아키텍처 문서

날짜	@August 10, 2025 12:27 PM
단계	개발 단계
유형	참고문서
작성자	㉿ 종철 이

FSKU LLM Challenge Model Architecture Document (v7 - 감사 수정본)

1. High Level Architecture

High Level Project Diagram (v3)

'Distill-M 2'의 대조적 증류 파이프라인을 반영하여, 학습 단계를 **'응답 생성 → 데이터 재포맷 → 최종 훈련'** 의 명확한 단일 흐름으로 구성합니다.

```
graph TD
    subgraph "Phase 1: 학습 과정 (Unconstrained Environment)"
        A[External Data] --> B(Data Processing);
        B --> C[FAISS Index Builder];
        B --> D[Fine-tuning Dataset];

        %% Distill-M 2 Workflow
        D --> E{Response Generation};
        E -- Teacher Model --> E;
        E -- Student Model --> E;
        E --> F[Reformat Data];
        F --> G(DistiLLM-2 Training);

        %% Validation
```

```
    G --> V[Validation];
    V -->|Score Check| G;
end

subgraph "Phase 2: 추론 과정 (Competition Environment)"
    H(Quantized Student Model) --> J[Inference Engine];
    I[FAISS Index] --> J;
    K(test.csv) --> J;
    J --> L(submission.csv);
end

G --> Q[Quantization];
Q --> H;
C --> I;
```

Architectural and Design Patterns

- `Contrastive Distillation (Distill-M 2)` : 교사-학생 모델의 응답을 모두 생성하고, 그 차이를 학습하여 학생 모델의 성능을 극대화하는 핵심 전략입니다.
- `Retrieval-Augmented Generation (RAG)` : 외부 지식 베이스를 활용하여 답변의 사실 근거를 마련하는 핵심 패턴입니다.
- `Modular Pipeline` : 개발 및 테스트 용이성을 위한 모듈형 스크립트 구조.
- `Monorepo` : 재현성과 관리 용이성을 위한 단일 저장소 구조.
- `Adapter Pattern (LoRA)` : PEFT 라이브러리를 활용한 효율적인 파인튜닝.
- `Circuit Breaker` : 개별 질문 처리 시간 초과 방지를 통해 전체 추론 시간 준수.
- `Cache-Aside` : 임베딩 및 검색 결과 캐싱으로 반복 연산 감소 및 속도 향상.

---

2. Tech Stack

Category	Technology	Version	Purpose
Language	Python	`3.10`	주요 개발 언어
ML Framework	PyTorch	`2.1.0`	딥러닝 프레임워크

Core ML	Transformers	`~4.43.2`	LLM 로드/사용
Optimization	Accelerate	`~0.29.2`	학습/추론 최적화
Response Gen	vllm	`~0.5.4`	교사/학생 응답 생성
Fine-Tuning	TRL	`~0.9.6`	`DistiLLMTrainer` 사용
Embedding	sentence-transformers	`~2.7.0`	벡터 임베딩
Vector DB	FAISS-CPU	`~1.8.0`	벡터 검색
Quantization	auto-gptq / bitsandbytes	`~0.7.1` / `~0.43.1`	GPTQ/QLoRA 양자화
Data/Doc Proc	Pandas, LangChain, PyMuPDF	`~2.2` , `~0.2` , `~1.24`	데이터/문서 처리
Search	bm25s	최신	Sparse 검색
Korean NLP	konlpy	`~0.6.0`	한국어 형태소 분석
Monitoring	tqdm, wandb	`~4.66` , `~0.17`	진행 표시, 실험 추적
Testing	pytest	`~8.2.0`	단위/통합 테스트

Model Selection Strategy

- **Student Model Candidates:** `mistralai/Mistral-7B-Instruct-v0.2` , `Solar-10.7B-Instruct` , `Qwen2.5-1.5B-Instruct`
- **Teacher Model Candidates:** `Meta-Llama-3.1-70B-Instruct` , `Qwen2.5-7B-Instruct`

---

3. Data Models

시스템의 각 컴포넌트가 주고받는 데이터의 형식을 명확히 정의합니다.

```
from typing import TypedDict, List, Optional, Dict

class DocumentChunk_v2(TypedDict):
    chunk_id: str
    content: str
    source: str
    domain: str
    embedding: Optional[List[float]]
    quality_score: float

class SyntheticQAPair_v2(TypedDict):
```

```
    qa_id: str
    question: str
    answer: str
    context_chunk_ids: List[str]
    teacher_model: str
    overall_quality_score: float
    difficulty_level: str
    student_loss: Optional[float]

class CompetitionQuestion_v2(TypedDict):
    ID: str
    Question: str
    question_type: str
    parsed_question: str
    choices: Optional[Dict[str, str]]

class SubmissionRow_v2(TypedDict):
    ID: str
    answer: str
    confidence: float
    inference_time: float
    used_fallback: bool
```

---

4. Components

시스템은 단일 책임 원칙(SRP)에 따라 10개의 독립적인 컴포넌트로 구성됩니다.

1. **데이터 전처리 컴포넌트**
2. **지식 베이스 컴포넌트**
3. **합성 데이터 생성 컴포넌트**
4. **모델 파인튜닝 컴포넌트**
5. **추론 오케스트레이터**
6. **최적화 컴포넌트**

7. **평가 및 모니터링 컴포넌트**
8. **캐싱 컴포넌트**
9. **질문 분류 컴포넌트**
10. **다단계 검색 컴포넌트**
    a. **LLM-as-Reranker 프롬프트**(컨텍스트 k개와 질의를 입력, 상위 n개 선택)
    b. BM25는 `bm25s`를 사용한다
    c. 스코어 융합은 정규화 후 가중합(weighted sum) 방식을 사용한다

---

5. External APIs

학습 및 준비 단계에서 의존하는 외부 서비스 목록입니다.

1. **Hugging Face Hub API (Models & Tokenizers)**
2. **Hugging Face Datasets API**
3. **Korean NLP Resources (KoNLPy, AIHub 등)**

---

6. Core Workflows

Workflow 1: 모델 개발

```
sequenceDiagram
    participant User
    participant DataPrep as Data Preprocessing
    participant SynGen as Synthetic Data Gen
    participant Eval as Evaluation
    participant FineTune as Model Fine-tuning
    participant Optim as Optimization

    User->>DataPrep: 원본 문서 제공
    DataPrep->>SynGen: DocumentChunk 전달

    SynGen->>SynGen: Teacher Model로 생성
```

```
    SynGen->>Eval: 품질 평가 요청

    alt 품질 < 7.0
        Eval-->>SynGen: 재생성 요청
    else 품질 >= 7.0
        Eval-->>FineTune: 승인된 데이터셋
    end

    loop Distill-M 2 Iterations
        FineTune->>FineTune: 재학습
    end

    FineTune->>Optim: 최종 모델 전달
    Optim->>Optim: 4-bit Quantization
    Optim-->>User: 경량화된 최종 모델
```

Workflow 2: RAG 추론

```
sequenceDiagram
    participant Runner as Competition Runner
    participant Orchestrator as Inference Orchestrator
    participant Cache as Cache Layer
    participant Retriever as Multi-Stage Retriever
    participant Model as Fine-tuned Model
    participant Fallback as Fallback Handler

    Runner->>Orchestrator: Batch[CompetitionQuestion]

    loop For each question
        Orchestrator->>Cache: 쿼리
        alt Cache Miss
            Cache->>Retriever: 검색 요청
            Retriever-->>Cache: 컨텍스트
            Cache->>Cache: 저장
        end
```

```
    Orchestrator->>Model: Generate with timeout

    alt 정상 처리
        Model-->>Orchestrator: 답변
    else Timeout or Low Confidence
        Orchestrator->>Fallback: 긴급 처리
        Fallback-->>Orchestrator: 대체 답변
    end
end

Orchestrator-->>Runner: List[SubmissionRow]
```

---

7. Source Tree

```
/
├── data/
│   ├── raw/, processed/, finetune/, knowledge_base/
├── models/
│   ├── student/, teacher/
├── packages/
│   ├── preprocessing/, training/, inference/
├── scripts/
├── tests/
├── requirements.txt
└── README.md
```

---

8. Infrastructure and Deployment

- **Infrastructure:** 대회에서 제공하는 Docker 컨테이너 환경.

