

# GPT Nasıl Yapıldı?

## Gpt transformer deep learning modelining eğitilmesi ile mi yapıldı?

Evet 👍

GPT (Generative Pre-trained Transformer), **derin öğrenme (deep learning)** ile eğitilmiş bir modeldir. Daha spesifik olarak:

- **Temel yapı taşı:** Transformer mimarisi (2017'de "Attention is All You Need" makalesinde tanıtıldı).
- **Model tipi:** Büyük dil modeli (LLM).
- **Eğitim süreci:**
  1. **Pretraining (Ön eğitim):** Çok büyük miktarda metin üzerinde *dil modelleme* (bir sonraki kelimeyi tahmin etme) göreviyle eğitilir. Bu aşamada model, dilin yapısını ve genel bilgiyi öğrenir.
  2. **Fine-tuning (İnce ayar):** Belirli görevler (soru-cevap, diyalog, kodlama vb.) için daha küçük veri kümeleriyle yeniden eğitilir veya "Instruction tuning" yapılır.
  3. **RLHF (Reinforcement Learning with Human Feedback):** İnsan geribildirimleriyle pekiştirmeli öğrenme uygulanır, böylece kullanıcı dostu yanıtlar üretmesi sağlanır.


Yani GPT, sıradan bir "klasik makine öğrenmesi algoritması" değil, milyonlarca-parametrelili **derin öğrenme tabanlı bir Transformer modeli**.

**Ben bu transformer mimarisini kullanıp kendi gpt mi yapmak istesem bu transformer mimarisi hazır bir şekilde indirebilir miyim nasıl kullanılıyor?**

Senin "kendi GPT'ni" yapmak istemen aslında şu anki **LLM arařtırmalarının temel konusu**. řöyle özetleyeyim:

---

## 1. Transformer mimarisi hazır mı?

Evet  Transformer mimarisi artık hazır kütüphanelerde var. Sıfırdan matematik yazmak zorunda değilsin.

- **PyTorch:** `torch.nn.Transformer` modülü ile geliyor.
  - **TensorFlow/Keras:** `keras.layers.MultiHeadAttention` ve `Transformer` örnekleri var.
  - **Hugging Face Transformers:** En popüler yöntem → hazır GPT, BERT, T5 vb. modelleri kullanmana ve ince ayar yapmana izin veriyor.
- 

## 2. Kullanım yolları

Senin hedefin ne olduğuna göre 2 yol var:

### A) Sıfırdan (Educational / Research amaçlı)

Kendi küçük GPT'ni eğitmek için:

- `torch.nn.TransformerEncoder` ve `TransformerDecoder` bloklarını kullanabilirsin.
- Küçük bir dataset (ör. Shakespeare, WikiText) ile çalışıp "mini GPT" yapabilirsin.
- Örnek repo: nanoGPT (Andrej Karpathy) → tamamen PyTorch ile GPT-2 benzeri mini bir model eğitiyor.

### B) Hazır büyük modelleri kullanıp Fine-tune yapmak

- Hugging Face `transformers` ile tek satırda model indirebilirsin:

```
from transformers import AutoModelForCausalLM, AutoTokenizer

model = AutoModelForCausalLM.from_pretrained("gpt2")
tokenizer = AutoTokenizer.from_pretrained("gpt2")
```

```
inputs = tokenizer("Merhaba dünya!", return_tensors="pt")
outputs = model.generate(**inputs, max_new_tokens=50)
print(tokenizer.decode(outputs[0]))
```

- Bu şekilde hazır GPT-2, GPT-J, LLaMA, Mistral gibi modelleri çekip kendi verine ince ayar yapabilirsin.

### 3. Kendi GPT'ni yapmak için ne lazım?

- **Donanım:** Büyük model eğitmek için çok güçlü GPU'lar (örn. A100, H100) ya da TPU gerekiyor.
- **Veri:** GPT gibi modeller yüzlerce GB-TB metinle eğitiliyor. Küçük denemeler için birkaç MB yeterli ama performans sınırlı olur.
- **Kütüphane seçimi:** Hugging Face Transformers ile başlamak en kolay yol.

## Kendi gpt mi öğrenme amaçlı yapmak istiyorum

Öğrenme amaçlı kendi GPT'ni yapmak için en mantıklı yol "küçük bir GPT klonu" inşa etmek. Böylece hem **Transformer'ın mantığını öğrenmiş olursun** hem de bilgisayarını yakmadan deney yapabilirsin 😊

Ben sana adım adım yol haritası vereyim:

### ◆ 1. Küçük Ölçekle Başla

Büyük GPT (mesela GPT-3) trilyonlarca parametreye sahip → senin bilgisayarında bu mümkün değil.

