Llama-3 8b Unsloth 2x Faster Fine-tuning

Llama-3 8b modeli, 15 trilyon token üzerinde eğitilmiş güçlü bir dil modelidir. Bu dokümantasyon, Unsloth kütüphanesi ile 2 kat daha hızlı fine-tuning işlemlerini nasıl yapabileceğimizi gösterecektir.

Gereksinimler

Google Colab üzerinde ücretsiz Tesla T4 GPU ile çalıştırabilirsiniz.

1-Hugging Face CLI Giriş

- Hugging Face platformuna giriş yapmak için API token kullanılır. Bu token, kullanıcıya özel olup, modellerin ve verilerin Hugging Face ile etkileşime girmesini sağlar.

```
!huggingface-cli login
```

2-Gerekli Kütüphanelerin Kurulumu

```
!pip install "unsloth[colab-new] @ git+https://github.com/unslothai/unsloth.git"

!pip install --no-deps "xformers<0.0.27" "trl<0.9.0" peft accelerate bitsandbytes
```

3-Modeli Yükleme ve Hazırlama

- Unsloth kütüphanesi ile modeli 4bit olarak yükleyerek bellek kullanımını azaltabilir ve daha hızlı işlem yapabiliriz.

```
from unsloth import FastLanguageModel
import torch
max_seq_length = 2048
dtype = None
load_in_4bit = True

model, tokenizer = FastLanguageModel.from_pretrained(
    model_name = "unsloth/llama-3-8b-bnb-4bit",
    max_seq_length = max_seq_length,
    dtype = dtype,
    load_in_4bit = load_in_4bit,
)
```

Açıklama:
Yukarıdaki kod bloğu, Llama-3 8b modelini 4bit quantization ile yükler.

‘max_seq_length’ değişkeni, modelin işlem yapabileceği maksimum sekans uzunluğunu belirtir.’ **load_in_4bit’** parametresi ise bellek optimizasyonu için kullanılır.

4-LoRA Adaptörlerini Ekleme

- LoRA (Low-Rank Adaptation) adaptörleri ekleyerek, model parametrelerinin sadece %1 ila %10'unu güncelleyerek daha hızlı bir fine-tuning işlemi gerçekleştirebiliriz.

```
model = FastLanguageModel.get_peft_model(
    model,
    r = 16, # Choose any number > 0 ! Suggested 8, 16, 32, 64, 128
    target_modules = ["q_proj", "k_proj", "v_proj", "o_proj",
                      "gate_proj", "up_proj", "down_proj",],
    lora_alpha = 16,
    lora_dropout = 0, # Supports any, but = 0 is optimized
    bias = "none",    # Supports any, but = "none" is optimized
    # [NEW] "unsloth" uses 30% less VRAM, fits 2x larger batch sizes!
    use_gradient_checkpointing = "unsloth", # True or "unsloth" for very
long context
    random_state = 3407,
    use_rslora = False,  # We support rank stabilized LoRA
    loftq_config = None, # And LoftQ
)
```

Açıklama:
Bu kod, model üzerinde LoRA adaptörlerini etkinleştirir. **‘r’** parametresi, adaptör boyutunu belirler. **‘target_modules’**, LoRA adaptörlerinin uygulanacağı modülleri belirtir. **‘use_gradient_checkpointing’** parametresi, bellek optimizasyonu sağlar.

5-Veri Setini Hazırlama

- Veri setini, modelin eğitimi için uygun formata getirmek gereklidir. Aşağıdaki örnekte, Alpaca prompt formatı kullanılarak veri seti hazırlanmıştır.

```
alpaca_prompt = """Below is an input that provides context. Write a
response that appropriately completes the request.

### Input:
{}

### Response:
{}"""

EOS_TOKEN = tokenizer.eos_token # Must add EOS_TOKEN
def formatting_prompts_func(examples):
    inputs       = examples["input"]
    outputs      = examples["output"]
    texts = []
    for input, output in zip(inputs, outputs):
        # Must add EOS_TOKEN, otherwise your generation will go on
forever!
        text = alpaca_prompt.format(input, output) + EOS_TOKEN
        texts.append(text)
    return { "text" : texts, }
```

Açıklama:

Bu fonksiyon, **'input'** ve **'output'** verilerini Alpaca prompt formatına göre birleştirir ve EOS token ekleyerek modelin tanıyabileceği bir formata dönüştürür. Bu işlem, modelin eğitim sırasında veri setini doğru bir şekilde işlemesini sağlar.