- **Deployment:** 대회 규칙에 따라 필수 파일을 포함한 압축 파일을 생성하여 제출하는 과정.

Appendix A: 데이터 파이프라인 기술 가이드

- **권장 파이프라인:**
  1. **파싱:** `PyMuPDF` 를 주력 파서로 사용.
  2. **전처리:** `ftfy` 라이브러리 및 정규식을 통한 국영문 혼합 텍스트 정제.
  3. **청킹:** `RecursiveCharacterTextSplitter` 를 기본 전략으로 채택 (청크 사이즈 400, 오버랩 20).
     a. 한글과 영어 혼합 문서에는 `separators=["\n\n","\n"," ","."...]` 에 한국어 마침표(`。`), 쉼표(`，`)를 추가하고 `chunk_size` 를 300–600 token(문장·문단 단위)으로, `chunk_overlap` 은 10–20 % 수준으로 설정한다.
  4. **청킹 전략:** "필드별(제목/본문/표) 다른 청킹 전략 적용", "숫자/통화 표현 표준화"
  5. **한국어 처리:** `KoNLPy(Mecab)` 를 사용한 선택적 형태소 분석.
  6. **검색 전략:** BM25와 벡터 검색을 결합한 **하이브리드 검색** 구현.
  7. **FAISS 최적화:** "데이터 규모에 따른 `nlist` , `nprobe` 파라미터 튜닝"

Appendix B: Distillm-2의 핵심 컴포넌트 원본코드

1. 데이터 재형식화 (Data Reformatting) reformat.py:1-54

이 스크립트는 teacher와 student 모델의 개별 응답을 DPO 훈련에 적합한 chosen/rejected 쌍으로 변환합니다.

```
import os
import json
import datasets
from datasets import load_dataset, concatenate_datasets, DatasetDict
from tqdm import tqdm
import argparse
```

```
def main(args):
    teacher_data = load_dataset('json', data_files=args.teacher_file, split='trai
n')
    student_data = load_dataset('json', data_files=args.student_file, split='trai
n')

    # make sure the pair
    samples = []
    dict_teacher = {x['prompt']: str(x) for x in teacher_data}
    dict_student = {x['prompt']: str(x) for x in student_data}

    for p in teacher_data['prompt']:
        try:
            chosen, rejected = eval(dict_teacher[p]), eval(dict_student[p])
            chosen = [
                {"content": p, "role": "user"},
                {"content": chosen['generated_text'], "role": "assistant"}
            ]
            rejected = [
                {"content": p, "role": "user"},
                {"content": rejected['generated_text'], "role": "assistant"}
            ]
            samples.append({"prompt": p, "chosen": chosen, "rejected": rejecte
d})

        except:
            continue

    if not os.path.exists(args.output_dir):
        os.makedirs(args.output_dir)
    with open(f'{args.output_dir}/train.json', 'w') as json_file:
        json.dump(samples, json_file)

    dataset = DatasetDict({
```

```
        'train': load_dataset('json', data_files=f'{args.output_dir}/train.json', split
='train'),
        'test': load_dataset('json', data_files=f'{args.output_dir}/train.json', split
='train').select(range(500)),
    })
    dataset.save_to_disk(args.output_dir)
    print (f"Binarized datasets save to {os.path.join(args.output_dir)}")

if __name__ == "__main__":
    parser = argparse.ArgumentParser()
    parser.add_argument("--teacher_file", type=str, required=True)
    parser.add_argument("--student_file", type=str, required=True)
    parser.add_argument("--output_dir", type=str, required=True)
    args = parser.parse_args()

    main(args)
```

2. vLLM 응답 생성 (Response Generation) generate_vllm.py:1-102

vLLM을 사용한 고효율 응답 생성 스크립트로, teacher와 student 모델 모두에서 사용됩니다.

```
from vllm import SamplingParams, LLM
from transformers import AutoTokenizer
from datasets import load_dataset, load_from_disk

import argparse
import json
import torch, time, json, os
from pathlib import Path
from tqdm import tqdm
from datetime import timedelta

import warnings
warnings.filterwarnings("ignore")

parser = argparse.ArgumentParser(description='Decode with vLLM')
```

```
parser.add_argument('--data_dir', type=str, default="ultrachat",
                    help='Directory containing the data')
parser.add_argument('--iter', type=int, default='1', help='training iteration')
parser.add_argument('--model', type=str, default='Qwen/Qwen2.5-0.5B-Instr
uct',
                    help='Path to the SLM model')
parser.add_argument('--teacher-model', type=str, default=None,
                    help='Path to the LLM model.')
parser.add_argument('--temperature', type=float, default=0.8,
                    help='Temperature for sampling')
parser.add_argument('--top_p', type=float, default=0.95,
                    help='Top-p probability for sampling')
parser.add_argument('--eps', type=float, default=0.04,
                    help='epsilon for typical acceptance sampler')
parser.add_argument('--max_tokens', type=int, default=1024,
                    help='Maximum number of tokens to generate')
parser.add_argument('--seed', type=int, default=42,
                    help='Random seed')
parser.add_argument('--output_dir', type=str, default="datasets/phi3_ultrafee
dback",
                    help='output_dir')
parser.add_argument('--split', type=str, default='train_prefs')
parser.add_argument('--frac_idx', type=int, default=0)
parser.add_argument('--frac_size', type=int, default=0)
parser.add_argument('--lora_path', type=str, default=None)

args = parser.parse_args()

data_dir = args.data_dir

# this is recommended for gemma-2 models; otherwise it is not needed
if 'gemma-2' in args.model:
    os.environ["VLLM_ATTENTION_BACKEND"] = "FLASHINFER"

if args.lora_path is not None:
    llm = LLM(model=args.model, enable_lora=True)
```

```
else:
    llm = LLM(model=args.model, dtype="bfloat16", tensor_parallel_size=2,)
tokenizer = llm.get_tokenizer()


if args.data_dir == 'evol-instruct':
    data_dir = "eval/evol-instruct/evol_inst_eval.json"
    train_dataset = load_dataset('json', data_files=data_dir, split='train')
    prompts = train_dataset['prompt']
elif args.data_dir == "alpaca-eval":
    data_dir = "eval/alpacaeval/alpaca_eval.json"
    train_dataset = load_dataset('json', data_files=data_dir, split='train')
    prompts = train_dataset['instruction']
elif args.data_dir == "ultrachat":
    prompts = [
        example[0]['content'] for example in load_dataset(f'UCLA-AGI/SPIN_iter
{args.iter}', split='train')['generated']
    ]
else:
    train_dataset= load_dataset(data_dir, split=args.split)
    prompts = sorted(list(set(train_dataset['prompt'])))

if args.frac_size > 0:
    assert args.frac_size > args.frac_idx
    sub_len = len(prompts) // args.frac_size + 1
    if sub_len*(args.frac_idx+1) > len(prompts):
        prompts = prompts[sub_len*args.frac_idx:]
    else:
        prompts = prompts[sub_len*args.frac_idx:sub_len*(args.frac_idx+1)]
else:
    prompts = prompts[:]

conversations = [tokenizer.apply_chat_template([{'role': 'user', 'content': prom
pt}], tokenize=False, add_generation_prompt=True) for prompt in prompts]

sampling_params = SamplingParams(
```

```
    temperature=args.temperature, top_p=args.top_p, max_tokens=args.max_t
okens, seed=args.seed
)
if args.lora_path is not None:
    from vllm.lora.request import LoRARequest
    outputs = llm.generate(conversations, sampling_params, lora_request=LoR
ARequest("lora", 1, args.lora_path))
else:
    outputs = llm.generate(conversations, sampling_params)

# Save the outputs as a JSON file.
output_data = []
for i, output in tqdm(enumerate(outputs)):
    prompt = output.prompt
    generated_text=output.outputs[0].text
    output_data.append({
        'prompt': prompts[i],
        "format_prompt": prompt,
        'generated_text': generated_text,
    })
```

