SFT Training: MathQA + Grade-School-Math

목표

1. **Scenario 1**: MathQA train에서 1500개 샘플로 학습 → MathQA test로 lm_eval 평가
2. **Scenario 2**: MathQA train 1500개 + grade-school-math 1500개 혼합 학습 → MathQA test로 lm_eval 평가

평가

- lm-evaluation-harness mathqa 태스크 (test split)
- 최종 정확도는 lm_eval로 산출

In [1]:
```
# 의존성 (필요 시)
# !pip install lm-eval==0.4.3 peft datasets transformers torch
```

1. 환경 설정

라이브러리 임포트 및 PyTorch 설정

필요한 라이브러리를 임포트하고 메모리 최적화 및 TF32 연산을 설정합니다.

Attention 구현 방식 설정

Flash Attention 2를 시도하고, 실패하면 SDPA를 사용합니다.

In [2]:
```
import os
os.environ["PYTORCH_CUDA_ALLOC_CONF"] = "expandable_segments:True"  # reduce fragmentation

import re
import random
import torch
import torch.nn.functional as F
from torch.utils.data import Dataset, DataLoader
from datasets import load_dataset
from transformers import (
    AutoModelForCausalLM,
    AutoTokenizer,
    get_linear_schedule_with_warmup,
)
from peft import LoraConfig, get_peft_model, TaskType
from tqdm import tqdm
import lm_eval
from lm_eval.models.huggingface import HFLM

torch.backends.cuda.matmul.allow_tf32 = True
torch.backends.cudnn.allow_tf32 = True
if hasattr(torch, "set_float32_matmul_precision"):
    torch.set_float32_matmul_precision("high")

print(f"CUDA available: {torch.cuda.is_available()}")
```

```
CUDA available: True
```

```
/usr/local/lib/python3.12/dist-packages/torch/backends/__init__.py:46: UserWarning: Please use the new API settings t
o control TF32 behavior, such as torch.backends.cudnn.conv.fp32_precision = 'tf32' or torch.backends.cuda.matmul.fp32
_precision = 'ieee'. Old settings, e.g, torch.backends.cuda.matmul.allow_tf32 = True, torch.backends.cudnn.allow_tf32
= True, allowTF32CuDNN() and allowTF32CuBLAS() will be deprecated after Pytorch 2.9. Please see https://pytorch.org/d
ocs/main/notes/cuda.html#tensorfloat-32-tf32-on-ampere-and-later-devices (Triggered internally at /pytorch/aten/src/A
Ten/Context.cpp:80.)
  self.setter(val)
```

In [3]:
```
def get_attn_implementation():
    try:
        from flash_attn import flash_attn_func
        return "flash_attention_2"
    except ImportError:
        return "sdpa"

ATTN_IMPL = get_attn_implementation()
print(f"Attention: {ATTN_IMPL}")
```

```
Attention: sdpa
```

MathQA 데이터셋 MC 변환 함수

allenai/math_qa 데이터셋의 options 필드를 파싱하여 lm-eval과 동일한 Multiple Choice 포맷으로 변환합니다.

MathQA train 데이터셋 로드 및 변환

allenai/math_qa train 데이터를 로드하고 MC 포맷으로 변환합니다.

2. 데이터 로드 및 MC 포맷 변환

Grade-school-math MC 변환 함수들

RESPONSE에서 최종 답 추출, 오답 옵션 생성, MC 데이터셋 구축 함수들을 정의합니다.

Grade-school-math 데이터셋 로드 및 MC 변환

GSM 데이터를 로드하고 MC 포맷으로 변환합니다.

2.1 MathQA train 파싱 (lm-eval doc_to_choice 방식)

Dataset 클래스 및 Collate 함수

MC 데이터를 위한 PyTorch Dataset과 배치 collate 함수를 정의합니다.

Log-likelihood 및 Loss 계산 함수

배치 단위로 옵션 log-likelihood를 계산하고 MC cross-entropy loss를 구합니다.

학습 에폭 함수

한 에폭 동안의 학습을 수행합니다.

학습 실행 함수

모델 로드, LoRA 적용, 학습, 저장을 수행하는 통합 함수입니다.