Ama **mini-GPT** veya **nanoGPT** gibi küçük versiyonları eğiterek mantığı öğrenebilirsin.

Önerim: Karpathy'nin nanoGPT projesi.

- PyTorch ile sıfırdan yazılmış.
  - Shakespeare metinleriyle "oyun yazabilen" küçük bir GPT eğitiyorsun.
  - Kod 500 satırdan kısa → Transformer'ı anlamak için mükemmel.
- 

## ◆ 2. Adım Adım Öğrenme Süreci

### A) Veri Hazırlama

- Küçük bir metin dosyası al (ör. Shakespeare, Türkçe şiirler, kendi notların).
- Tokenizer (BPE veya karakter bazlı) ile kelimeleri sayılara çevir.

### B) Model Kurulumu

Basit bir GPT şu bileşenlerden oluşur:

1. **Embedding Layer** → kelimeleri vektör yapar.
2. **Positional Encoding** → kelime sırasını öğretir.
3. **Multi-Head Self-Attention** → bağlamı öğrenir.
4. **Feed Forward Network** → bilgiyi işler.
5. **Softmax Output** → sıradaki kelimeyi tahmin eder.

### C) Eğitim

- Loss: Cross Entropy (tahmin edilen kelime vs gerçek kelime).
- Optimizer: AdamW.
- Küçük batch size (ör. 16).
- Learning rate ayarı (ör.  $3e-4$ ).

### D) Deneme

Eğittikten sonra modeline bir başlangıç cümlesi ver:

```
prompt = "Bir zamanlar"
```

ve modelin metni nasıl devam ettirdiğini gör.

### ◆ 3. Örnek Basit Kod (çok küçük bir GPT blok şeması)

```
import torch
import torch.nn as nn
import torch.nn.functional as F

class MiniGPT(nn.Module):
    def __init__(self, vocab_size, d_model, n_heads, n_layers, block_size):
        super().__init__()
        self.token_embedding = nn.Embedding(vocab_size, d_model)
        self.pos_embedding = nn.Embedding(block_size, d_model)
        self.blocks = nn.ModuleList([
            nn.TransformerEncoderLayer(d_model, n_heads, dim_feedforward=4*
d_model)
            for _ in range(n_layers)
        ])
        self.ln = nn.LayerNorm(d_model)
        self.fc_out = nn.Linear(d_model, vocab_size)

    def forward(self, x):
        B, T = x.size()
        tok_emb = self.token_embedding(x)
        pos_emb = self.pos_embedding(torch.arange(T, device=x.device))
        x = tok_emb + pos_emb
        for block in self.blocks:
            x = block(x)
        x = self.ln(x)
        logits = self.fc_out(x)
```

```
return logits
```

Bu kod tek başına **küçük bir GPT iskeleti**.

Sonra `CrossEntropyLoss` ile eğitiyorsun.

## ◆ 4. Sonraki Adımlar

1. **nanoGPT**'yi indir → çalıştır → Shakespeare GPT'yi eğit.
2. Kodunu anlamaya çalış → "embedding → attention → loss" akışını çöz.
3. Kendi verinle (mesela Türkçe hikâyeler) dene.
4. Daha büyük GPU'ya erişimin olursa "daha derin" GPT denemeleri yapabilirsin.

👉 İstersen sana ben "**mini GPT eğitmek için tam çalışan bir Python notebook**" hazırlayabilirim.

Bunu ister **nanoGPT üzerinden**, ister **sıfırdan basit kod** üzerinden yapabilirim.

Hangisini tercih edersin?

- **A** nanoGPT (hazır repo, hızlı sonuç)
- **B** Sıfırdan basit bir notebook (her adımı görerek)

## GPT'nin eğitiminde ne tür derin öğrenme tekniği ve optimizasyon yöntemleri kullanıldı?

GPT'nin eğitiminde kullanılan **derin öğrenme teknikleri** ve **optimizasyon yöntemleri** şöyle özetlenebilir:

--

## ◆ 1. Kullanılan Derin Öğrenme Teknikleri

- **Transformer Mimarisi:** GPT tamamen *self-attention* mekanizması üzerine kuruludur. RNN veya CNN yerine, uzun dizilerdeki bağımlılıkları yakalamak için *Multi-Head Self-Attention* kullanır.
  - **Dil Modelleme (Causal Language Modeling):** Eğitim sırasında temel görev, bir dizideki *bir sonraki token'i* tahmin etmektir. Yani "otoregresif" (autoregressive) bir yaklaşım.
  - **Large-scale Pretraining:** Çok büyük veri kümeleri (kitaplar, makaleler, web verisi) üzerinde, gözetimsiz/yarı-gözetimli şekilde ön eğitim yapılır.
  - **Fine-tuning ve Instruction Tuning:** Önceden eğitilmiş model daha sonra özel görevlerde ince ayar yapılır.
  - **RLHF (Reinforcement Learning from Human Feedback):** İnsanların yazdığı iyi/kötü örneklerden ödül fonksiyonu çıkarılarak model pekiştirmeli öğrenmeyle optimize edilir.
- 

## ◆ 2. Optimizasyon Yöntemleri

- **Kayıp Fonksiyonu:**
  - *Cross-Entropy Loss:* Tahmin edilen dağılım ile gerçek kelimenin olasılığı arasındaki farkı minimize eder.
- **Optimizatörler:**
  - *Adam* ve daha gelişmiş versiyonu *AdamW* (weight decay'li Adam), LLM eğitiminde standarttır.
- **Öğrenme Oranı Planlama:**
  - *Learning Rate Warmup:* Eğitimin ilk adımlarında küçük öğrenme oranıyla başlar, kademeli yükseltilir.
  - *Cosine Decay* veya *Linear Decay* ile zamanla düşürülür.
- **Düzenleştirme (Regularization):**
  - *Dropout:* Aşırı uyumu engellemek için.
  - *Layer Normalization:* Eğitim kararlılığı için.

- *Weight Decay*: Ağırlıkların çok büyümesini engeller.

### ◆ 3. Eğitimde Kullanılan Ek Teknikler

- **Mixed Precision Training (FP16, BF16)**: Daha hızlı ve bellek dostu eğitim için düşük hassasiyetli sayı formatları kullanılır.
- **Gradient Clipping**: Gradyanların aşırı büyümesini engellemek için sınırlandırma yapılır.
- **Model Parallelism & Data Parallelism**: Çok büyük modellerin eğitimi için parametrelerin GPU/TPU'lara dağıtılması.
- **Batching ve Sequence Truncation**: Eğitim verilerinde farklı uzunluktaki diziler uygun şekilde işlenir.

### ◆ 4. Sonuç

Özetle GPT'nin başarısı, sadece Transformer mimarisinden değil; **AdamW optimizasyonu, büyük ölçekli veriler, dikkatli öğrenme oranı planlaması, düzenlileştirme teknikleri ve insan geribildirimiyle yapılan RLHF** aşamalarından kaynaklanır.

## GPT'nin eğitiminde kullanılan ne tür derin öğrenme tekniği ve optimizasyon yöntemleri kullanıldığını da anlat

GPT'nin eğitiminde kullanılan **derin öğrenme teknikleri** ve **optimizasyon yöntemleri** şöyle özetlenebilir:

### ◆ 1. Kullanılan Derin Öğrenme Teknikleri



- **Transformer Mimarisi:** GPT tamamen *self-attention* mekanizması üzerine kuruludur. RNN veya CNN yerine, uzun dizilerdeki bağımlılıkları yakalamak için *Multi-Head Self-Attention* kullanır.
  - **Dil Modelleme (Causal Language Modeling):** Eğitim sırasında temel görev, bir dizideki *bir sonraki token'i* tahmin etmektir. Yani "otoregresif" (autoregressive) bir yaklaşım.
  - **Large-scale Pretraining:** Çok büyük veri kümeleri (kitaplar, makaleler, web verisi) üzerinde, gözetimsiz/yarı-gözetimli şekilde ön eğitim yapılır.
  - **Fine-tuning ve Instruction Tuning:** Önceden eğitilmiş model daha sonra özel görevlerde ince ayar yapılır.
  - **RLHF (Reinforcement Learning from Human Feedback):** İnsanların yazdığı iyi/kötü örneklerden ödül fonksiyonu çıkarılarak model pekiştirmeli öğrenmeyle optimize edilir.
- 

## ◆ 2. Optimizasyon Yöntemleri

- **Kayıp Fonksiyonu:**
  - *Cross-Entropy Loss:* Tahmin edilen dağılım ile gerçek kelimenin olasılığı arasındaki farkı minimize eder.
- **Optimizatörler:**
  - *Adam* ve daha gelişmiş versiyonu *AdamW* (weight decay'li Adam), LLM eğitiminde standarttır.
- **Öğrenme Oranı Planlama:**
  - *Learning Rate Warmup:* Eğitimin ilk adımlarında küçük öğrenme oranıyla başlar, kademeli yükseltilir.
  - *Cosine Decay* veya *Linear Decay* ile zamanla düşürülür.
- **Düzenleştirme (Regularization):**
  - *Dropout:* Aşırı uyumu engellemek için.
  - *Layer Normalization:* Eğitim kararlılığı için.
  - *Weight Decay:* Ağırlıkların çok büyümesini engeller.