3. DistiLLM 훈련 메인 스크립트 run_distillm.py:1-145

```
#!/usr/bin/env python
# coding=utf-8
# Copyright 2023 The HuggingFace Inc. team. All rights reserved.
#
# Licensed under the Apache License, Version 2.0 (the "License");
# you may not use this file except in compliance with the License.
# You may obtain a copy of the License at
#
#     http://www.apache.org/licenses/LICENSE-2.0
#
# Unless required by applicable law or agreed to in writing, software
# distributed under the License is distributed on an "AS IS" BASIS,
```

```
# WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or i
mplied.
# See the License for the specific language governing permissions and
# limitations under the License.
import logging
import random
import sys
import os

import torch
import transformers
from transformers import AutoModelForCausalLM, set_seed
from datasets import load_dataset, DatasetDict

from alignment import (
    DataArguments,
    DPOConfig,
    H4ArgumentParser,
    ModelArguments,
    apply_chat_template,
    decontaminate_humaneval,
    get_checkpoint,
    get_datasets,
    get_kbit_device_map,
    get_peft_config,
    get_quantization_config,
    get_tokenizer,
    is_adapter_model,
)
from peft import PeftConfig, PeftModel
from distillm_trainer import DistiLLMTrainer

logger = logging.getLogger(__name__)


def main():
```

```
    parser = H4ArgumentParser((ModelArguments, DataArguments, DPOConfi
g))
    model_args, data_args, training_args = parser.parse()

    #######
    # Setup
    #######
    logging.basicConfig(
        format="%(asctime)s - %(levelname)s - %(name)s - %(message)s",
        datefmt="%Y-%m-%d %H:%M:%S",
        handlers=[logging.StreamHandler(sys.stdout)],
    )
    log_level = training_args.get_process_log_level()
    logger.setLevel(log_level)
    transformers.utils.logging.set_verbosity(log_level)
    transformers.utils.logging.enable_default_handler()
    transformers.utils.logging.enable_explicit_format()

    # Log on each process the small summary:
    logger.info(f"Model parameters {model_args}")
    logger.info(f"Data parameters {data_args}")
    logger.info(f"Training/evaluation parameters {training_args}")

    # Check for last checkpoint
    last_checkpoint = get_checkpoint(training_args)
    if last_checkpoint is not None and training_args.resume_from_checkpoint is
None:
        logger.info(f"Checkpoint detected, resuming training at {last_checkpoint
=}.")

    # Set seed for reproducibility
    set_seed(training_args.seed)

    ###############
    # Load datasets
    ###############
```

```
    raw_datasets = get_datasets(
        data_args,
        splits=data_args.dataset_splits,
        configs=data_args.dataset_configs,
        columns_to_keep=["messages", "chosen", "rejected", "prompt", "complet
ion", "label"],
    )
    logger.info(
        f"Training on the following splits: {[split + ' : ' + str(dset.num_rows) for sp
lit, dset in raw_datasets.items()]}"
    )
    column_names = list(raw_datasets["train"].features)

    #####################################
    # Load tokenizer and process datasets
    #####################################
    data_args.truncation_side = "left"  # Truncate from left to ensure we don't lo
se labels in final turn
    tokenizer = get_tokenizer(model_args, data_args)

    ######################
    # Apply chat template
    ######################
    raw_datasets = raw_datasets.map(
        apply_chat_template,
        fn_kwargs={
            "tokenizer": tokenizer,
            "task": "dpo",
            "auto_insert_empty_system_msg": data_args.auto_insert_empty_syste
m_msg,
        },
        num_proc=data_args.preprocessing_num_workers,
        remove_columns=column_names,
        desc="Formatting comparisons with prompt template",
    )
```

```
    ############################
    # Decontaminate benchmarks
    ############################
    num_raw_train_samples = len(raw_datasets["train"])
    raw_datasets = raw_datasets.filter(
        decontaminate_humaneval,
        fn_kwargs={"text_column": "text_chosen"},
        batched=True,
        batch_size=10_000,
        num_proc=1,
        desc="Decontaminating HumanEval samples",
    )
    num_filtered_train_samples = num_raw_train_samples - len(raw_datasets["tr
ain"])
    logger.info(
        f"Decontaminated {num_filtered_train_samples} ({num_filtered_train_sam
ples/num_raw_train_samples * 100:.2f}%) samples from the training set."
    )

    # Replace column names with what TRL needs, text_chosen -> chosen and
text_rejected -> rejected
    for split in ["train", "test"]:
        raw_datasets[split] = raw_datasets[split].rename_columns(
            {"text_prompt": "prompt", "text_chosen": "chosen", "text_rejected": "re
jected"}
        )

    # Log a few random samples from the training set:
    for index in random.sample(range(len(raw_datasets["train"])), 3):
        logger.info(f"Prompt sample {index} of the raw training set:\n\n{raw_data
sets['train'][index]['prompt']}")
        logger.info(f"Chosen sample {index} of the raw training set:\n\n{raw_data
sets['train'][index]['chosen']}")
        logger.info(f"Rejected sample {index} of the raw training set:\n\n{raw_dat
asets['train'][index]['rejected']}")
```

```
    torch_dtype = (
        model_args.torch_dtype if model_args.torch_dtype in ["auto", None] else
getattr(torch, model_args.torch_dtype)
    )
    quantization_config = get_quantization_config(model_args)
```

전체 훈련 파이프라인을 조율하는 메인 스크립트입니다. run_distillm.py:181-210

```
if model_args.ref_model_name_or_path is None:
        ref_model = model
    else:
        ref_model = model_args.ref_model_name_or_path
    ref_model_kwargs = model_kwargs

    ############################
    # Instantiate DPO trainer
    ############################
    trainer = DistiLLMTrainer(
        model,
        ref_model,
        model_init_kwargs=model_kwargs,
        ref_model_init_kwargs=ref_model_kwargs,
        args=training_args,
        beta=training_args.beta,
        train_dataset=raw_datasets["train"],
        eval_dataset=raw_datasets["test"],
        tokenizer=tokenizer,
        max_length=training_args.max_length,
        max_prompt_length=training_args.max_prompt_length,
        peft_config=get_peft_config(model_args),
        loss_type=training_args.loss_type,
        force_use_ref_model=True,
    )

    ################
    # Training loop
```

```
    ################
    checkpoint = None
```

DistiLLMTrainer 초기화 부분입니다. run_distillm.py:244-264

```
  ###########
  # Evaluate
  ###########
    if training_args.do_eval:
        logger.info("*** Evaluate ***")
        metrics = trainer.evaluate()
        metrics["eval_samples"] = len(raw_datasets["test"])
        trainer.log_metrics("eval", metrics)
        trainer.save_metrics("eval", metrics)

    if training_args.push_to_hub is True:
        logger.info("Pushing to hub...")
        trainer.push_to_hub(**kwargs)

    logger.info("*** Training complete! ***")


if __name__ == "__main__":
    os.environ["WANDB_DISABLED"] = "true"
    main()
```

평가 및 모델 저장 부분입니다.

