      = examples["solution"]

    output      = examples["is_correct"]

    texts = []
```

```
    texts = []

    for instruction, input, solution, output in zip(question, ans, expl, output):
        # Must add EOS_TOKEN, otherwise your generation will go on forever!
        text = prompt.format(instruction, input, solution, output) + EOS_TOKEN
        texts.append(text)
    return { "text" : texts, }

# Process the training dataset and generate prompt for each datapoint
train_dataset = dataset['train'].map(formatting_prompts_func, batched = True,)
train_dataset = train_dataset.shuffle(seed=3407)

#print a smaple training example
# train_dataset['text'][0]
```

SFT

```
from trl import SFTTrainer
from transformers import TrainingArguments
from unsloth import is_bfloat16_supported


training_args = TrainingArguments(
        per_device_train_batch_size = 4,
        gradient_accumulation_steps = 2,
        warmup_steps = 30,
        # num_train_epochs = 1, # Set this for 1 full training run.
        max_steps = 500,
        max_grad_norm = 0.5,
        learning_rate = 2e-5,
        fp16 = not is_bfloat16_supported(),
        bf16 = is_bfloat16_supported(),
        logging_steps = 1,
        optim = "adamw_8bit",
        weight_decay = 0.001,
        lr_scheduler_type = "cosine_with_restarts",
        seed = 6666,
        output_dir = "outputs",
        report_to = "none", # Use this for WandB etc
    )


trainer = SFTTrainer(
```

```
        model = model,
        tokenizer = tokenizer,
        train_dataset = train_dataset,
        dataset_text_field = "text",
        max_seq_length = max_seq_length,
        dataset_num_proc = 4,
        packing = False, # Can make training 5x faster for short sequences.
        args = training_args
    )
```