---

### ◆ 3. Eğitimde Kullanılan Ek Teknikler

- **Mixed Precision Training (FP16, BF16):** Daha hızlı ve bellek dostu eğitim için düşük hassasiyetli sayı formatları kullanılır.
  - **Gradient Clipping:** Gradyanların aşırı büyümesini engellemek için sınırlandırma yapılır.
  - **Model Parallelism & Data Parallelism:** Çok büyük modellerin eğitimi için parametrelerin GPU/TPU'lara dağıtılması.
  - **Batching ve Sequence Truncation:** Eğitim verilerinde farklı uzunluktaki diziler uygun şekilde işlenir.
- 

### ◆ 4. Sonuç

Özetle GPT'nin başarısı, sadece Transformer mimarisinden değil; **AdamW optimizasyonu, büyük ölçekli veriler, dikkatli öğrenme oranı planlaması, düzenlileştirme teknikleri ve insan geribildirimiyle yapılan RLHF** aşamalarından kaynaklanır.

## Model Parallelism & Data Parallelism nasıl yapılır

### 1) Data Parallelism (veri paralelliği)

**Amaç:** Aynı model kopyasını birden çok GPU'da çalıştırıp, batch'i parçalara (micro-batch) bölerek her kopyada ileri/geri yayılım yapmak; gradyanlar senkronize edilir.

#### En pratik: DistributedDataParallel (DDP)

**Ne zaman?** Tek düğüm (multi-GPU) ya da çok düğüm için altın standart.

## Adımlar

1. `torchrun` ile çoklu süreç başlat.
2. Süreç grubu ( `nccl` ) başlat.
3. DDP ile modeli sarmala.
4. `DistributedSampler` kullan.
5. Her iterasyonda `loss.backward()` ; DDP gradyanları otomatik all-reduce eder.

## Komut

```
torchrun --standalone --nproc_per_node=4 train.py
```

## train.py (iskelet)

```
import os, torch, torch.distributed as dist
from torch.utils.data import DataLoader, DistributedSampler
from torch.nn.parallel import DistributedDataParallel as DDP

def setup():
    dist.init_process_group(backend="nccl")
    torch.cuda.set_device(int(os.environ["LOCAL_RANK"]))

def cleanup():
    dist.destroy_process_group()

def main():
    setup()
    local_rank = int(os.environ["LOCAL_RANK"])
    device = torch.device(f"cuda:{local_rank}")

    model = MyModel().to(device)
    model = DDP(model, device_ids=[local_rank])
```

```

dataset = MyDataset()
sampler = DistributedSampler(dataset)
loader = DataLoader(dataset, batch_size=16, sampler=sampler, num_workers=4, pin_memory=True)

opt = torch.optim.AdamW(model.parameters(), lr=3e-4)

for epoch in range(num_epochs):
    sampler.set_epoch(epoch) # veri karışımı için önemli
    for batch in loader:
        batch = batch.to(device, non_blocking=True)
        logits = model(batch)
        loss = loss_fn(logits, batch.targets)
        opt.zero_grad(set_to_none=True)
        loss.backward()
        opt.step()

cleanup()

if __name__ == "__main__":
    main()

```

## Notlar

- `DataParallel` (tek süreç, çok GPU) yerine **DDP** kullan (daha hızlı/sağlam).
- Kararlı performans için: **num\_workers**, **pin\_memory**, **non\_blocking**, **gradient accumulation** (büyük batch emülasyonu) uygula.

## 2) Model Parallelism (model paralelliği)

**Amaç:** Model tek GPU'ya sığmıyorsa parametreleri ve hesapları *bölmek*. Üç ana tür:

## A) Pipeline Parallelism (katman bazında boru hattı)

Transformer katmanlarını sıraya böl: ör. Katman 0–11 GPU0’da, 12–23 GPU1’de. Micro-batch’leri ardışık akıttı ilerlet, bubble’ı azaltmak için **gradient accumulation steps** ayarla.