In [4]:
```python
def build_mathqa_mc_dataset(mathqa_ds):
    """
    allenai/math_qa를 lm-eval mathqa와 동일한 MC 포맷으로 변환.
    options: 'a ) 24 , b ) 120 , ...' → choices: ['24', '120', ...]
    """
    data = []
    for doc in mathqa_ds:
        try:
            choices = [
                c[4:].rstrip(" ,")
                for c in re.findall(r"[abcd] \) .*?, |e \) .*?$", doc["options"])
            ]
            if len(choices) != 5:
                continue
            correct_idx = ["a", "b", "c", "d", "e"].index(doc["correct"])
        except (ValueError, IndexError):
            continue
        data.append({
            "question": doc["Problem"],
            "options": choices,
            "correct_idx": correct_idx,
        })
    return data
```

Scenario 1 학습 실행

MathQA train에서 1500개만 사용하여 학습하고 Google Drive에 업로드합니다.

In [5]:
```python
# MathQA train 로드
mathqa_ds = load_dataset("allenai/math_qa", split="train")
mathqa_mc = build_mathqa_mc_dataset(mathqa_ds)
print(f"MathQA train MC samples: {len(mathqa_mc)}")
ex = mathqa_mc[0]
print(f"Example Q: {ex['question'][:80]}...")
print(f"Options: {ex['options']}, correct_idx: {ex['correct_idx']}")
```

```
/usr/local/lib/python3.12/dist-packages/huggingface_hub/utils/_auth.py:104: UserWarning: 
Error while fetching `HF_TOKEN` secret value from your vault: 'Requesting secret HF_TOKEN timed out. Secrets can only 
be fetched when running from the Colab UI.'.
You are not authenticated with the Hugging Face Hub in this notebook.
If the error persists, please let us know by opening an issue on GitHub (https://github.com/huggingface/huggingface_h
ub/issues/new).
  warnings.warn(
```

```
MathQA train MC samples: 29837
Example Q: the banker ' s gain of a certain sum due 3 years hence at 10 % per annum is rs ....
Options: ['rs . 400', 'rs . 300', 'rs . 500', 'rs . 350', 'none of these'], correct_idx: 0
```

2.2 Grade-school-math MC 변환

Scenario 2 학습 실행

MathQA 1500개 + GSM 1500개를 혼합하여 학습합니다.

In [6]:
```python
def extract_final_answer(response: str):
    if not response or not response.strip():
        return None
    eq_matches = list(re.finditer(r"=\s*(-?\d+(?:\.\d+)?)\b", response))
    if eq_matches:
        last_eq = eq_matches[-1].group(1)
        try:
            val = float(last_eq)
            return str(int(val)) if val == int(val) else str(val)
        except ValueError:
            pass
    num_matches = list(re.finditer(r"(-?\d+(?:\.\d+)?)\b", response))
    if num_matches:
        last_num = num_matches[-1].group(1)
        try:
            val = float(last_num)
            return str(int(val)) if val == int(val) else str(val)
        except ValueError:
            pass
    return None

def to_mathqa_question(instruction: str) -> str:
    suffixes = ["\nGive me a solution to this problem", "\nCan you show me the way?", "\nSolve this step by step."]
    q = instruction.strip()
    for suf in suffixes:
        if q.endswith(suf):
            q = q[:-len(suf)].strip()
            break
    return q

def generate_options(correct_answer: str, n_options=5, seed=None):
    if seed is not None:
        random.seed(seed)
    try:
        val = float(correct_answer)
        is_int = val == int(val)
        ival = int(val) if is_int else val
    except (ValueError, TypeError):
        return None
    def fmt(x):
        return str(int(x)) if isinstance(x, float) and x == int(x) else str(x)
    candidates = []
    if is_int:
        for delta in [1, 2, -1, -2, 5, -5, 10, -10, 20, -20]:
            candidates.append(ival + delta)
        candidates.extend([ival * 2, ival * 3, ival // 2 if ival else 1, ival // 3 if ival else 1])
        if abs(ival) >= 50:
            candidates.extend([ival + 50, ival - 50, int(ival * 0.5), int(ival * 1.5)])
    else:
        for delta in [1.0, 2.0, -1.0, 0.5, -0.5, 5.0, -5.0]:
            candidates.append(val + delta)
        candidates.extend([val * 2, val * 3, val / 2, val / 3])
    wrong = []
    for c in candidates:
        try:
            fc = float(c)
            if fc != val and 0 < fc < 1e6:
                wrong.append(fmt(fc))
        except (ValueError, TypeError):
            pass
    wrong = list(dict.fromkeys(wrong))[: n_options - 1]
    if len(wrong) < n_options - 1:
        extra = [ival + 7, ival - 7, ival * 4] if is_int else [val + 3, val - 2]
        for e in extra:
            try:
                if float(e) != val and 0 < float(e) < 1e6 and fmt(e) not in wrong:
                    wrong.append(fmt(e))
            except (ValueError, TypeError):
                pass
    options = [correct_answer] + wrong[: n_options - 1]
    random.shuffle(options)
    return options, options.index(correct_answer)

def build_gsm_mc_dataset(raw_samples):
    data = []
    for i, s in enumerate(raw_samples):
        instruction = s.get("INSTRUCTION", "")
        response = s.get("RESPONSE", "")
        correct = extract_final_answer(response)
        if correct is None:
            continue
        result = generate_options(correct, seed=i)
        if result is None or len(result[0]) != 5:
            continue
        options, correct_idx = result
        data.append({"question": to_mathqa_question(instruction), "options": options, "correct_idx": correct_idx})
    return data
```

