

DeepSeek R1 데이터 입력 방법 및 절차

1. 데이터 전처리 단계

1.1 원본 데이터 정리 및 구조화

수집한 법률 서류 데이터는 먼저 DeepSeek R1이 이해할 수 있는 형태로 변환해야 합니다. 이 과정을 데이터 전처리라고 합니다.

A. 텍스트 정제 과정

법률 서류 텍스트를 정제하는 함수

```
def clean_legal_document(raw_text):
```

```
    # 1단계: 불필요한 특수문자 제거
```

```
    cleaned_text = raw_text.replace("\n\n", '\n') # 빈 줄 정리
```

```
    cleaned_text = re.sub(r'[^\w\s가-할.,!?:.]', "", cleaned_text) # 특수문자 제거
```

```
    # 2단계: 법률 용어 표준화
```

```
    legal_terms = {
```

```
        '원고': 'PLAINTIFF',
```

```
        '피고': 'DEFENDANT',
```

```
        '법원': 'COURT',
```

```
        '판사': 'JUDGE'
```

```
    }
```

```
    for korean_term, standard_term in legal_terms.items():
```

```
        cleaned_text = cleaned_text.replace(korean_term, standard_term)
```

```
    # 3단계: 문장 단위로 분할
```

```
    sentences = cleaned_text.split('.')
```

```
    cleaned_sentences = [sent.strip() for sent in sentences if len(sent.strip()) > 5]
```

```
    return cleaned_sentences
```

B. 데이터 구조화 방식

법률 서류를 DeepSeek R1이 학습하기 쉬운 구조로 변환합니다:

법률 서류를 구조화된 형태로 변환

```
def structure_legal_document(document_text, document_type):
```

```
    structured_data = {
```

```
        "document_type": document_type, # 소장, 답변서, 준비서면 등
```

```
        "case_info": {
```

```
            "case_number": extract_case_number(document_text),
```

```
            "court": extract_court_name(document_text),
```

```

        "date": extract_date(document_text)
    },
    "parties": {
        "plaintiff": extract_plaintiff_info(document_text),
        "defendant": extract_defendant_info(document_text)
    },
    "claims": extract_claims(document_text),
    "facts": extract_facts(document_text),
    "legal_basis": extract_legal_basis(document_text),
    "prayer": extract_prayer(document_text)
}

return structured_data

```

1.2 학습 데이터 포맷 변환

DeepSeek R1은 대화형 데이터 형식을 선호하므로, 법률 서류를 질문-답변 형태로 변환합니다.

A. 질문-답변 쌍 생성

법률 서류를 질문-답변 형태로 변환

```
def create_qa_pairs(structured_document):
```

```
    qa_pairs = []
```

서류 작성 질문-답변 쌍

```
    document_generation_qa = {
```

```
        "instruction": f"{structured_document['document_type']}을 작성해주세요.",
```

```
        "input": f"""
```

```
        사건 정보: {structured_document['case_info']}
```

```
        당사자 정보: {structured_document['parties']}
```

```
        청구 내용: {structured_document['claims']}
```

```
        사실관계: {structured_document['facts']}
```

```
        """,
```

```
        "output": generate_full_document(structured_document)
```

```
    }
```

```
    qa_pairs.append(document_generation_qa)
```

서류 검토 질문-답변 쌍

```
    document_review_qa = {
```

```
        "instruction": "다음 법률 서류를 검토하고 문제점을 지적해주세요.",
```

```
        "input": generate_full_document(structured_document),
```

```
        "output": generate_review_comments(structured_document)
```

```
    }
```

```
    qa_pairs.append(document_review_qa)
```

```
    return qa_pairs
```

B. 프롬프트 템플릿 적용

DeepSeek R1 학습용 프롬프트 템플릿

LEGAL_DOCUMENT_TEMPLATE = """

<|user|>

당신은 대한민국의 전문 법무 담당자입니다.

다음 정보를 바탕으로 {document_type}을 작성해주세요.