### PyTorch Pipe ile fikri iskelet

```
import torch
import torch.nn as nn
from torch.distributed.pipeline.sync import Pipe

# Örnek: iki parçaya ayrılmış "sıralı" Transformer
part1 = nn.Sequential(*layers[:12]).to("cuda:0")
part2 = nn.Sequential(*layers[12:]).to("cuda:1")

model = nn.Sequential(part1, part2)
chunks = 4 # micro-batch sayısı (pipeline verimi için)
pipelined_model = Pipe(model, chunks=chunks)

for micro in micro_loader:      # micro_loader batch'i micro-batch'lere böler
    out = pipelined_model(micro)
    loss = loss_fn(out, targets)
    loss.backward()
    optimizer.step(); optimizer.zero_grad()
```

**Artılar:** Kolay ölçeklenir, bellek yükünü böler.

**Eksiler:** Pipeline “bubble” overhead. Katmanların dengeli bölünmesi önemli.

## B) Tensor Parallelism (katman içi bölme)

Bir katmanın (ör. attention veya MLP’nin) **ağırlık matrislerini** yatay/dikey parçalara bölüp çoklu GPU’larda aynı katmanı birlikte hesaplamak. Ör: QKV projeksiyonlarını başlara göre bölmek.

- PyTorch çekirdeğinde "hazır sihirli düğme" az; pratikte **Megatron-LM/Tensor Parallel** implementasyonları kullanılır.
- Yeni PyTorch **DTensor / torch.distributed.tensor** API'leri de bu iş için geliyor (gelişmiş kullanım).

#### Pratik yol:

- **Megatron-LM:** `tensor-model-parallel-size` ayarla (örn. 2 veya 4); model tanımı dikkatle bölünmüştür (ColumnParallelLinear, RowParallelLinear).
- **DeepSpeed-Megatron** entegrasyonları ile *tensor + pipeline + data* paralelliği birlikte (3D parallel).

## C) Sharding (parametre/gradyan/optimizer durumlarını bölme)

Tüm parametreleri kopyalamak yerine, **optimizer state / gradyan / parametreleri** GPU'lara paylaştır:

- **FSDP (Fully Sharded Data Parallel)** – PyTorch yerel.
- **DeepSpeed ZeRO** (stage 1/2/3) – optimizer state → grad → parametre sharding.

### FSDP minimal örnek

```
import torch
import torch.distributed as dist
from torch.distributed.fsdp import FullyShardedDataParallel as FSDP
from torch.distributed.fsdp.wrap import transformer_auto_wrap_policy
from torch.nn import TransformerEncoderLayer

def setup():
    dist.init_process_group("nccl")
    torch.cuda.set_device(int(os.environ["LOCAL_RANK"]))

def main():
    setup()
    device = torch.device(f"cuda:{int(os.environ['LOCAL_RANK'])}")
```

```
policy = transformer_auto_wrap_policy({TransformerEncoderLayer})
model = MyTransformerModel().to(device)
model = FSDP(model, auto_wrap_policy=policy, use_orig_params=True)

opt = torch.optim.AdamW(model.parameters(), lr=3e-4)
for batch in loader:
    batch = batch.to(device, non_blocking=True)
    out = model(batch)
    loss = loss_fn(out, batch.targets)
    opt.zero_grad(set_to_none=True)
    loss.backward()
    opt.step()

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

**Artılar:** Tek düğümde bile devasa modelleri eğitilebilir kılar.

**Dikkat:** Checkpointing, CPU offload (opsiyonel), mixed precision ayarları (bf16/fp16) önemlidir.

### 3) “Hangi durumda hangisi?”

- **Model sığıyorsa:** Sadece **DDP** (kolay ve hızlı).
- **Model sığmıyorsa 1:** **FSDP** (veya ZeRO) ile şerit (shard) ederek sığdır.
- **Aşırı büyük model:**
  - **3D Parallel:** Data (DDP) + Pipeline + Tensor parallel birlikte.
  - **Megatron-LM / DeepSpeed** paketlerini tercih et.
- **GPU bellek yetmiyor ama bant genişliğin iyi:** FSDP + CPU offload / activation checkpointing.

## 4) Performans ipuçları

- **bf16/fp16 (AMP)** kullan: `torch.cuda.amp.autocast(dtype=torch.bfloat16)`
  - **Gradient checkpointing**: Aktivasyon belleğini düşürür, ek hesaplama getirir.
  - **AdamW + fused kernels** (Apex/FlashAttention gibi kütüphaneler) hız kazandırır.
  - **İ/O**: `persistent_workers=True` , `prefetch_factor` ile DataLoader ayarlarını dene.
  - **İletişim**: NCCL versiyonu, PCIe vs NVLink topolojisi performansı belirler.
- 

## 5) Hızlı karar ağacı

1. **Tek makine, 2-4 GPU, model sığıyor** → DDP.
2. **Tek/çok makine, model sığmıyor** → FSDP (veya DeepSpeed ZeRO-3).
3. **Daha da büyük** → Pipeline (katman böl) + Tensor (katman içi böl) + Data (DDP) = 3D.
4. **Kütüphane seçimi**:
  - "Sıfırdan" öğrenmek için → DDP + FSDP.
  - Üretim ölçeği/GPT benzeri devasa modeller → **Megatron-LM** veya **DeepSpeed-Megatron**.

## RLHF (Reinforcement Learning from Human Feedback) nasıl yapılır örnek python kodu oluştur

aşağıda **uçtan uca (oyuncak)** bir RLHF hattı var:

1. **SFT (denetimli ince ayar),**



2. **Ödül modeli** (ikili tercih verisiyle),
3. **PPO ile pekiştirmeli ayar** (TRL kütüphanesi).

Not: Bu öğrenme amaçlı, küçük ölçekli bir örnektir (GPT-2 ile). Gerçek üretim senaryolarında veri, model boyutu, dağıtık eğitim, güvenlik filtreleri vb. çok daha kapsamlı olur.

## 0) Kurulum