In [7]:
```
# Grade-school-math 로드
gsm_ds = load_dataset("qwedsacf/grade-school-math-instructions", split="train")
gsm_samples = [gsm_ds[i] for i in range(len(gsm_ds))]
gsm_mc = build_gsm_mc_dataset(gsm_samples)
print(f"Grade-school-math MC samples: {len(gsm_mc)}")
ex = gsm_mc[0]
print(f"Example Q: {ex['question'][:80]}...")
print(f"Options: {ex['options']}, correct_idx: {ex['correct_idx']}")
```

```
Grade-school-math MC samples: 8784
Example Q: This math problem has got me stumped: Natalia sold clips to 48 of her friends in...
Options: ['74', '73', '72', '70', '71'], correct_idx: 2
```

lm_eval 평가 함수

lm-evaluation-harness를 사용하여 MathQA test split으로 모델을 평가합니다.

모델 평가 실행

학습된 두 모델(MathQA-only, MathQA+GSM)을 lm_eval로 평가합니다.

최종 결과 요약

평가 결과를 DataFrame으로 정리하여 출력합니다.

3. Dataset 클래스 및 학습 유틸

In [8]:
```
MAX_PREFIX_LEN = 256
MAX_FULL_LEN = 320

class MCDatasetAligned(Dataset):
    def __init__(self, data, tokenizer):
        self.data = data
        self.tokenizer = tokenizer
    def __len__(self):
        return len(self.data)
    def __getitem__(self, idx):
        item = self.data[idx]
        prefix = f"Question: {item['question']}\nAnswer:"
        continuations = [f" {opt}" for opt in item["options"]]
        return {"prefix": prefix, "continuations": continuations, "correct_idx": item["correct_idx"]}

def collate_batch(batch):
    return {
        "prefix": [b["prefix"] for b in batch],
        "continuations": [b["continuations"] for b in batch],
        "correct_idx": torch.tensor([b["correct_idx"] for b in batch], dtype=torch.long),
    }
```

In [9]:
```
def compute_option_log_likelihoods_batched(model, tokenizer, prefix, continuations, device):
    per_sample = continuations and isinstance(continuations[0], (list, tuple))
    if per_sample:
        batch_size = len(prefix)
        log_likelihoods_per_option = []
        for k in range(5):
            full_texts = [prefix[i] + continuations[i][k] for i in range(batch_size)]
            full_enc = tokenizer(full_texts, return_tensors="pt", truncation=True, max_length=MAX_FULL_LEN, padding=
True, add_special_tokens=True)
            full_ids = full_enc.input_ids.to(device)
            attn_mask = full_enc.attention_mask.to(device)
            prefix_lengths = tokenizer(prefix, return_tensors="pt", truncation=True, max_length=MAX_PREFIX_LEN, padd
ing=True, add_special_tokens=True).attention_mask.sum(dim=1)
            outputs = model(full_ids, attention_mask=attn_mask)
            logits = outputs.logits
            batch_lls = []
            for b in range(batch_size):
                n_prefix = prefix_lengths[b].item()
                n_full = attn_mask[b].sum().item()
                n_cont = n_full - n_prefix
                if n_cont <= 0:
                    batch_lls.append(logits[b, 0, 0] * 0.0 - 1e9)
                    continue
                ll_sum = logits[b, 0, 0] * 0.0
                for j in range(n_cont):
                    pos = n_prefix - 1 + j
                    next_id = full_ids[b, pos + 1].item()
                    ll_sum = ll_sum + F.log_softmax(logits[b, pos], dim=-1)[next_id]
                batch_lls.append(ll_sum)
            log_likelihoods_per_option.append(torch.stack(batch_lls))
        out = torch.stack(log_likelihoods_per_option, dim=1)
        return out.squeeze(0) if batch_size == 1 else out
    return None

def mc_cross_entropy_loss(log_likelihoods, correct_idx):
    if log_likelihoods.dim() == 1:
        log_likelihoods = log_likelihoods.unsqueeze(0)
    if not isinstance(correct_idx, torch.Tensor):
        correct_idx = torch.tensor([correct_idx], device=log_likelihoods.device, dtype=torch.long)
    elif correct_idx.dim() == 0:
        correct_idx = correct_idx.unsqueeze(0)
    log_probs = F.log_softmax(log_likelihoods, dim=-1)
    return F.nll_loss(log_probs, correct_idx)
```