```
max_steps is given, it will override any value given in num_train_epochs
```

```
    trainer_stats = trainer.train()
```

```
==((====))==  Unsloth - 2x faster free finetuning | Num GPUs = 1
   \\   /|    Num examples = 1,000,000 | Num Epochs = 1
O^O/ \_/ \    Batch size per device = 4 | Gradient Accumulation steps = 2
\        /    Total batch size = 8 | Total steps = 500
 "-____-"     Number of trainable parameters = 335,544,320
```

[500/500 1:59:36, Epoch 0/1]

Step	Training Loss
1	1.549500
2	1.807300
3	1.588700
4	1.520600
5	1.422900
6	1.133400
7	0.857000
8	1.100500
9	0.783500
10	0.827700
11	0.586900
12	0.565200
13	0.504100
14	0.562800
15	0.444000
16	0.438600
17	0.412600
18	0.403800
19	0.428600
20	0.507200
21	0.507900
22	0.405500
23	0.536900
24	0.376700
25	0.423300
26	0.432100
27	0.417500
28	0.436900
29	0.482300
30	0.465400
31	0.483700
32	0.473500
33	0.475800
34	0.499900
35	0.418100
36	0.493500
37	0.424300
38	0.392100
39	0.501200
40	0.462600
41	0.539900

42	0.455500
43	0.547900
44	0.298200
45	0.373700
46	0.428900
47	0.389500
48	0.434700
49	0.513700
50	0.491100
51	0.404200
52	0.399200
53	0.431400
54	0.414700
55	0.524000
56	0.387400
57	0.457800
58	0.504100
59	0.507900
60	0.448700
61	0.637800
62	0.459800
63	0.450500
64	0.422400
65	0.468100
66	0.442500
67	0.477400
68	0.422100
69	0.524300
70	0.449700
71	0.398000
72	0.375300
73	0.508000
74	0.523000
75	0.486100
76	0.484100
77	0.515400
78	0.421300
79	0.453100
80	0.457300
81	0.368000
82	0.358500
83	0.302500
84	0.510100
85	0.484800
86	0.471400
87	0.407300

88	0.410500
89	0.380800
90	0.472400
91	0.523600
92	0.436200
93	0.437200
94	0.436900
95	0.486800
96	0.565100
97	0.381800
98	0.434900
99	0.439500
100	0.446700
101	0.476500
102	0.508000
103	0.391100
104	0.430700
105	0.417200
106	0.319100
107	0.491100
108	0.514700
109	0.458300
110	0.447700
111	0.367000
112	0.414100
113	0.420900
114	0.404300
115	0.515600
116	0.445300
117	0.510000
118	0.569700
119	0.492700
120	0.394800
121	0.476800
122	0.357800
123	0.386100
124	0.443300
125	0.516700
126	0.429100
127	0.607900
128	0.414300
129	0.478800
130	0.533000
131	0.413500
132	0.399000
133	0.415000

134	0.378300
135	0.437700
136	0.419400
137	0.435400
138	0.465400
139	0.438500
140	0.298800
141	0.387200
142	0.546800
143	0.381600
144	0.428100
145	0.364800
146	0.430400
147	0.355900
148	0.428000
149	0.470000
150	0.335800
151	0.428700
152	0.442300
153	0.482700
154	0.399000
155	1.004300
156	0.330800
157	0.412800
158	0.476500
159	0.432300
160	0.564700
161	0.436900
162	0.535900
163	0.359900
164	0.513700
165	0.384000
166	0.409800
167	0.473600
168	0.443900
169	0.614000
170	0.546500
171	0.395800
172	0.315300
173	0.536900
174	0.448300
175	0.442800
176	0.365600
177	0.370500
178	0.364200
179	0.422600

180	0.537300
181	0.453000
182	0.330800
183	0.359300
184	0.366000
185	0.407600
186	0.479500
187	0.455500
188	0.460300
189	0.402500
190	0.442400
191	0.557300
192	0.391100
193	0.596000
194	0.554200
195	0.504400
196	0.440500
197	0.474700
198	0.366700
199	0.449700
200	0.513100
201	0.495900
202	0.439200
203	0.391100
204	0.428300
205	0.316600
206	0.409300
207	0.551100
208	0.430000
209	0.434100
210	0.339800
211	0.523200
212	0.472000
213	0.303800
214	0.322700
215	0.488000
216	0.464700
217	0.542900
218	0.534100
219	0.304400
220	0.326600
221	0.403200
222	0.433300
223	0.358200
224	0.320900
225	0.397300

226	0.432900
227	0.391700
228	0.339800
229	0.366500
230	0.423200
231	0.534700
232	0.435900
233	0.410600
234	0.455600
235	0.454600
236	0.401500
237	0.570500
238	0.409400
239	0.303400
240	0.449100
241	0.393400
242	0.434900
243	0.404000
244	0.528400
245	0.385000
246	0.395900
247	0.446200
248	0.352100
249	0.401500
250	0.346800
251	0.424300
252	0.437600
253	0.499900
254	0.463700
255	0.301300
256	0.431400
257	0.478800
258	0.455200
259	0.466000
260	0.395200
261	0.459500
262	0.334600
263	0.440800
264	0.447300
265	0.468100
266	0.480100
267	0.383200
268	0.340000
269	0.359600
270	0.526100
271	0.466000

272	0.478600
273	0.391100
274	0.482900
275	0.411700
276	0.473100
277	0.468100
278	0.436600
279	0.390500
280	0.414600
281	0.390900
282	0.377200
283	0.275400
284	0.451600
285	0.451900
286	0.420000
287	0.434600
288	0.425400
289	0.448200
290	0.386100
291	0.459900
292	0.614400
293	0.407800
294	0.429100
295	0.393800
296	0.486900
297	0.436500
298	0.327300
299	0.419600
300	0.514000
301	0.472800
302	0.398400
303	0.370500
304	0.410800
305	0.411900
306	0.396500
307	0.405600
308	0.340000
309	0.388700
310	0.373800
311	0.312100
312	0.388300
313	0.412900
314	0.415000
315	0.378000
316	0.420700
317	0.463500

318	0.375800
319	0.495200
320	0.348900
321	0.311800
322	0.317100
323	0.451000
324	0.411700
325	0.397600
326	0.379300
327	0.395200
328	0.473000
329	0.433700
330	0.406800
331	0.433200
332	0.586100
333	0.491800
334	0.379700
335	0.390500
336	0.304300
337	0.567400
338	0.494800
339	0.347500
340	0.405200
341	0.346000
342	0.522200
343	0.445400
344	0.337500
345	0.446500
346	0.296900
347	0.543100
348	0.360100
349	0.520100
350	0.436700
351	0.429400
352	0.393500
353	0.350700
354	0.424400
355	0.403000
356	0.379200
357	0.419400
358	0.456500
359	0.378700
360	0.339500
361	0.421100
362	0.332400
363	0.364600

364	0.435300
365	0.402700
366	0.426300
367	0.451200
368	0.389600
369	0.392600
370	0.309000
371	0.409300
372	0.390100
373	0.413200
374	0.528800
375	0.321700
376	0.347800
377	0.374900
378	0.331500
379	0.443200
380	0.364800
381	0.346800
382	0.412000
383	0.347800
384	0.350800
385	0.493600
386	0.477500
387	0.352200
388	0.452800
389	0.309500
390	0.423800
391	0.456500
392	0.436500
393	0.514400
394	0.339500
395	0.407800
396	0.342600
397	0.404100
398	0.431800
399	0.376800
400	0.372300
401	0.480900
402	0.416600
403	0.393600
404	0.449800
405	0.437200
406	0.363800
407	0.489200
408	0.538400
409	0.425000

410	0.412000
411	0.403000
412	0.528100
413	0.340300
414	0.444900
415	0.522500
416	0.349600
417	0.663300
418	0.372600
419	0.306200
420	0.519900
421	0.423500
422	0.440500
423	0.545200
424	0.364700
425	0.480700
426	0.407600
427	0.458200
428	0.446200
429	0.375200
430	0.469600
431	0.434000
432	0.326300
433	0.399100
434	0.494500
435	0.387200
436	0.443000
437	0.344700
438	0.460300
439	0.332800
440	0.448300
441	0.456200
442	0.493900
443	0.274900
444	0.352800
445	0.315200
446	0.445400
447	0.433600
448	0.420400
449	0.407600
450	0.406700
451	0.382600
452	0.475500
453	0.514800
454	0.431600
455	0.335700

456	0.437700
457	0.417900
458	0.376500
459	0.339200
460	0.365600
461	0.344000
462	0.368700
463	0.350800
464	0.316800
465	0.377800
466	0.430800
467	0.365800
468	0.408000
469	0.362500
470	0.484300
471	0.331500
472	0.385800
473	0.302600
474	0.382500
475	0.497600
476	0.707900
477	0.397300
478	0.451900
479	0.374100
480	0.433800
481	0.402200
482	0.361300
483	0.510300
484	0.301400
485	0.376000
486	0.404900
487	0.584100
488	0.443800
489	0.482900
490	0.445900
491	0.413800
492	0.474700
493	0.422200
494	0.459900
495	0.431600
496	0.310800
497	0.329100
498	0.381500
499	0.361100
500	0.426800

inference