Veri setinizi Hugging Face Dataset kütüphanesinden yükleyebilir ve işleyebilirsiniz:

```
from datasets import load_dataset
dataset = load_dataset("odenmehmet/TRObjectTEst", split = "train")
# Verisetindeki anahtar adını değiştirin
dataset = dataset.map(formatting_prompts_func, batched = True,)
```

Açıklama:

Bu kod, Hugging Face Dataset kütüphanesini kullanarak veri setini yükler ve yukarıda tanımlanan **'formatting_prompts_func'** fonksiyonu ile verileri işlemeye hazır hale getirir.

6-Modeli Eğitme

- Modelin fine-tuning işlemi için **SFTTrainer** kullanılır. Aşağıdaki kod, modelinizi eğitmek için bir eğitim döngüsü oluşturur.

```
from trl import SFTTrainer
from transformers import TrainingArguments
from unsloth import is_bfloat16_supported

trainer = SFTTrainer(
    model = model,
    tokenizer = tokenizer,
    train_dataset = dataset,
    dataset_text_field = "text",
    max_seq_length = max_seq_length,
    dataset_num_proc = 2,
    packing = False, # Can make training 5x faster for short sequences.
    args = TrainingArguments(
        per_device_train_batch_size = 3,
        gradient_accumulation_steps = 6,
        warmup_steps = 5,
        max_steps = 120,
        learning_rate = 2e-4,
        fp16 = not is_bfloat16_supported(),
        bf16 = is_bfloat16_supported(),
        logging_steps = 1,
        optim = "adamw_8bit",
        weight_decay = 0.01,
        lr_scheduler_type = "linear",
        seed = 3407,
        output_dir = "outputs",
    ),
)
trainer_stats = trainer.train()
```

Açıklama:

- **‘per_device_train_batch_size’:** Her GPU için eğitim sırasında kullanılacak batch boyutunu belirtir.
- **‘gradient_accumulation_steps’:** Eğitim sırasında gradientlerin kaç adımda bir güncelleneceğini belirtir.
- **‘max_steps’:** Eğitim süresince gerçekleştirilecek maksimum adım sayısını belirtir.
- **‘learning_rate’:** Modelin öğrenme hızını belirtir.
- **‘fp16’** ve’ **bf16’:** Modelin hesaplama türünü belirler; fp16 ve bf16 seçenekleriyle bellek kullanımını optimize edebilirsiniz.
- **‘output_dir’:** Eğitim çıktılarının kaydedileceği dizini belirtir.

7-Modeli Kaydetme

- Eğitimi tamamlanan modeli yerel bir dizine veya doğrudan Hugging Face Hub'a kaydedebilirsiniz..

```
model.save_pretrained("lora_model") # Local saving
tokenizer.save_pretrained("lora_model")
```

Açıklama:
Bu kod, eğitilen model ve tokenizer'ı **"lora_model"** dizinine kaydeder. Bu dosyaları daha sonra kullanarak modelinizi yeniden yükleyebilir ve inference işlemleri gerçekleştirebilirsiniz.

8-Inference(Çıktı Alma)

- Eğitilen modelle inference işlemleri yapmak için aşağıdaki kodu kullanabilirsiniz. Bu örnekte, modelden bir menü oluşturması istenmektedir.

```
inputs = tokenizer(
[
    alpaca_prompt.format(
        "What is a famous tall tower in Paris?", # instruction
        "", # input
        "", # output - leave this blank for generation!
    )
], return_tensors = "pt").to("cuda")

outputs = model.generate(**inputs, max_new_tokens = 64, use_cache =
True)
tokenizer.batch_decode(outputs)
```

Açıklama:

- max_new_tokens: Modelin üreteceği maksimum yeni token sayısını belirtir.
- use_cache: Modelin daha hızlı inference yapabilmesi için cache kullanmasını sağlar.

Bu kod, verilen komuta göre modelden bir yanıt üretir ve çıktıyı ekrana basar.

9-Modeli Hugging Face Hub'a Yükleme

```
if False: model.save_pretrained_gguf("model", tokenizer,
quantization_method = "q4_k_m")
if True: model.push_to_hub_gguf("odenmehmet/modelll", tokenizer,
quantization_method = "q4_k_m", token = "your_HFtoken")
```

- **Modeli GGUF formatında kaydetme ('save_pretrained_gguf')**:
  - Bu fonksiyon, modelinizi yerel olarak GGUF (Quantized GGUF Format) formatında kaydetmek için kullanılır.
  - **'quantization_method = "q4_k_m"'** ile modelinizin quantize edilmesi sağlanır. Bu yöntem, modelin boyutunu küçülterek bellek kullanımını azaltır, ancak performanstan ödün vermez.
- **Modeli Hugging Face Hub'a GGUF formatında yükleme ('push_to_hub_gguf'):**
  - Bu fonksiyon, modelinizi yerel olarak GGUF (Quantized GGUF Format) formatında kaydetmek için kullanılır.
  - **'quantization_method = "q4_k_m"'** parametresiyle yine aynı quantize işlemi yapılır.
  - **'token'** parametresi, Hugging Face Hub'daki depoya erişim sağlamak için gerekli olan erişim anahtarıdır. Bu anahtar gizli tutulmalıdır.