사건 정보:

- 사건 번호: {case_number}
- 관할 법원: {court}
- 사건 유형: {case_type}

당사자 정보:

- 원고: {plaintiff_name} ({plaintiff_address})
- 피고: {defendant_name} ({defendant_address})

청구 내용:

{claims}

사실관계:

{facts}

법적 근거:

{legal_basis}

<|assistant|>

{generated_document}

"""

템플릿에 데이터 적용

def apply_template(structured_data):

```
    formatted_prompt = LEGAL_DOCUMENT_TEMPLATE.format(
        document_type=structured_data['document_type'],
        case_number=structured_data['case_info']['case_number'],
        court=structured_data['case_info']['court'],
        case_type=structured_data['case_info']['case_type'],
        plaintiff_name=structured_data['parties']['plaintiff']['name'],
        plaintiff_address=structured_data['parties']['plaintiff']['address'],
        defendant_name=structured_data['parties']['defendant']['name'],
        defendant_address=structured_data['parties']['defendant']['address'],
        claims=structured_data['claims'],
        facts=structured_data['facts'],
        legal_basis=structured_data['legal_basis'],
        generated_document=structured_data['output_document']
    )
```

```
    return formatted_prompt
```

1.3 데이터 품질 검증

학습 데이터의 품질을 확인하는 과정입니다:

데이터 품질 검증 함수

```
def validate_training_data(qa_pair):  
    validation_results = {  
        "completeness": check_completeness(qa_pair),  
        "accuracy": check_legal_accuracy(qa_pair),  
        "format_compliance": check_format_compliance(qa_pair),  
        "length_appropriate": check_length_limits(qa_pair)  
    }
```

모든 검증 항목이 통과해야 함

```
is_valid = all(validation_results.values())
```

```
return is_valid, validation_results
```

완전성 검사

```
def check_completeness(qa_pair):  
    required_fields = ['instruction', 'input', 'output']  
    return all(field in qa_pair and qa_pair[field] for field in required_fields)
```

법적 정확성 검사

```
def check_legal_accuracy(qa_pair):  
    legal_terms = ['원고', '피고', '법원', '사건번호']  
    output_text = qa_pair['output']
```

필수 법률 용어 포함 여부 확인

```
terms_present = sum(1 for term in legal_terms if term in output_text)  
return terms_present >= 3 # 최소 3개 이상 포함
```

형식 준수 검사

```
def check_format_compliance(qa_pair):  
    output_text = qa_pair['output']
```

법률 서류 기본 구조 확인

```
required_sections = ['사건번호', '당사자', '청구취지', '청구원인']  
sections_present = sum(1 for section in required_sections if section in output_text)
```

```
return sections_present >= 2 # 최소 2개 섹션 이상 포함
```

2. DeepSeek R1 모델 준비

2.1 모델 로드 및 설정

DeepSeek R1 모델을 컴퓨터에 불러오고 Fine-tuning 준비를 합니다:

```
# 필요한 라이브러리 가져오기
from transformers import AutoTokenizer, AutoModelForCausalLM
from peft import LoraConfig, get_peft_model, prepare_model_for_kbit_training
import torch

# DeepSeek R1 모델과 토큰나이저 로드
def load_deepseek_r1_model():
    model_name = "deepseek-ai/deepseek-r1"

    # 토큰나이저 로드 (텍스트를 숫자로 변환하는 도구)
    tokenizer = AutoTokenizer.from_pretrained(model_name)
    if tokenizer.pad_token is None:
        tokenizer.pad_token = tokenizer.eos_token

    # 모델 로드
    model = AutoModelForCausalLM.from_pretrained(
        model_name,
        torch_dtype=torch.float16, # 메모리 효율성을 위한 설정
        device_map="auto",        # GPU 자동 할당
        trust_remote_code=True
    )

    return model, tokenizer

# 모델 준비 함수
def prepare_model_for_training(model):
    # 모델을 학습 가능하도록 설정
    model = prepare_model_for_kbit_training(model)

    # LoRA 설정 (효율적인 Fine-tuning 방법)
    lora_config = LoraConfig(
        r=64,                # LoRA 랭크
        lora_alpha=16,        # LoRA 알파값
        target_modules=[      # 학습할 레이어 지정
            "q_proj", "v_proj", "k_proj", "o_proj",
            "gate_proj", "up_proj", "down_proj"
        ],
        lora_dropout=0.1,     # 드롭아웃 비율
        bias="none",          # 바이어스 설정
        task_type="CAUSAL_LM" # 태스크 유형
    )

    # LoRA 적용
    model = get_peft_model(model, lora_config)
```

```
return model
```