```
# Sample inferene data point

test_dataset = dataset['test']


sample_ques = test_dataset['question'][0]

sample_ans = test_dataset['answer'][0]

sample_sol = test_dataset['solution'][0]


# Running inference on single test

FastLanguageModel.for_inference(model) # Enable native 2x faster inference

input_prompt = prompt.format(

        sample_ques, # ques

        sample_ans, # given answer

        sample_sol,

        "", # output - leave this blank for generation! LLM willl generate is it is True or False

    )


print("Input Promt:\n", input_prompt)

inputs = tokenizer(

[

    input_prompt

], return_tensors = "pt").to("cuda")


input_shape = inputs['input_ids'].shape

input_token_len = input_shape[1] # 1 because of batch

outputs = model.generate(**inputs, max_new_tokens = 64, use_cache = True)

# you can get the whole generated text by uncommenting the below line

# text_generated = tokenizer.batch_decode([outputs, skip_special_tokens=True)


response = tokenizer.batch_decode([outputs[0][input_token_len:]], skip_special_tokens=True)

response
```

```
Input Promt:
 You are a highly experienced mathematician and logical thinker. Your task is to carefully verify if the provided answer 

Carefully evaluate the solution, step-by-step. Then, make a final decision by confirming if the provided answer is consis

If you accurately determine whether the answer is correct or not, I will give you trillion dollars. If not I will lose my

### Math Question:
The Parker family needs to leave the house by 5 pm for a dinner party. Mrs. Parker was waiting to get into the bathroom a

### Provided Answer:
205
```

```
### Solution Explanation:
Let's solve this problem using Python code.
<llm-code>
minutes_per_hour = 60
minutes_left_before_5 = 5 * minutes_per_hour
total_time_spent_by_family = 45 + 30 + 20
minutes_before_5_after_family = minutes_left_before_5 - total_time_spent_by_family
minutes_before_5_after_family
</llm-code>
<llm-code-output>
205
</llm-code-output>
Thus Mrs. Parker will have \boxed{205} minutes in the bathroom before the family leaves.