4. DistiLLM 핵심 손실 함수 distillm_trainer.py:1-170

```
# DPO Authors: Rafael Rafailov, Archit Sharma, Eric Mitchell, Stefano Ermon, C
hristopher D. Manning, and Chelsea Finn 2023
# Copyright 2023 The HuggingFace Team. All rights reserved.
#
# Licensed under the Apache License, Version 2.0 (the "License");
# you may not use this file except in compliance with the License.
```

```
# You may obtain a copy of the License at
#
#     http://www.apache.org/licenses/LICENSE-2.0
#
# Unless required by applicable law or agreed to in writing, software
# distributed under the License is distributed on an "AS IS" BASIS,
# WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or i
mplied.
# See the License for the specific language governing permissions and
# limitations under the License.
import math
import inspect
import random
import warnings
from collections import defaultdict
from contextlib import contextmanager, nullcontext
from copy import deepcopy
from functools import wraps
from typing import Any, Callable, Dict, List, Literal, Optional, Tuple, Union

import numpy as np
import torch
import torch.nn as nn
import torch.nn.functional as F
from accelerate import PartialState
from accelerate.utils import is_deepspeed_available, tqdm
from datasets import Dataset
from huggingface_hub.utils._deprecation import _deprecate_arguments
from torch.utils.data import DataLoader
from transformers import (
    AutoModelForCausalLM,
    DataCollator,
    PreTrainedModel,
    PreTrainedTokenizerBase,
    Trainer,
)
```

```
from transformers.models.auto.modeling_auto import MODEL_FOR_VISION_2_
SEQ_MAPPING_NAMES
from transformers.trainer_callback import TrainerCallback
from transformers.trainer_utils import EvalLoopOutput

from trl.import_utils import is_peft_available, is_wandb_available
from trl.models import PreTrainedModelWrapper, create_reference_model
from trl.trainer.dpo_config import DPOConfig, FDivergenceConstants, FDiverg
enceType
from trl.trainer.utils import (
    DPODataCollatorWithPadding,
    RunningMoments,
    SyncRefModelCallback,
    cap_exp,
    disable_dropout_in_model,
    pad_to_length,
    peft_module_casting_to_bf16,
    trl_sanitze_kwargs_for_tagging,
)

if is_peft_available():
    from peft import PeftModel, get_peft_model, prepare_model_for_kbit_trainin
g


if is_wandb_available():
    import wandb

if is_deepspeed_available():
    import deepspeed


class DistiLLMTrainer(Trainer):
    r"""
    Initialize DPOTrainer.
```

```
    Args:
        model (`transformers.PreTrainedModel`):
            The model to train, preferably an `AutoModelForSequenceClassificatio
n`.
        ref_model (`PreTrainedModelWrapper`):
            Hugging Face transformer model with a casual language modelling hea
d. Used for implicit reward computation and loss. If no
            reference model is provided, the trainer will create a reference model
with the same architecture as the model to be optimized.
        args (`DPOConfig`):
            The DPO config arguments to use for training.
        data_collator (`transformers.DataCollator`):
            The data collator to use for training. If None is specified, the default da
ta collator (`DPODataCollatorWithPadding`) will be used
            which will pad the sequences to the maximum length of the sequences
in the batch, given a dataset of paired sequences.
        train_dataset (`datasets.Dataset`):
            The dataset to use for training.
        eval_dataset (`datasets.Dataset`):
            The dataset to use for evaluation.
        tokenizer (`transformers.PreTrainedTokenizerBase`):
            The tokenizer to use for training. This argument is required if you want
to use the default data collator.
        model_init (`Callable[[], transformers.PreTrainedModel]`):
            The model initializer to use for training. If None is specified, the default
model initializer will be used.
        callbacks (`List[transformers.TrainerCallback]`):
            The callbacks to use for training.
        optimizers (`Tuple[torch.optim.Optimizer, torch.optim.lr_scheduler.Lambd
aLR]`):
            The optimizer and scheduler to use for training.
        preprocess_logits_for_metrics (`Callable[[torch.Tensor, torch.Tensor], tor
ch.Tensor]`):
            The function to use to preprocess the logits before computing the metr
ics.
        peft_config (`Dict`, defaults to `None`):
```

```
        The PEFT configuration to use for training. If you pass a PEFT configur
ation, the model will be wrapped in a PEFT model.
    compute_metrics (`Callable[[EvalPrediction], Dict]`, *optional*):
        The function to use to compute the metrics. Must take a `EvalPredictio
n` and return
        a dictionary string to metric values.
    """

    _tag_names = ["trl", "dpo"]

    @_deprecate_arguments(
        version="1.0.0",
        deprecated_args=[
            "beta",
            "label_smoothing",
            "loss_type",
            "label_pad_token_id",
            "padding_value",
            "truncation_mode",
            "max_length",
            "max_prompt_length",
            "max_target_length",
            "is_encoder_decoder",
            "disable_dropout",
            "generate_during_eval",
            "precompute_ref_log_probs",
            "dataset_num_proc",
            "model_init_kwargs",
            "ref_model_init_kwargs",
            "model_adapter_name",
            "ref_adapter_name",
            "reference_free",
            "force_use_ref_model",
        ],
        custom_message="Deprecated positional argument(s) used in DPOTrain
er, please use the DPOConfig to set these arguments instead.",
```

```
    )
    def __init__(
        self,
        model: Optional[Union[PreTrainedModel, nn.Module, str]] = None,
        ref_model: Optional[Union[PreTrainedModel, nn.Module, str]] = None,
        beta: float = 0.1,
        label_smoothing: float = 0,
        loss_type: str = "hinge",
        args: Optional[DPOConfig] = None,
        data_collator: Optional[DataCollator] = None,
        label_pad_token_id: int = -100,
        padding_value: Optional[int] = None,
        truncation_mode: str = "keep_end",
        train_dataset: Optional[Dataset] = None,
        eval_dataset: Optional[Union[Dataset, Dict[str, Dataset]]] = None,
        tokenizer: Optional[PreTrainedTokenizerBase] = None,
        model_init: Optional[Callable[[], PreTrainedModel]] = None,
        callbacks: Optional[List[TrainerCallback]] = None,
        optimizers: Tuple[torch.optim.Optimizer, torch.optim.lr_scheduler.Lambda
LR] = (None, None),
        preprocess_logits_for_metrics: Optional[Callable[[torch.Tensor, torch.Ten
sor], torch.Tensor]] = None,
        max_length: Optional[int] = None,
        max_prompt_length: Optional[int] = None,
        max_target_length: Optional[int] = None,
        peft_config: Optional[Dict] = None,
        is_encoder_decoder: Optional[bool] = None,
        disable_dropout: bool = True,
        generate_during_eval: bool = False,
        compute_metrics: Optional[Callable[[EvalLoopOutput], Dict]] = None,
        precompute_ref_log_probs: bool = False,
        dataset_num_proc: Optional[int] = None,
        model_init_kwargs: Optional[Dict] = None,
        ref_model_init_kwargs: Optional[Dict] = None,
        model_adapter_name: Optional[str] = None,
        ref_adapter_name: Optional[str] = None,
```

```
        reference_free: bool = False,
        force_use_ref_model: bool = False,
    ):
        if model_init_kwargs is not None:
```

DistiLLMTrainer 클래스의 초기화 부분입니다. distillm_trainer.py:343-434

```
 if self.is_vision_model:
            self.processor = tokenizer
            self.tokenizer = tokenizer.tokenizer  # tokenizer is actually a processor
at this point
            # elif hasattr(tokenizer, "_tokenizer"): self.tokenizer = tokenizer._tokeni
zer
            # else: raise NotImplementedError
        else:
            self.tokenizer = tokenizer

        self.is_peft_model = is_peft_available() and isinstance(model, PeftModel)
        if model_adapter_name is not None:
            warnings.warn(
                "You passed `model_adapter_name` to the DPOTrainer, the value yo
u passed will override the one in the `DPOConfig`."
            )
            args.model_adapter_name = model_adapter_name
        self.model_adapter_name = args.model_adapter_name

        if ref_adapter_name is not None:
            warnings.warn(
                "You passed `ref_adapter_name` to the DPOTrainer, the value you pa
ssed will override the one in the `DPOConfig`."
            )
            args.ref_adapter_name = ref_adapter_name
        self.ref_adapter_name = args.ref_adapter_name

        if reference_free:
            warnings.warn(
```

```
                "You passed `reference_free` to the DPOTrainer, the value you passe
d will override the one in the `DPOConfig`."
            )
            args.reference_free = reference_free
        self.reference_free = args.reference_free

        if precompute_ref_log_probs:
            warnings.warn(
                "You passed `precompute_ref_log_probs` to the DPOTrainer, the valu
e you passed will override the one in the `DPOConfig`."
            )
            args.precompute_ref_log_probs = precompute_ref_log_probs

        if ref_model:
            self.ref_model = ref_model
        elif self.is_peft_model or args.precompute_ref_log_probs:
            # The `model` with adapters turned off will be used as the reference m
odel
            self.ref_model = None
        else:
            self.ref_model = create_reference_model(model)

        if tokenizer is None:
            raise ValueError("tokenizer must be specified to tokenize a DPO datase
t.")

        if max_length is not None:
            warnings.warn(
                "You passed `max_length` to the DPOTrainer, the value you passed
will override the one in the `DPOConfig`."
            )
            args.max_length = max_length
        if args.max_length is None:
            warnings.warn(
                "`max_length` is not set in the DPOConfig's init"
                " it will default to `512` by default, but you should do it yourself in the
```

```
future.",
                UserWarning,
            )
            args.max_length = 512

        if max_prompt_length is not None:
            warnings.warn(
                "You passed `max_prompt_length` to the DPOTrainer, the value you
passed will override the one in the `DPOConfig`."
            )
            args.max_prompt_length = max_prompt_length
        if args.max_prompt_length is None:
            warnings.warn(
                "`max_prompt_length` is not set in the DPOConfig's init"
                " it will default to `128` by default, but you should do it yourself in the
future.",
                UserWarning,
            )
            args.max_prompt_length = 128

        if max_target_length is not None:
            warnings.warn(
                "You passed `max_target_length` to the DPOTrainer, the value you p
assed will override the one in the `DPOConfig`."
            )
            args.max_target_length = max_target_length
        if args.max_target_length is None and self.is_encoder_decoder:
            warnings.warn(
                "When using an encoder decoder architecture, you should set `max_
target_length` in the DPOConfig's init"
                " it will default to `128` by default, but you should do it yourself in the
future.",
                UserWarning,
            )
            args.max_target_length = 128
```

```
        if label_pad_token_id != -100:
            warnings.warn(
                "You passed `label_pad_token_id` to the DPOTrainer, the value you p
assed will override the one in the `DPOConfig`."
            )
            args.label_pad_token_id = label_pad_token_id
        if data_collator is None:
```