2.2 토큰나이저 설정

텍스트 데이터를 모델이 이해할 수 있는 숫자로 변환하는 과정입니다:

```
# 법률 서류 텍스트를 토큰으로 변환
```

```
def tokenize_legal_document(text, tokenizer, max_length=4096):
```

```
    # 텍스트를 토큰(숫자)으로 변환
```

```
    tokens = tokenizer(
        text,
        truncation=True,      # 길이 제한
        padding='max_length', # 패딩 추가
        max_length=max_length, # 최대 길이
        return_tensors='pt'    # PyTorch 텐서로 반환
    )
```

```
    return tokens
```

```
# 대화형 데이터 토큰나이징
```

```
def tokenize_conversation(qa_pair, tokenizer):
```

```
    # 전체 대화를 하나의 텍스트로 결합
```

```
    conversation_text =
```

```
    f"<|user|>\n{qa_pair['instruction']}\n{qa_pair['input']}\n<|assistant|>\n{qa_pair['output']}"
```

```
    # 토큰나이징
```

```
    tokens = tokenizer(
        conversation_text,
        truncation=True,
        padding=False,
        return_tensors='pt'
    )
```

```
# 레이블 설정 (학습 목표)
```

```
labels = tokens['input_ids'].clone()
```

```
return {
    'input_ids': tokens['input_ids'],
    'attention_mask': tokens['attention_mask'],
    'labels': labels
}
```

3. 데이터셋 생성 및 로딩

3.1 PyTorch 데이터셋 클래스 생성

학습 데이터를 효율적으로 처리하기 위한 데이터셋 클래스를 만듭니다:

```
from torch.utils.data import Dataset, DataLoader
import json

# 법률 서류 학습 데이터셋 클래스
class LegalDocumentDataset(Dataset):
    def __init__(self, data_file, tokenizer, max_length=4096):
        self.tokenizer = tokenizer
        self.max_length = max_length

        # JSON 파일에서 데이터 로드
        with open(data_file, 'r', encoding='utf-8') as f:
            self.data = json.load(f)

    def __len__(self):
        return len(self.data)

    def __getitem__(self, idx):
        # 개별 데이터 항목 반환
        item = self.data[idx]

        # 대화형 텍스트 구성
        conversation =
f"<|user|>\n{item['instruction']}\n{item['input']}\n<|assistant|>\n{item['output']}"

        # 토큰나이징
        tokens = self.tokenizer(
            conversation,
            truncation=True,
            padding='max_length',
            max_length=self.max_length,
            return_tensors='pt'
        )

        return {
            'input_ids': tokens['input_ids'].flatten(),
            'attention_mask': tokens['attention_mask'].flatten(),
            'labels': tokens['input_ids'].flatten()
        }

# 데이터셋 생성 함수
def create_legal_dataset(train_file, val_file, tokenizer):
    # 학습용 데이터셋
    train_dataset = LegalDocumentDataset(train_file, tokenizer)
```

```
# 검증용 데이터셋
val_dataset = LegalDocumentDataset(val_file, tokenizer)

return train_dataset, val_dataset
```

3.2 데이터 로더 설정

배치 단위로 데이터를 효율적으로 처리하기 위한 설정입니다:

```
# 데이터 로더 생성
def create_data_loaders(train_dataset, val_dataset, batch_size=4):
    # 학습용 데이터 로더
    train_loader = DataLoader(
        train_dataset,
        batch_size=batch_size,
        shuffle=True,          # 데이터 순서 무작위 섞기
        num_workers=4,        # 병렬 처리 수
        pin_memory=True        # GPU 메모리 최적화
    )

    # 검증용 데이터 로더
    val_loader = DataLoader(
        val_dataset,
        batch_size=batch_size,
        shuffle=False,         # 검증 시에는 순서 유지
        num_workers=4,
        pin_memory=True
    )

    return train_loader, val_loader
```