```
pip install "transformers>=4.42" "datasets>=2.19" "accelerate>=0.33" "trl>=0.9" torch --upgrade
```

## 1) SFT: Talimat-yanıt çiftleriyle ön-inceltme

```
# sft_train.py
import torch, random
from datasets import Dataset
from transformers import AutoTokenizer, AutoModelForCausalLM, Trainer, TrainingArguments, DataCollatorForLanguageModeling

MODEL_NAME = "gpt2" # küçük ve hızlı olsun diye
tokenizer = AutoTokenizer.from_pretrained(MODEL_NAME)
tokenizer.pad_token = tokenizer.eos_token

# Oyuncak veri (prompt → response)
pairs = [
    {"prompt": "Soru: Çayın ideal demleme süresi?\nCevap:", "response": "Gene lde 5-7 dakikadır."},
    {"prompt": "Komik bir fıkra anlat.\nCevap:", "response": "Temel bir gün... (kısa bir fıkra)"},
    {"prompt": "Python'da liste nasıl ters çevrilir?\nCevap:", "response": "list(rev
```

```

ersed(x)) ya da x[::-1] kullanılır."},
]
def format_example(ex):
    return {"text": ex["prompt"] + ex["response"]}

ds = Dataset.from_list([format_example(p) for p in pairs])

def tokenize(batch):
    out = tokenizer(batch["text"], truncation=True, max_length=512)
    return out

ds = ds.map(tokenize, batched=True, remove_columns=["text"])

model = AutoModelForCausalLM.from_pretrained(MODEL_NAME)
model.resize_token_embeddings(len(tokenizer))

args = TrainingArguments(
    output_dir="out_sft",
    per_device_train_batch_size=2,
    num_train_epochs=2,
    learning_rate=5e-5,
    weight_decay=0.01,
    fp16=torch.cuda.is_available(),
    logging_steps=5,
)

collator = DataCollatorForLanguageModeling(tokenizer, mlm=False)
trainer = Trainer(model=model, args=args, train_dataset=ds, data_collator=collator)
trainer.train()
trainer.save_model("sft_model")
tokenizer.save_pretrained("sft_model")

```

Çalıştır:

```
python sft_train.py
```

## 2) Ödül Modeli: İkili tercih (chosen vs rejected)

**Amaç:** Bir metne (prompt + model cevabı) **skor** veren model eğitmek. Aşağıda *aynı taban dil modeli* üstüne tek nöronluk bir **özel kafa** ekleniyor ve **Bradley–Terry** tarzı ikili tercih kaybı kullanılıyor.