토크나이저 설정 및 모델 구성 부분입니다. distillm_trainer.py:464-655

```
self.max_length = args.max_length
        self.generate_during_eval = args.generate_during_eval
        self.label_pad_token_id = args.label_pad_token_id
        if padding_value is not None:
            warnings.warn(
                "You passed `padding_value` to the DPOTrainer, the value you passe
d will override the one in the `DPOConfig`."
            )
            args.padding_value = padding_value
        self.padding_value = args.padding_value if padding_value is not None els
e self.tokenizer.pad_token_id
        self.max_prompt_length = args.max_prompt_length
        if truncation_mode != "keep_end":
            warnings.warn(
                "You passed `truncation_mode` to the DPOTrainer, the value you pas
sed will override the one in the `DPOConfig`."
            )
            args.truncation_mode = truncation_mode
        self.truncation_mode = args.truncation_mode
        self.max_target_length = args.max_target_length
        self.precompute_ref_log_probs = args.precompute_ref_log_probs

        # Since ref_logs are precomputed on the first call to get_train/eval_datalo
ader
        # keep track of first called to avoid computation of future calls
        self._precomputed_train_ref_log_probs = False
```

```
        self._precomputed_eval_ref_log_probs = False

        if loss_type != "sigmoid":
            warnings.warn(
                "You passed `loss_type` to the DPOTrainer, the value you passed will
override the one in the `DPOConfig`."
            )
            args.loss_type = loss_type
        if label_smoothing != 0:
            warnings.warn(
                "You passed `label_smoothing` to the DPOTrainer, the value you pas
sed will override the one in the `DPOConfig`."
            )
            args.label_smoothing = label_smoothing
        if beta != 0.1:
            warnings.warn(
                "You passed `beta` to the DPOTrainer, the value you passed will over
ride the one in the `DPOConfig`."
            )
            args.beta = beta
        self.beta = args.beta
        self.label_smoothing = args.label_smoothing
        self.loss_type = args.loss_type
        self.base_alpha_1, self.base_alpha_2 = 0.1, 0.1
        self.update_alpha = False
        self.gradual_beta = False
        self.logp_logq, self.logq_logp = None, None

        self._stored_metrics = defaultdict(lambda: defaultdict(list))

        if dataset_num_proc is not None:
            warnings.warn(
                "You passed `dataset_num_proc` to the DPOTrainer, the value you p
assed will override the one in the `DPOConfig`."
            )
            args.dataset_num_proc = dataset_num_proc
```

```
        self.dataset_num_proc = args.dataset_num_proc

        # Compute that only on the main process for faster data processing.
        # see: https://github.com/huggingface/trl/pull/1255
        with PartialState().local_main_process_first():
            # tokenize the dataset, lower writer batch size to avoid OOM (frequent
in vision models)
            train_dataset = train_dataset.map(self.tokenize_row, num_proc=self.dat
aset_num_proc, writer_batch_size=10)
            if eval_dataset is not None:
                eval_dataset = eval_dataset.map(
                    self.tokenize_row, num_proc=self.dataset_num_proc, writer_batch
_size=10
                )
        super().__init__(
            model=model,
            args=args,
            data_collator=data_collator,
            train_dataset=train_dataset,
            eval_dataset=eval_dataset,
            tokenizer=tokenizer,
            model_init=model_init,
            compute_metrics=compute_metrics,
            callbacks=callbacks,
            optimizers=optimizers,
            preprocess_logits_for_metrics=preprocess_logits_for_metrics,
        )

        # Add tags for models that have been loaded with the correct transforme
rs version
        if hasattr(self.model, "add_model_tags"):
            self.model.add_model_tags(self._tag_names)

        if not hasattr(self, "accelerator"):
            raise AttributeError(
                "Your `Trainer` does not have an `accelerator` object. Consider upgr
```

```
ading `transformers`."
            )

        # Deepspeed Zero-3 does not support precompute_ref_log_probs
        if self.is_deepspeed_enabled:
            if self.accelerator.state.deepspeed_plugin.zero_stage == 3 and self.pre
compute_ref_log_probs:
                raise ValueError(
                    "You cannot use `precompute_ref_log_probs=True` with Deepspe
ed ZeRO-3. Please set `precompute_ref_log_probs=False`."
                )

        if self.ref_model is None:
            if not (self.is_peft_model or self.precompute_ref_log_probs):
                raise ValueError(
                    "No reference model and model is not a Peft model. Try setting `pr
ecompute_ref_log_probs=True`"
                )
            if args.sync_ref_model:
                raise ValueError(
                    "You currently cannot use `ref_model=None` with TR-DPO metho
d. Please provide `ref_model`."
                )
        else:
            if self.is_deepspeed_enabled:
                self.ref_model = self._prepare_deepspeed(self.ref_model)
            else:
                self.ref_model = self.accelerator.prepare_model(self.ref_model, eval
uation_mode=True)

        if args.sync_ref_model:
            if precompute_ref_log_probs:
                raise ValueError(
                    "You cannot use `precompute_ref_log_probs=True` with TR-DPO
method. Please set `precompute_ref_log_probs=False`."
                )
```

```
            self.add_callback(SyncRefModelCallback(ref_model=self.ref_model, accelerator=self.accelerator))

    def _prepare_deepspeed(self, model: PreTrainedModelWrapper):
        # Adapted from accelerate: https://github.com/huggingface/accelerate/blob/739b135f8367becb67ffaada12fe76e3aa60fefd/src/accelerate/accelerator.py#L1473
        deepspeed_plugin = self.accelerator.state.deepspeed_plugin
        config_kwargs = deepcopy(deepspeed_plugin.deepspeed_config)

        if model is not None:
            if hasattr(model, "config"):
                hidden_size = (
                    max(model.config.hidden_sizes)
                    if getattr(model.config, "hidden_sizes", None)
                    else getattr(model.config, "hidden_size", None)
                )
                if hidden_size is not None and config_kwargs["zero_optimization"]["stage"] == 3:
                    # Note that `stage3_prefetch_bucket_size` can produce DeepSpeed messages like: `Invalidate trace cache @ step 0: expected module 1, but got module 0`
                    # This is expected and is not an error, see: https://github.com/microsoft/DeepSpeed/discussions/4081
                    config_kwargs.update(
                        {
                            "zero_optimization.reduce_bucket_size": hidden_size * hidden_size,
                            "zero_optimization.stage3_param_persistence_threshold": 10 * hidden_size,
                            "zero_optimization.stage3_prefetch_bucket_size": 0.9 * hidden_size * hidden_size,
                        }
                    )
```

```
        # If ZeRO-3 is used, we shard both the active and reference model.
        # Otherwise, we assume the reference model fits in memory and is initiali
zed on each device with ZeRO disabled (stage 0)
        if config_kwargs["zero_optimization"]["stage"] != 3:
            config_kwargs["zero_optimization"]["stage"] = 0
        model, *_ = deepspeed.initialize(model=model, config=config_kwargs)
        model.eval()
        return model

    def get_train_dataloader(self) -> DataLoader:
        """
        Returns the training [`~torch.utils.data.DataLoader`].

        Subclass of transformers.src.transformers.trainer.get_train_dataloader to
precompute `ref_log_probs`.
        """

        if self.precompute_ref_log_probs and not self._precomputed_train_ref_log
_probs:
            dataloader_params = {
                "batch_size": self.args.per_device_train_batch_size,
                "collate_fn": self.data_collator,
                "num_workers": self.args.dataloader_num_workers,
                "pin_memory": self.args.dataloader_pin_memory,
                "shuffle": False,
            }

            # prepare dataloader
            data_loader = self.accelerator.prepare(DataLoader(self.train_dataset, *
*dataloader_params))

            reference_chosen_logps = []
            reference_rejected_logps = []
            for padded_batch in tqdm(iterable=data_loader, desc="Train dataset re
ference log probs"):
                reference_chosen_logp, reference_rejected_logp = self.compute_ref
```

```
erence_log_probs(padded_batch)
                reference_chosen_logp, reference_rejected_logp = self.accelerator.g
ather_for_metrics(
                    (reference_chosen_logp, reference_rejected_logp)
                )
                reference_chosen_logps.append(reference_chosen_logp.cpu())
                reference_rejected_logps.append(reference_rejected_logp.cpu())

            all_reference_chosen_logps = torch.cat(reference_chosen_logps).float
().numpy()
            all_reference_rejected_logps = torch.cat(reference_rejected_logps).flo
at().numpy()

            self.train_dataset = self.train_dataset.add_column(
                name="reference_chosen_logps", column=all_reference_chosen_log
ps
            )
            self.train_dataset = self.train_dataset.add_column(
                name="reference_rejected_logps", column=all_reference_rejected_l
ogps
            )

            self._precomputed_train_ref_log_probs = True

        return super().get_train_dataloader()
```