In [10]:
```
def train_epoch(model, tokenizer, dataloader, optimizer, scheduler, device, epoch):
    model.train()
    total_loss = 0.0
    pbar = tqdm(dataloader, desc=f"Epoch {epoch}")
    for item in pbar:
        prefix = item["prefix"]
        continuations = item["continuations"]
        correct_idx = item["correct_idx"].to(device)
        log_likelihoods = compute_option_log_likelihoods_batched(model, tokenizer, prefix, continuations, device)
        loss = mc_cross_entropy_loss(log_likelihoods, correct_idx)
        optimizer.zero_grad()
        loss.backward()
        torch.nn.utils.clip_grad_norm_(model.parameters(), 1.0)
        optimizer.step()
        if scheduler:
            scheduler.step()
        total_loss += loss.item()
        pbar.set_postfix({"loss": f"{loss.item():.4f}"})
    return total_loss / len(dataloader)
```

```
In [11]: def run_training(train_data, output_dir, model_id="Qwen/Qwen2.5-0.5B", num_epochs=3, batch_size=4, lr=2e-4):
    tokenizer = AutoTokenizer.from_pretrained(model_id, trust_remote_code=True)
    if tokenizer.pad_token is None:
        tokenizer.pad_token = tokenizer.eos_token
    model = AutoModelForCausalLM.from_pretrained(model_id, torch_dtype=torch.bfloat16, device_map="auto", trust_remo
te_code=True, attn_implementation=ATTN_IMPL)
    lora_config = LoraConfig(r=16, lora_alpha=32, lora_dropout=0.05, bias="none", task_type=TaskType.CAUSAL_LM, targ
et_modules=["q_proj", "k_proj", "v_proj", "o_proj", "gate_proj", "up_proj", "down_proj"])
    model = get_peft_model(model, lora_config)
    model.enable_input_require_grads()  # required for LoRA + gradient checkpointing
    model.gradient_checkpointing_enable()  # trade compute for memory
    dataset = MCDatasetAligned(train_data, tokenizer)
    split_idx = int(len(dataset) * 0.9)
    train_ds, eval_ds = torch.utils.data.random_split(dataset, [split_idx, len(dataset) - split_idx])
    train_loader = DataLoader(train_ds, batch_size=batch_size, shuffle=True, collate_fn=collate_batch, num_workers=
0)
    optimizer = torch.optim.AdamW(model.parameters(), lr=lr)
    total_steps = len(train_loader) * num_epochs
    scheduler = get_linear_schedule_with_warmup(optimizer, num_warmup_steps=int(0.03 * total_steps), num_training_st
eps=total_steps)
    device = next(model.parameters()).device
    for epoch in range(1, num_epochs + 1):
        train_loss = train_epoch(model, tokenizer, train_loader, optimizer, scheduler, device, epoch)
        print(f"Epoch {epoch} | Train Loss: {train_loss:.4f}")
    model.save_pretrained(output_dir)
    tokenizer.save_pretrained(output_dir)
    print(f"Saved to {output_dir}")
    return output_dir
```

4. Scenario 1: MathQA train만

```
In [12]: # MathQA train 1500개만 사용 (랜덤 서브샘플, 시드 고정)
random.seed(42)
mathqa_mc_1500 = random.sample(mathqa_mc, 1500)
print(f"Training with {len(mathqa_mc_1500)} MathQA samples")

OUTPUT_MATHQA_ONLY = "./outputs/04_mathqa_only"
run_training(mathqa_mc_1500, OUTPUT_MATHQA_ONLY, num_epochs=3, batch_size=4)

# Upload ./outputs/04_mathqa_only to Google Drive
from google.colab import drive
import shutil

drive.mount("/content/drive")
dst = "/content/drive/MyDrive/outputs/04_mathqa_only"
shutil.copytree(OUTPUT_MATHQA_ONLY, dst, dirs_exist_ok=True)
print(f"Uploaded to {dst}")
```

Base 모델 평가 (선택, 비교용)