```
# reward_train.py
import torch, math
from torch import nn
from datasets import Dataset
from transformers import AutoTokenizer, AutoModelForCausalLM, AutoConfig

device = "cuda" if torch.cuda.is_available() else "cpu"
BASE = "gpt2" # aynı taban; pratikte daha büyük/iyi model ve çok daha fazla
              veri kullanın

tokenizer = AutoTokenizer.from_pretrained(BASE)
tokenizer.pad_token = tokenizer.eos_token

# Oyuncak tercih verisi: her bir prompt için iki yanıt (chosen > rejected)
pref_pairs = [
    {
        "prompt": "Nazikçe reddetme cümlesi yaz.\nCevap:",
        "chosen": "Üzgünüm, şu an yardımcı olamıyorum ama umarım yakında gö  
rüşürüz.",
        "rejected": " Hayır."
    },
    {
        "prompt": "Kahveyi nasıl demlerim?\nCevap:",
```

```

        "chosen": "Taze öğüt, 1:15 oran, 92-96°C su; 3-4 dk demle.",
        "rejected": "Sıcak su dök gitsin."
    },
]

def pack(example):
    # prompt + response → tek dize
    return {
        "chosen_text": example["prompt"] + example["chosen"],
        "rejected_text": example["prompt"] + example["rejected"],
    }

ds = Dataset.from_list([pack(p) for p in pref_pairs])

def tok_fn(batch):
    c = tokenizer(batch["chosen_text"], truncation=True, max_length=512, return_tensors=None)
    r = tokenizer(batch["rejected_text"], truncation=True, max_length=512, return_tensors=None)
    return {
        "c_input_ids": c["input_ids"], "c_attn": c["attention_mask"],
        "r_input_ids": r["input_ids"], "r_attn": r["attention_mask"],
    }

ds = ds.map(tok_fn, batched=True, remove_columns=["chosen_text", "rejected_text"])

# Basit ödül başlığı: son gizil durumun ortalaması → linear → skaler
class RewardModel(nn.Module):
    def __init__(self, base_name):
        super().__init__()
        self.backbone = AutoModelForCausalLM.from_pretrained(base_name)
        # dil modeli logits'i kullanmayacağız; son gizil katmanı alacağız
        hidden = self.backbone.transformer.wte.weight.shape[1]
        self.value_head = nn.Linear(hidden, 1, bias=False)

```

```

def score(self, input_ids, attention_mask):
    out = self.backbone.transformer(input_ids=input_ids, attention_mask=attention_mask)
    last_hidden = out.last_hidden_state # [B, T, H]
    mask = attention_mask.unsqueeze(-1) # [B, T, 1]
    mean_pooled = (last_hidden * mask).sum(dim=1) / mask.sum(dim=1).clamp(min=1)
    r = self.value_head(mean_pooled).squeeze(-1) # [B]
    return r

def forward(self, **kwargs):
    raise NotImplementedError("Kullanım: .score(...)")

rm = RewardModel(BASE).to(device)
opt = torch.optim.AdamW(rm.parameters(), lr=1e-5)

def bt_loss(r_chosen, r_rejected): # Bradley-Terry
    # L = -log( sigmoid(r_c - r_r) )
    return torch.nn.functional.softplus(-(r_chosen - r_rejected)).mean()

EPOCHS = 3
for epoch in range(EPOCHS):
    for row in ds:
        c_ids = torch.tensor([row["c_input_ids"]], device=device)
        c_attn = torch.tensor([row["c_attn"]], device=device)
        r_ids = torch.tensor([row["r_input_ids"]], device=device)
        r_attn = torch.tensor([row["r_attn"]], device=device)

        r_c = rm.score(c_ids, c_attn)
        r_r = rm.score(r_ids, r_attn)
        loss = bt_loss(r_c, r_r)

        opt.zero_grad(set_to_none=True)
        loss.backward()
        opt.step()
    print(f"epoch {epoch} loss={loss.item():.4f}")

```

```
torch.save(rm.state_dict(), "reward_model.pt")
tokenizer.save_pretrained("reward_tok")
```

Gerçekte: çok daha büyük tercih setleri (ör. chosen/rejected), regularization, kalibrasyon ve bazen KL cezası hedefi için referans model saklanır.

### 3) PPO ile RLHF (TRL kullanarak)

TRL, PPO döngüsünü pratikleştirir: *policy* (SFT'den kopya), *ref policy* (donmuş kopya), *reward model* → **ödül** = **RM skoru** –  $\beta \cdot \text{KL}(\text{policy} \parallel \text{ref})$ .