훈련 설정 및 데이터 전처리 부분입니다. distillm_trainer.py:708-772

```
def build_tokenized_answer(self, prompt, answer, images=None):
        """
        Llama tokenizer does satisfy `enc(a + b) = enc(a) + enc(b)`.
        It does ensure `enc(a + b) = enc(a) + enc(a + b)[len(enc(a)):]`.
        Reference:
            https://github.com/EleutherAI/lm-evaluation-harness/pull/531#issueco
mment-1595586257
        """
```

```
        if self.is_vision_model:
            if answer.count("<image>") > 0:
                raise NotImplementedError("Answer contains <image> token, which is not supported yet.")
            full_tokenized = self.processor(prompt + answer, images, add_special_tokens=False)
            full_tokenized = {k: v[0] for k, v in full_tokenized.items()}  # Unbatch, not done when using idefics
            prompt_input_ids = self.processor(prompt, images, add_special_tokens=False)["input_ids"][0]
        else:
            full_tokenized = self.tokenizer(prompt + answer, add_special_tokens=False)
            prompt_input_ids = self.tokenizer(prompt, add_special_tokens=False)["input_ids"]

        answer_input_ids = full_tokenized["input_ids"][len(prompt_input_ids) :]
        answer_attention_mask = full_tokenized["attention_mask"][len(prompt_input_ids) :]

        # Concat tokens to form `enc(a) + enc(a + b)[len(enc(a)):]`
        full_concat_input_ids = np.concatenate([prompt_input_ids, answer_input_ids])

        # Prepare input tokens for token by token comparison
        full_input_ids = np.array(full_tokenized["input_ids"])

        if len(full_input_ids) != len(full_concat_input_ids):
            raise ValueError("Prompt input ids and answer input ids should have the same length.")

        # On some tokenizers, like Llama-2 tokenizer, there are occasions where tokens
        # can be merged together when tokenizing prompt+answer. This could result
        # on the last token from the prompt being different when tokenized on its
```

```
own
        # vs when done as prompt+answer.
        response_token_ids_start_idx = len(prompt_input_ids)

        # If tokenized prompt is different than both prompt+answer, then it mean
s the
        # last token has changed due to merging.
        if prompt_input_ids != full_tokenized["input_ids"][:response_token_ids_st
art_idx]:
            response_token_ids_start_idx -= 1

        prompt_input_ids = full_tokenized["input_ids"][:response_token_ids_start
_idx]
        prompt_attention_mask = full_tokenized["attention_mask"][:response_tok
en_ids_start_idx]

        if len(prompt_input_ids) != len(prompt_attention_mask):
            raise ValueError("Prompt input ids and attention mask should have the
same length.")

        answer_input_ids = full_tokenized["input_ids"][response_token_ids_start_
idx:]
        answer_attention_mask = full_tokenized["attention_mask"][response_toke
n_ids_start_idx:]

        if "pixel_values" in full_tokenized:
            return dict(
                prompt_input_ids=prompt_input_ids,
                prompt_attention_mask=prompt_attention_mask,
                prompt_pixel_values=full_tokenized["pixel_values"],
                input_ids=answer_input_ids,
                attention_mask=answer_attention_mask,
            )
        else:
            return dict(
                prompt_input_ids=prompt_input_ids,
```

```
            prompt_attention_mask=prompt_attention_mask,
            input_ids=answer_input_ids,
            attention_mask=answer_attention_mask,
        )
```