5. Scenario 2: MathQA + Grade-school-math 혼합

```
In [14]: # Free GPU and variables for memory (before Scenario 2)
import gc
gc.collect()
torch.cuda.empty_cache()

# MathQA + GSM 혼합 (GSM 1500개 샘플, 셔플)
random.seed(42)
gsm_mc_1500 = random.sample(gsm_mc, 1500)
random.seed(42)
mathqa_mc_1500 = random.sample(mathqa_mc, 1500)

combined_mc = mathqa_mc_1500 + gsm_mc_1500
random.shuffle(combined_mc)
print(f"Combined train samples: {len(combined_mc)} (MathQA: {len(mathqa_mc_1500)}, GSM: {len(gsm_mc_1500)})")

OUTPUT_COMBINED = "./outputs/04_mathqa_gsm_combined"
run_training(combined_mc, OUTPUT_COMBINED, num_epochs=1, batch_size=2)
```

```
Combined train samples: 3000 (MathQA: 1500, GSM: 1500)

`torch_dtype` is deprecated! Use `dtype` instead!
Epoch 1:   0%|          | 0/1350 [00:00<?, ?it/s]`use_cache=True` is incompatible with gradient checkpointing. Settin
g `use_cache=False`.
Epoch 1: 100%|██████████| 1350/1350 [30:49<00:00,  1.37s/it, loss=1.6406]

Epoch 1 | Train Loss: 1.6721
Saved to ./outputs/04_mathqa_gsm_combined
```

```
Out[14]: './outputs/04_mathqa_gsm_combined'
```

In [15]:
```
# Upload ./outputs/04_mathqa_only to Google Drive
from google.colab import drive
import shutil

drive.mount("/content/drive")
dst = "/content/drive/MyDrive/outputs/04_mathqa_gsm_combined"
shutil.copytree(OUTPUT_COMBINED, dst, dirs_exist_ok=True)
print(f"Uploaded to {dst}")
```

```
Drive already mounted at /content/drive; to attempt to forcibly remount, call drive.mount("/content/drive", force_rem
ount=True).
Uploaded to /content/drive/MyDrive/outputs/04_mathqa_gsm_combined
```

6. lm_eval로 MathQA test 평가

In [16]:
```
def evaluate_with_lm_eval(model_path, model_name):
    print(f"\n{'='*60}")
    print(f"Evaluating: {model_name}")
    print(f"Path: {model_path}")
    print(f"{'='*60}")
    is_local = model_path.startswith(".") or model_path.startswith("/")
    model = AutoModelForCausalLM.from_pretrained(model_path, torch_dtype=torch.float16, device_map="auto", trust_rem
ote_code=True)
    tokenizer = AutoTokenizer.from_pretrained(model_path, trust_remote_code=True, use_fast=not is_local)
    lm = HFLM(pretrained=model, tokenizer=tokenizer, max_length=1024, batch_size='auto', trust_remote_code=True)
    results = lm_eval.simple_evaluate(model=lm, tasks=["mathqa"], task_manager=lm_eval.tasks.TaskManager())
    acc = results['results']['mathqa']['acc,none']
    acc_stderr = results['results']['mathqa'].get('acc_stderr,none', 0)
    acc_norm = results['results']['mathqa'].get('acc_norm,none', acc)
    print(f"{model_name}: acc={acc:.4f} (+/- {acc_stderr:.4f}), acc_norm={acc_norm:.4f}")
    del model, tokenizer, lm
    torch.cuda.empty_cache()
    import gc
    gc.collect()
    return {"acc": acc, "acc_stderr": acc_stderr, "acc_norm": acc_norm}
```

In [17]:
```
eval_results = {}

# Base 모델 (선택, 비교용)
# eval_results["Base"] = evaluate_with_lm_eval("Qwen/Qwen2.5-0.5B", "Qwen2.5-0.5B (Base)")

eval_results["MathQA-only"] = evaluate_with_lm_eval(OUTPUT_MATHQA_ONLY, "MathQA-only")
eval_results["MathQA+GSM"] = evaluate_with_lm_eval(OUTPUT_COMBINED, "MathQA+GSM")
```

In [ ]:
```
# 최종 결과 요약
import pandas as pd
rows = [
    {"Model": "MathQA-only", "Accuracy": eval_results["MathQA-only"]["acc"], "Std Error": eval_results["MathQA-onl
y"]["acc_stderr"]},
    {"Model": "MathQA+GSM", "Accuracy": eval_results["MathQA+GSM"]["acc"], "Std Error": eval_results["MathQA+GSM"]
["acc_stderr"]},
]
if "Base" in eval_results:
    rows.insert(0, {"Model": "Base", "Accuracy": eval_results["Base"]["acc"], "Std Error": eval_results["Base"]["acc
_stderr"]})
df = pd.DataFrame(rows)
print("\n=== Final MathQA Test Results (lm_eval) ===")
print(df.to_string(index=False))
```