Ein Modell Feintunen

```python
import os
import glob
import json
from dotenv import load_dotenv
from transformers import AutoTokenizer, AutoModelForCausalLM, Trainer, TrainingArguments, DataCollatorForLangu
from datasets import Dataset, load_dataset
import torch

load_dotenv()

MODEL_NAME = "deepseek-ai/DeepSeek-R1-Distill-Qwen-1.5B"  # Geändert auf das angegebene Modell
OUTPUT_DIR = "trainiertes_modell"
TRAIN_FILE = "train.jsonl"
MAX_LENGTH = 512
BATCH_SIZE = 8
EPOCHS = 3

def create_training_data(beitraege_verzeichnis):
    alle_texte = []
    for sprachverzeichnis in os.listdir(beitraege_verzeichnis):
        sprachpfad = os.path.join(beitraege_verzeichnis, sprachverzeichnis)
        if not os.path.isdir(sprachpfad):
            continue
        for dateipfad in glob.glob(os.path.join(sprachpfad, "*.md")):
            try:
                with open(dateipfad, 'r', encoding='utf-8') as f:
                    inhalt = f.read()
                    # Front Matter entfernen
                    inhalt = inhalt.split("---", 2)[-1].strip()
                    alle_texte.append(inhalt)
            except Exception as e:
                print(f"Fehler beim Lesen der Datei {dateipfad}: {e}")
    return alle_texte

def prepare_dataset(texte, tokenizer):
    encodings = tokenizer(texte, truncation=True, padding=True, max_length=MAX_LENGTH, return_tensors="pt")
    return Dataset.from_dict(encodings)
```

```
def train_model(dataset, tokenizer):
    training_args = TrainingArguments(
        output_dir=OUTPUT_DIR,
        overwrite_output_dir=True,
        num_train_epochs=EPOCHS,
        per_device_train_batch_size=BATCH_SIZE,
        save_steps=10_000,
        save_total_limit=2,
        prediction_loss_only=True,
        remove_unused_columns=False,
    )
    model = AutoModelForCausalLM.from_pretrained(MODEL_NAME, trust_remote_code=True)
    data_collator = DataCollatorForLanguageModeling(tokenizer=tokenizer, mlm=False)
    trainer = Trainer(
        model=model,
        args=training_args,
        train_dataset=dataset,
        data_collator=data_collator,
    )
    trainer.train()
    trainer.save_model(OUTPUT_DIR)

def main():
    beitraege_verzeichnis = "_beitraege"
    texte = create_training_data(beitraege_verzeichnis)
    tokenizer = LlamaTokenizerFast.from_pretrained(MODEL_NAME, trust_remote_code=True, use_fast=True)
    tokenizer.pad_token = tokenizer.eos_token
    dataset = prepare_dataset(texte, tokenizer)
    train_model(dataset, tokenizer)

if __name__ == "__main__":
    main()
```