토크나이제이션 처리를 위한 `build_tokenized_answer` 메서드입니다. distillm_trainer.py:1143-1460

```
 tea_per_token_logps = torch.gather(tea_vocab_logps, dim=2, index=labels.unsqueeze(2)).squeeze(2)

        if "distillm" in loss_type:
            try:
                assert loss_type == "distillm_v2" and logp_logq is not None
                anchor = (1-base_alpha_1) * logp_logq
                logps_logqs = ((tea_per_token_logps * loss_mask).sum(-1) / loss_mask.sum(-1)).exp() - \
                    ((per_token_logps * loss_mask).sum(-1) / loss_mask.sum(-1)).exp() # sentence-level
                alpha_1 = torch.clip(1 - anchor / (logps_logqs+1e-5), min=1e-2, max=base_alpha_1).unsqueeze(-1).unsqueeze(-1)
            except:
                alpha_1 = base_alpha_1

            try:
                if isinstance(alpha_1, torch.Tensor):
                    log_alpha_1, log_one_minus_alpha_1 = torch.log(alpha_1), torch.log(1-alpha_1)
                else:
                    log_alpha_1, log_one_minus_alpha_1 = math.log(alpha_1), math.log(1-alpha_1)
                mix_vocab_logps = torch.logsumexp(
                    torch.stack([
                        log_alpha_1 + tea_vocab_logps, log_one_minus_alpha_1 + vocab_logps
                    ], dim=0), dim=0
```

```
            )
            tea_pos_kl = (tea_vocab_logps.exp() * (tea_vocab_logps - mix_vocab
_logps)).sum(-1)
        except torch.OutOfMemoryError:
            torch.cuda.empty_cache()
            if isinstance(alpha_1, torch.Tensor):
                log_alpha_1, log_one_minus_alpha_1 = torch.log(alpha_1), torch.log
(1-alpha_1)
            else:
                log_alpha_1, log_one_minus_alpha_1 = math.log(alpha_1), math.log
(1-alpha_1)
            mix_vocab_logps = torch.logsumexp(
                torch.stack([
                    log_alpha_1 + tea_vocab_logps, log_one_minus_alpha_1 + vocab
_logps
                ], dim=0), dim=0
            )
            tea_pos_kl = (tea_vocab_logps.exp() * (tea_vocab_logps - mix_vocab
_logps)).sum(-1)
        del mix_vocab_logps

        try:
            assert loss_type == "distillm_v2" and logq_logp is not None
            anchor = (1-base_alpha_2) * logq_logp
            logqs_logps = ((per_token_logps * loss_mask).sum(-1) / loss_mask.s
um(-1)).exp() - \
                ((tea_per_token_logps * loss_mask).sum(-1) / loss_mask.sum(-1)).
exp() # sentence-level
            alpha_2 = torch.clip(1 - anchor / (logqs_logps+1e-5), min=1e-2, max
=base_alpha_2).unsqueeze(-1).unsqueeze(-1)
        except:
            alpha_2 = base_alpha_2

        try:
            if isinstance(alpha_2, torch.Tensor):
                log_alpha_2, log_one_minus_alpha_2 = torch.log(alpha_2), torch.lo
```

```
g(1-alpha_2)
            else:
                log_alpha_2, log_one_minus_alpha_2 = math.log(alpha_2), math.lo
g(1-alpha_2)
            mix_vocab_logps = torch.logsumexp(
                torch.stack([
                    log_one_minus_alpha_2 + tea_vocab_logps, log_alpha_2 + voca
b_logps.detach()
                ], dim=0), dim=0
            )
            ref_pos_kl = (vocab_logps.exp() * (vocab_logps - mix_vocab_logp
s)).sum(-1)
        except torch.OutOfMemoryError:
            torch.cuda.empty_cache()
            if isinstance(alpha_2, torch.Tensor):
                log_alpha_2, log_one_minus_alpha_2 = torch.log(alpha_2), torch.lo
g(1-alpha_2)
            else:
                log_alpha_2, log_one_minus_alpha_2 = math.log(alpha_2), math.lo
g(1-alpha_2)
            mix_vocab_logps = torch.logsumexp(
                torch.stack([
                    log_one_minus_alpha_2 + tea_vocab_logps, log_alpha_2 + voca
b_logps.detach()
                ], dim=0), dim=0
            )
            ref_pos_kl = (vocab_logps.exp() * (vocab_logps - mix_vocab_logp
s)).sum(-1)
        del mix_vocab_logps; del tea_vocab_logps

    elif loss_type == "gkd":
        alpha = 0.9
        mix_vocab_logps = torch.logsumexp(
            torch.stack([
                math.log(alpha) + tea_vocab_logps, math.log(1-alpha) + vocab_log
ps
```

```
            ], dim=0), dim=0
        )
        tea_pos_kl = alpha * (tea_vocab_logps.exp() * (tea_vocab_logps - mix_
vocab_logps)).sum(-1) + \
            (1-alpha) * (vocab_logps.exp() * (vocab_logps - mix_vocab_logps)).s
um(-1) # jensen-shannon distance
        ref_pos_kl = alpha * (tea_vocab_logps.exp() * (tea_vocab_logps - mix_v
ocab_logps)).sum(-1) + \
            (1-alpha) * (vocab_logps.exp() * (vocab_logps - mix_vocab_logps)).s
um(-1) # jensen-shannon distance

    else:
        tea_pos_kl = (tea_vocab_logps.exp() * (tea_vocab_logps - vocab_logp
s)).sum(-1) # forward kl
        ref_pos_kl = (vocab_logps.exp() * (tea_vocab_logps - vocab_logps)).su
m(-1) # reverse kl

    if average_log_prob:
        result = (
            (per_token_logps * loss_mask).sum(-1) / loss_mask.sum(-1), (tea_per
_token_logps * loss_mask).sum(-1) / loss_mask.sum(-1),
            (tea_pos_kl * loss_mask).sum(-1) / loss_mask.sum(-1), (ref_pos_kl * l
oss_mask).sum(-1) / loss_mask.sum(-1) if ref_pos_kl is not None else None
        )
    else:
        result = (
            (per_token_logps * loss_mask).sum(-1), (tea_per_token_logps * loss_
mask).sum(-1),
            (tea_pos_kl * loss_mask).sum(-1), (ref_pos_kl * loss_mask).sum(-1) if
ref_pos_kl is not None else None
        )

    del per_token_logps; del tea_per_token_logps; del tea_pos_kl; del ref_pos
_kl
    return result
```

```
    def concatenated_forward(
        self, model: nn.Module, ref_model: nn.Module, batch: Dict[str, Union[List, torch.LongTensor]]
    ) -> Tuple[torch.FloatTensor, torch.FloatTensor, torch.FloatTensor, torch.FloatTensor, torch.FloatTensor]:
        """Run the given model on the given batch of inputs, concatenating the chosen and rejected inputs together.

        We do this to avoid doing two forward passes, because it's faster for FSDP.
        """
        concatenated_batch = self.concatenated_inputs(
            batch,
            is_encoder_decoder=self.is_encoder_decoder,
            is_vision_model=self.is_vision_model,
            label_pad_token_id=self.label_pad_token_id,
            padding_value=self.padding_value,
            device=self.accelerator.device,
        )
        len_chosen = batch["chosen_labels"].shape[0]

        model_kwargs = {}

        if self.is_encoder_decoder:
            model_kwargs["labels"] = concatenated_batch["concatenated_labels"]
            model_kwargs["decoder_input_ids"] = concatenated_batch.pop("concatenated_decoder_input_ids", None)

        if self.is_vision_model:
            model_kwargs["pixel_values"] = concatenated_batch["pixel_values"]

        all_logits = model(
            concatenated_batch["concatenated_input_ids"],
            attention_mask=concatenated_batch["concatenated_attention_mask"],
            use_cache=False,
            **model_kwargs,
```

```
        ).logits

        if all_logits.shape[:2] != concatenated_batch["concatenated_labels"].sha
pe[:2]:
            # for llava, the model returns logits for the entire sequence, including t
he image tokens (placed before the text tokens)
            seq_len = concatenated_batch["concatenated_labels"].shape[1]
            all_logits = all_logits[:, -seq_len:]

        with torch.no_grad():
            tea_all_logits = ref_model(
                concatenated_batch["concatenated_input_ids"],
                attention_mask=concatenated_batch["concatenated_attention_mas
k"],
                use_cache=False,
                **model_kwargs,
            ).logits

            if tea_all_logits.shape[:2] != concatenated_batch["concatenated_label
s"].shape[:2]:
                # for llava, the model returns logits for the entire sequence, includin
g the image tokens (placed before the text tokens)
                seq_len = concatenated_batch["concatenated_labels"].shape[1]
                tea_all_logits = tea_all_logits[:, -seq_len:]

        all_logps, tea_all_logps, tea_pos_kl, ref_pos_kl = self.get_batch_logps(
            all_logits, tea_all_logits,
            concatenated_batch["concatenated_labels"],
            average_log_prob=True,
            is_encoder_decoder=self.is_encoder_decoder,
            label_pad_token_id=self.label_pad_token_id,
            loss_type=self.loss_type,
            logp_logq=self.logp_logq, logq_logp=self.logq_logp
        )
        del tea_all_logits
```

```
        chosen_logps, tea_chosen_logps = all_logps[:len_chosen], tea_all_logps[:
len_chosen]
        rejected_logps, tea_rejected_logps = all_logps[len_chosen:], tea_all_logps
[len_chosen:]

        if self.loss_type == "distillm_v1":
            chosen_pos_kl = ref_pos_kl[:len_chosen]
        else:
            chosen_pos_kl = tea_pos_kl[:len_chosen]
        if ref_pos_kl is not None:
            rejected_pos_kl = ref_pos_kl[len_chosen:]
        else:
            rejected_pos_kl = tea_pos_kl[len_chosen:]

        return chosen_logps, rejected_logps, tea_chosen_logps, tea_rejected_log
ps, chosen_pos_kl, rejected_pos_kl

    def get_batch_loss_metrics(
        self,
        model,
        batch: Dict[str, Union[List, torch.LongTensor]],
        train_eval: Literal["train", "eval"] = "train",
    ):
        """Compute the DPO loss and other metrics for the given batch of inputs
for train or test."""
        metrics = {}

        forward_output = self.concatenated_forward(model, self.ref_model, batc
h)
        (
            policy_chosen_logps, policy_rejected_logps, reference_chosen_logps, r
eference_rejected_logps,
            chosen_position_kl, rejected_position_kl, *_
        ) = forward_output[:7]

        losses = self.kd_loss(
```

```
            policy_chosen_logps, policy_rejected_logps,
            reference_chosen_logps, reference_rejected_logps,
            chosen_position_kl, rejected_position_kl
        )

        prefix = "eval_" if train_eval == "eval" else ""
        metrics[f"{prefix}logqs/rejected"] = policy_rejected_logps.detach().mean
().cpu()
        metrics[f"{prefix}logqs/chosen"] = policy_chosen_logps.detach().mean().
cpu()
        metrics[f"{prefix}logqs_logps/rejected"] = (policy_rejected_logps.exp() -
reference_rejected_logps.exp()).detach().mean().cpu()
        metrics[f"{prefix}logps_logqs/chosen"] = (reference_chosen_logps.exp()
- policy_chosen_logps.exp()).detach().mean().cpu()

        return losses.mean(), metrics

    def compute_loss(
        self,
        model: Union[PreTrainedModel, nn.Module],
        inputs: Dict[str, Union[torch.Tensor, Any]],
        return_outputs=False,
    ) -> Union[torch.Tensor, Tuple[torch.Tensor, Dict[str, torch.Tensor]]]:
        if not self.use_dpo_data_collator:
            warnings.warn(
                "compute_loss is only implemented for DPODataCollatorWithPaddin
g, and you passed a datacollator that is different than "
                "DPODataCollatorWithPadding - you might see unexpected behavio
r. Alternatively, you can implement your own prediction_step method if you ar
e using a custom data collator"
            )

        compute_loss_context_manager = torch.cuda.amp.autocast if self._peft_h
as_been_casted_to_bf16 else nullcontext

        with compute_loss_context_manager():
```

```
            loss, metrics = self.get_batch_loss_metrics(model, inputs, train_eval="t
rain")

        # Make sure to move the loss to the device the original accumulating loss
is at back in the `Trainer` class:
        loss = loss.to(self.args.device)
        # force log the metrics
        self.store_metrics(metrics, train_eval="train")

        if return_outputs:
            return (loss, metrics)
        return loss

    def get_batch_samples(self, model, batch: Dict[str, torch.LongTensor]) -> T
uple[str, str]:
        """Generate samples from the model and reference model for the given b
atch of inputs."""

        # If one uses `generate_during_eval` with peft + bf16, we need to explicitl
y call generate with
        # the torch cuda amp context manager as some hidden states are silently
casted to full precision.
        generate_context_manager = nullcontext if not self._peft_has_been_caste
d_to_bf16 else torch.cuda.amp.autocast

        with generate_context_manager():
            policy_output = model.generate(
                input_ids=batch["prompt_input_ids"],
                attention_mask=batch["prompt_attention_mask"],
                max_length=self.max_length,
                do_sample=True,
                pad_token_id=self.tokenizer.pad_token_id,
            )

            # if reference_output in batch use that otherwise use the reference mo
del
```

```
            if "reference_output" in batch:
                reference_output = batch["reference_output"]
            else:
                if self.ref_model is None:
                    with self.null_ref_context():
                        reference_output = self.model.generate(
                            input_ids=batch["prompt_input_ids"],
                            attention_mask=batch["prompt_attention_mask"],
                            max_length=self.max_length,
                            do_sample=True,
                            pad_token_id=self.tokenizer.pad_token_id,
                        )
                else:
                    reference_output = self.ref_model.generate(
                        input_ids=batch["prompt_input_ids"],
                        attention_mask=batch["prompt_attention_mask"],
                        max_length=self.max_length,
                        do_sample=True,
                        pad_token_id=self.tokenizer.pad_token_id,
                    )

            policy_output = pad_to_length(policy_output, self.max_length, self.tokenizer.pad_token_id)
            policy_output_decoded = self.tokenizer.batch_decode(policy_output, skip_special_tokens=True)

            reference_output = pad_to_length(reference_output, self.max_length, self.tokenizer.pad_token_id)
            reference_output_decoded = self.tokenizer.batch_decode(reference_output, skip_special_tokens=True)

            return policy_output_decoded, reference_output_decoded

        def prediction_step(
            self,
            model: Union[PreTrainedModel, nn.Module],
```

```
        inputs: Dict[str, Union[torch.Tensor, Any]],
        prediction_loss_only: bool,
        ignore_keys: Optional[List[str]] = None,
    ):
        if not self.use_dpo_data_collator:
            warnings.warn(
                "prediction_step is only implemented for DPODataCollatorWithPaddi
ng, and you passed a datacollator that is different than "
                "DPODataCollatorWithPadding - you might see unexpected behavio
r. Alternatively, you can implement your own prediction_step method if you ar
e using a custom data collator"
            )
        if ignore_keys is None:
            if hasattr(model, "config"):
                ignore_keys = getattr(model.config, "keys_to_ignore_at_inference",
[])
            else:
                ignore_keys = []

        prediction_context_manager = torch.cuda.amp.autocast if self._peft_has_
been_casted_to_bf16 else nullcontext

        with torch.no_grad(), prediction_context_manager():
            loss, metrics = self.get_batch_loss_metrics(model, inputs, train_eval
="eval")

        # force log the metrics
        self.store_metrics(metrics, train_eval="eval")

        if prediction_loss_only:
            return (loss.detach(), None, None)

        # logits for the chosen and rejected samples from model
        logits_dict = {
            "eval_logits/chosen": metrics["eval_logits/chosen"],
            "eval_logits/rejected": metrics["eval_logits/rejected"],
```

```
        }
        logits = tuple(v.unsqueeze(dim=0) for k, v in logits_dict.items() if k not in i
gnore_keys)
        logits = torch.stack(logits).mean(axis=1).to(self.accelerator.device)
        labels = torch.zeros(logits.shape[0], device=self.accelerator.device)

        return (loss.detach(), logits, labels)
```