4. Fine-tuning 설정 및 실행

4.1 훈련 설정

모델 학습을 위한 세부 설정을 정의합니다:

```
from transformers import TrainingArguments, Trainer

# 훈련 매개변수 설정
def setup_training_arguments():
    training_args = TrainingArguments(
        output_dir='./legal_ai_model',    # 모델 저장 위치
        num_train_epochs=3,                # 학습 반복 횟수
        per_device_train_batch_size=2,     # 배치 크기
```

```

per_device_eval_batch_size=2,      # 평가 배치 크기
gradient_accumulation_steps=8,     # 그래디언트 누적 스텝
warmup_steps=100,                  # 워밍업 스텝
learning_rate=2e-4,                # 학습률
fp16=True,                         # 혼합 정밀도 학습
logging_steps=10,                  # 로그 출력 간격
save_steps=500,                    # 모델 저장 간격
eval_steps=500,                    # 평가 간격
save_total_limit=2,                # 저장할 체크포인트 수
remove_unused_columns=False,       # 사용하지 않는 컬럼 제거하지 않음
dataloader_pin_memory=False,       # 메모리 고정 비활성화
group_by_length=True,              # 길이별 그룹화
report_to=None                      # 리포트 비활성화
)

return training_args

```

4.2 트레이너 설정 및 실행

실제 학습을 진행하는 트레이너를 설정합니다:

트레이너 생성 및 실행

```
def train_legal_ai_model(model, tokenizer, train_dataset, val_dataset):
```

```
    # 훈련 설정 로드
```

```
    training_args = setup_training_arguments()
```

트레이너 생성

```
    trainer = Trainer(
```

```
        model=model,                # 학습할 모델
```

```
        args=training_args,         # 훈련 설정
```

```
        train_dataset=train_dataset, # 학습 데이터
```

```
        eval_dataset=val_dataset,   # 검증 데이터
```

```
        tokenizer=tokenizer,         # 토큰나이저
```

```
        data_collator=None,         # 데이터 콜레이터
```

```
    )
```

학습 시작

```
    print("법률 AI 모델 학습을 시작합니다...")
```

```
    trainer.train()
```

최종 모델 저장

```
    trainer.save_model()
```

```
    tokenizer.save_pretrained('./legal_ai_model')
```

```
    print("모델 학습이 완료되었습니다!")
```

```
    return trainer
```

5. 전체 실행 프로세스

5.1 메인 실행 함수

전체 과정을 순서대로 실행하는 메인 함수입니다:

```
def main():
    print("=== 법률 AI 모델 개발 프로세스 시작 ===")

    # 1단계: 데이터 전처리
    print("1. 데이터 전처리 중...")
    preprocess_legal_documents()

    # 2단계: 모델 로드
    print("2. DeepSeek R1 모델 로드 중...")
    model, tokenizer = load_deepseek_r1_model()
    model = prepare_model_for_training(model)

    # 3단계: 데이터셋 생성
    print("3. 학습 데이터셋 생성 중...")
    train_dataset, val_dataset = create_legal_dataset(
        'train_data.json',
        'val_data.json',
        tokenizer
    )

    # 4단계: 모델 학습
    print("4. 모델 Fine-tuning 시작...")
    trainer = train_legal_ai_model(model, tokenizer, train_dataset, val_dataset)

    # 5단계: 모델 평가
    print("5. 모델 성능 평가 중...")
    eval_results = trainer.evaluate()
    print(f"평가 결과: {eval_results}")

    print("=== 법률 AI 모델 개발 완료 ===")

# 프로그램 실행
if __name__ == "__main__":
    main()
```