```
# ppo_rlhf.py
import torch
from datasets import Dataset
from transformers import AutoTokenizer, AutoModelForCausalLM
from trl import PPOConfig, PPOTrainer, AutoModelForCausalLMWithValueHead

device = "cuda" if torch.cuda.is_available() else "cpu"

# 1) İlke (policy): SFT modelini yükle
POLICY_DIR = "sft_model"
tokenizer = AutoTokenizer.from_pretrained(POLICY_DIR)
tokenizer.pad_token = tokenizer.eos_token

policy = AutoModelForCausalLMWithValueHead.from_pretrained(POLICY_DIR).to(device)

# 2) Referans politika (KL için donmuş)
ref_policy = AutoModelForCausalLM.from_pretrained(POLICY_DIR).to(device)
ref_policy.eval()
for p in ref_policy.parameters():
```

```

p.requires_grad_(False)

# 3) Prompt veri seti (oyuncak)
prompts = [
    "Nazikçe reddetme cümlesi yaz.",
    "Kahveyi nasıl demlerim?",
    "Birine yapıcı geri bildirim nasıl verilir?",
]
ds = Dataset.from_list([{"prompt": p} for p in prompts])

# 4) Ödül modeli (2. adımda eğitilen)
from reward_train import RewardModel # aynı dosyadan sınıfı alıyoruz
rm = RewardModel("gpt2").to(device)
rm.load_state_dict(torch.load("reward_model.pt", map_location=device))
rm.eval()
for p in rm.parameters():
    p.requires_grad_(False)

# 5) PPO konfigürasyonu
config = PPOConfig(
    model_name=POLICY_DIR,
    learning_rate=1e-5,
    batch_size=2,
    mini_batch_size=2,
    ppo_epochs=2,
    adap_kl_ctrl=True, # KL otomatik kontrol
    init_kl_coef=0.1, # beta (başlangıç)
    target_kl=0.1,
)

ppo_trainer = PPOTrainer(config, policy, ref_policy, tokenizer=tokenizer, dataset=ds)

# 6) PPO döngüsü
gen_kwargs = dict(max_new_tokens=64, do_sample=True, top_p=0.9, temperature=1.0)

```

```

for epoch in range(3):
    for batch in ppo_trainer.dataloader:
        prompts_txt = batch["prompt"]
        # Tokenize prompts
        inputs = tokenizer(prompts_txt, return_tensors="pt", padding=True).to(device)

        # Policy'den yanıt üret
        with torch.no_grad():
            gen_ids = policy.generate(**inputs, **gen_kwargs)
            responses_txt = tokenizer.batch_decode(gen_ids[:, inputs["input_ids"].shape[1]:], skip_special_tokens=True)

        # Ödülleri hesapla: RM skoru - KL cezası (TRL içinde KL zaten hesaba katılır;
        # burada saf RM skorunu hesaplıyor ve ppo_trainer'a veriyoruz)
        # TRL, passed "rewards" + kl_ctrl ile toplamı optimize eder.
        rewards = []
        for ptxt, rtxt in zip(prompts_txt, responses_txt):
            text = ptxt + "\nCevap:" + rtxt
            toks = tokenizer(text, return_tensors="pt", truncation=True, max_length=512).to(device)
            with torch.no_grad():
                rscore = rm.score(toks["input_ids"], toks["attention_mask"]).detach()
            rewards.append(rscore.squeeze().cpu())

        # PPO update
        stats = ppo_trainer.step(prompts_txt, responses_txt, [torch.tensor(r.item()) for r in rewards])
        ppo_trainer.log_stats(stats, {"prompts": prompts_txt, "responses": responses_txt}, rewards)

    # Eğitilmiş politikayı kaydet
    ppo_trainer.save_pretrained("ppo_policy")
    tokenizer.save_pretrained("ppo_policy")

```



```
print("PPO tamamlandı, model 'ppo_policy' klasöründe.")
```

Çalıştırma sırası:

```
python sft_train.py  
python reward_train.py  
python ppo_rlhf.py
```

## Kısa açıklamalar / Pratik ipuçları

- **SFT veri kalitesi** RLHF başarısını belirler; talimat-yanıtlar temiz olmalı.
- **Ödül modeli:** Mümkünse *separate encoder* (ör. DeBERTa) ya da daha büyük bir LM üstüne value head; güçlü tercih verisi gerekir.
- **KL kontrolü:** Yanıtların SFT dağılımından sapmasını denetler (TRL’de otomatik).
- **Stabilite:** bf16/fp16, gradient clipping, daha büyük batch’ler (accumulation), doğru max\_len ve prompt şablonu önemli.
- **Güvenlik:** RLHF ödülü yan etkili olabilir; ek kısıtlayıcı kurallar, içerik filtreleri, “harms”-“helpfulness” çok boyutlu ödüller düşünün.