대조적 증류의 핵심인 `get_batch_logps` 메서드와 distillm_v2 손실 함수 구현입니다. distillm_trainer.py:1515-1545

```
 def log(self, logs: Dict[str, float]) → None:
        """
        Log `logs` on the various objects watching training, including stored metri
cs.

        Args:
            logs (`Dict[str, float]`):
                The values to log.
        """
        # logs either has 'loss' or 'eval_loss'
        train_eval = "train" if "loss" in logs else "eval"
        # Add averaged stored metrics to logs
        for key, metrics in self._stored_metrics[train_eval].items():
            logs[key] = torch.tensor(metrics).mean().item()
        if self.update_alpha:
            if self.logp_logq is None:
                self.logp_logq = self._stored_metrics['logps_logqs/chosen']
            if self.logq_logp is None:
                self.logq_logp = self._stored_metrics['logqs_logps/rejected']
        del self._stored_metrics[train_eval]
        return super().log(logs)

    @wraps(Trainer.push_to_hub)
    def push_to_hub(self, commit_message: Optional[str] = "End of training", bl
ocking: bool = True, **kwargs) → str:
```

```
        """
        Overwrite the `push_to_hub` method in order to force-add the tag "dpo" when pushing the
        model on the Hub. Please refer to `~transformers.Trainer.push_to_hub` for more details.
        """
        kwargs = trl_sanitze_kwargs_for_tagging(model=self.model, tag_names=self._tag_names, kwargs=kwargs)

        return super().push_to_hub(commit_message=commit_message, blocking=blocking, **kwargs)
```

로깅 및 메트릭 관리 부분입니다.

5. LoRA 어댑터 병합 merging.py:1-26

훈련된 LoRA 어댑터를 베이스 모델에 병합하는 유틸리티입니다.

```
from transformers import AutoModelForCausalLM, AutoTokenizer, AutoModelForVision2Seq, AutoProcessor
from peft import PeftModel
import os
import argparse

def main(args):
    try:
        base_model = AutoModelForCausalLM.from_pretrained(args.base_model_name) # LLMs
        tokenizer = AutoTokenizer.from_pretrained(args.base_model_name) # LLMs
    except:
        base_model = AutoModelForVision2Seq.from_pretrained(args.base_model_name) # VLMs
        tokenizer = AutoProcessor.from_pretrained(args.base_model_name) # VLMs
```

```
    lora_model = PeftModel.from_pretrained(base_model, args.lora_model_nam
e)
    merged_model = lora_model.merge_and_unload()
    save_path = f"{args.lora_model_name}/merged"
    merged_model.save_pretrained(save_path); tokenizer.save_pretrained(save
_path)

if __name__ == "__main__":
    # For vllm 0.5.4, inference with the LoRA model is less accurate than with th
e merged model.
    parser = argparse.ArgumentParser(description="Merging the Base and LoR
A models for accurate inference")
    parser.add_argument('--base-model-name', type=str, required=True)
    parser.add_argument('--lora-model-name', type=str, required=True)
    args = parser.parse_args()
    main(args)
```