5.2 실행 전 확인사항

모델 학습을 시작하기 전에 확인해야 할 사항들입니다:

```

# 시스템 요구사항 확인
def check_system_requirements():
    # GPU 메모리 확인
    if torch.cuda.is_available():
        gpu_memory = torch.cuda.get_device_properties(0).total_memory / (1024**3)
        print(f"GPU 메모리: {gpu_memory:.1f}GB")

        if gpu_memory < 24:
            print("경고: GPU 메모리가 부족할 수 있습니다. 배치 크기를 줄여주세요.")

    # 디스크 공간 확인
    import shutil
    free_space = shutil.disk_usage('.').free / (1024**3)
    print(f"사용 가능한 디스크 공간: {free_space:.1f}GB")

    if free_space < 100:
        print("경고: 디스크 공간이 부족할 수 있습니다.")

    # 데이터 파일 존재 확인
    import os
    required_files = ['train_data.json', 'val_data.json']
    for file in required_files:
        if not os.path.exists(file):
            print(f"오류: {file} 파일이 없습니다.")
            return False

    return True

# 실행 전 확인
if check_system_requirements():
    main()
else:
    print("시스템 요구사항을 확인하고 다시 실행해주세요.")

```

6. 학습 모니터링 및 최적화

6.1 학습 과정 모니터링

모델 학습 상태를 실시간으로 확인하는 방법입니다:

```

# 학습 과정 콜백 함수
from transformers import TrainerCallback

class LegalAITrainingCallback(TrainerCallback):
    def on_log(self, args, state, control, logs=None, **kwargs):
        # 학습 로그 출력

```

```

if logs:
    print(f"Step {state.global_step}: Loss = {logs.get('train_loss', 'N/A')}")

def on_evaluate(self, args, state, control, logs=None, **kwargs):
    # 평가 결과 출력
    if logs:
        print(f"Evaluation at step {state.global_step}:")
        print(f" - Eval Loss: {logs.get('eval_loss', 'N/A')}")
        print(f" - Perplexity: {logs.get('eval_perplexity', 'N/A')}")

def on_save(self, args, state, control, **kwargs):
    # 모델 저장 시 알림
    print(f"모델이 step {state.global_step}에서 저장되었습니다.")

# 콜백 적용
def train_with_monitoring(model, tokenizer, train_dataset, val_dataset):
    training_args = setup_training_arguments()

    trainer = Trainer(
        model=model,
        args=training_args,
        train_dataset=train_dataset,
        eval_dataset=val_dataset,
        tokenizer=tokenizer,
        callbacks=[LegalAITrainingCallback()] # 모니터링 콜백 추가
    )

    trainer.train()
    return trainer

```

6.2 학습 최적화 팁

모델 학습 효율성을 높이는 방법들입니다:

```

# 학습 최적화 설정
def optimize_training_settings():
    # 1. 동적 배치 크기 조정
    def adjust_batch_size_for_gpu():
        if torch.cuda.is_available():
            gpu_memory = torch.cuda.get_device_properties(0).total_memory / (1024**3)
            if gpu_memory >= 80:
                return 8 # A100 80GB
            elif gpu_memory >= 40:
                return 4 # A100 40GB
            else:
                return 2 # RTX 4090 등
        return 1

```

```
# 2. 그래디언트 체크포인팅 활성화
model.gradient_checkpointing_enable()
```

```
# 3. 컴파일 최적화 (PyTorch 2.0 이상)
if hasattr(torch, 'compile'):
    model = torch.compile(model)
```

```
return model
```

```
# 메모리 효율적인 학습 설정
```

```
def setup_memory_efficient_training():
    training_args = TrainingArguments(
        output_dir='./legal_ai_model',
        num_train_epochs=3,
        per_device_train_batch_size=1,          # 작은 배치 크기
        gradient_accumulation_steps=16,         # 그래디언트 누적으로 보상
        gradient_checkpointing=True,            # 그래디언트 체크포인팅
        fp16=True,                             # 혼합 정밀도
        dataloader_pin_memory=False,            # 메모리 고정 비활성화
        optim="adamw_torch_fused",             # 융합된 옵티마이저
        learning_rate=2e-4,
        warmup_steps=100,
        logging_steps=10,
        save_steps=500,
        eval_steps=500,
        save_total_limit=2,
        remove_unused_columns=False,
        group_by_length=True,
        report_to=None
    )

    return training_args
```

이러한 단계별 절차를 따르면 수집한 법률 데이터를 **DeepSeek R1** 모델에 효과적으로 입력하고 **Fine-tuning**을 수행할 수 있습니다. 각 단계는 순차적으로 진행되며, 중간에 문제가 발생하면 해당 단계를 다시 확인하고 수정할 수 있습니다.