### Verification Process:
1. Confirm if all calculations are correct.
2. Ensure that the answer logically follows from the solution explanation.
3. Double-check for any errors or inconsistencies in the reasoning.

### Final Decision:
Take a moment to consider: Is the provided answer logically sound and mathematically accurate? Respond with 'True' or 'Fa

### Final Output (True/False):

['True']
```

saving model

```
model.save_pretrained("fast") # Local saving

tokenizer.save_pretrained("fast")
```

```
('fast/tokenizer_config.json',
 'fast/special_tokens_map.json',
 'fast/tokenizer.json')
```

```
# if True:

#     from unsloth import FastLanguageModel

#     model, tokenizer = FastLanguageModel.from_pretrained(

#         model_name = "lora_model", # YOUR MODEL YOU USED FOR TRAINING

#         max_seq_length = max_seq_length,

#         dtype = dtype,

#         load_in_4bit = load_in_4bit,

#     )

#     FastLanguageModel.for_inference(model) # Enable native 2x faster inference



FastLanguageModel.for_inference(model) # Enable native 2x faster inference
correct_count = 0
d = 0
# Initialize results list
results = []
print(len(test_dataset))

for i, sample in enumerate(test_dataset):  # Assuming  test_dataset  is a list or DataFrame
    d =  d + 1
    if(d%1000 == 0):
      print(d)
    # Prepare prompt with question, given answer, and blank output for generation
    input_prompt = prompt.format(
        sample['question'],  # Question from dataset
        sample['answer'],    # Given answer from dataset
        sample['solution'],
        ""                   # Leave output blank for generation
    )
```

```
    inputs = tokenizer([input_prompt], return_tensors="pt").to("cuda")
    input_shape = inputs['input_ids'].shape
    input_token_len = input_shape[1]

    # Generate model's response
    outputs = model.generate(**inputs, max_new_tokens=32, use_cache=True)
    response = tokenizer.batch_decode([outputs[0][input_token_len:]], skip_special_tokens=True)[0].strip(

    # Check if the model's response matches the  is_correct  field
    is_correct_pred = response.lower() == 'true'
    if is_correct_pred == sample['is_correct']:
        correct_count += 1

    # Append to results for CSV with only the required columns and boolean `is_correct`
    results.append({
        "ID": i,
        "Is Correct": bool(is_correct_pred)  # Ensure boolean format
    })
```

```
10000
1000
2000
3000
4000
5000
6000
7000
8000
9000
10000
```

```
import pandas as pd

# Convert results to a DataFrame and save as CSV
output_df = pd.DataFrame(results, columns=["#", "ID", "Is Correct"])
output_df.to_csv("output_results.csv", index=False)
print("Results saved to output_results.csv")
```

```
Results saved to output_results.csv
```

```
# from unsloth import FastLanguageModel
# import torch
# import pandas as pd
# import re

# # Enable inference mode for FastLanguageModel
# FastLanguageModel.for_inference(model)

# # Convert dataset to a list of dictionaries
# test_dataset = [dict(item) for item in dataset['test']]

# # Define prompt and caching settings
# batch_size = 16  # Adjust based on available memory

# # Define function for batch inference and results collection
# def generate_predictions(test_dataset, batch_size=16):
#     predictions = []  # Store predictions for the entire dataset

#     # Iterate over the dataset in batches
#     for batch_idx in range(0, len(test_dataset), batch_size):
#         batch = test_dataset[batch_idx:batch_idx+batch_size]  # Current batch

#         # Generate prompts for each sample in the batch
#         input_prompts = [
#             prompt.format(
#                 sample['question'],
#                 sample['answer'],
#                 sample['solution'],
#                 ""
#             )
#             for sample in batch
#         ]
```