Modeli Local'e İndirdikten Sonra API Uç Noktasından Metin Üretimi

1- Gerekli Kütüphaneler ve Modüller

- FastAPI uygulamasını başlatmak, model yüklemek ve asenkron işlemleri desteklemek için gerekli kütüphaneler yüklenir.

```
from fastapi import FastAPI
from pydantic import BaseModel
import llama_cpp
import nest_asyncio
import uvicorn
```

2- Asenkron Destek için Ayarların Yapılması

- **'nest_asyncio'** kütüphanesi, asenkron işlemleri uyumlu hale getirir, böylece FastAPI uygulaması sorunsuz çalışır.

```
nest_asyncio.apply()
```

3- FastAPI Uygulamasının Tanımlanması

- FastAPI kullanılarak bir web API'si oluşturulur.

```
app = FastAPI()
```

4- Modelin Yüklenmesi

- Modelin yükleneceği dosya yolu belirtilir ve **'llama_cpp'** kütüphanesi kullanılarak model yüklenir.

```
# Modelin yolunu belirtme
model_path = "model_path"

# Llama_cpp ile modeli yükleme
llm = llama_cpp.Llama(model_path=model_path,
                      n_gpu_Layers=-1,
                      n_threads=10,
                      )
```

5- İstek ve Yanıt Modellerinin Tanımlanması

- API'ye gönderilen verileri ve dönecek yanıtları temsil eden modeller tanımlanır.
- **'TextGenerationRequest'** modeli, kullanıcıdan gelen **'prompt'** ve **'max_length'** değerlerini içerir. **'TextGenerationResponse'** modeli ise üretilen metni içerir.

```
# İstek modeli
class TextGenerationRequest(BaseModel):
    prompt: str
    max_length: int = 150

# Yanıt modeli
class TextGenerationResponse(BaseModel):
    generated_text: str
```

6-Metin Üretimi İçin API Endpoint Tanımlaması

- API üzerinde **'/generate'** adlı bir POST endpoint'i tanımlanır. Kullanıcının gönderdiği **'prompt'** ile modele talimat verilir ve modelin ürettiği yanıt döndürülür.

```
@app.post("/generate", response_model=TextGenerationResponse)
def generate_text(request: TextGenerationRequest):
    # Prompt oluşturma
    alpaca_prompt = """Below is an instruction that describes a task.
Write a response that appropriately completes the request.

### Instruction:
{}

### Response:
{}"""

    # inputs (prompt) oluşturma
    full_prompt = alpaca_prompt.format(request.prompt, "")

    # Modeli kullanarak metin üretimi
    output = llm(full_prompt,
                 max_tokens=request.max_length,
                 temperature=0.1,  # Örneğin, bu değeri ayarlayarak
deneme yapabilirsiniz
                 top_p=0.9,        # Bu değeri de deneyebilirsiniz
                 echo=False)
    generated_text = output['choices'][0]['text'].strip()

    return TextGenerationResponse(generated_text=generated_text)
```

Parametre Açıklamaları:

- **max_tokens:** Modelin üreteceği maksimum yeni token sayısını belirtir. Bu, üretilecek metnin uzunluğunu sınırlar.
- **temperature:** Modelin yaratıcılığını kontrol eder. Düşük değerler daha tutarlı, yüksek değerler ise daha yaratıcı yanıtlar üretir.
- **top_p:** Çıkış metninin çeşitliğini artırmak için kullanılan bir diğer parametre. **'top_p=0.9'** ile en olası çıkışların %90'ını kapsayan bir örnekleme yapılır.
- **echo:** Yanıt içinde verilen prompt'un tekrarlanıp tekrarlanmayacağını kontrol eder. False değeri ile prompt yanıt içinde yer almaz.

7-Uygulamanın Başlatılması

- API sunucusunun başlatılması için 'uvicorn' kullanılır. Sunucu, belirtilen host ve port üzerinden çalışır.

```
if __name__ == "__main__":
    uvicorn.run(app, host="0.0.0.0", port=8000)
```

Çalışma Prensibi

Bu kod, belirli bir **prompt** kullanarak modelden bir metin üretir. İstek JSON formatında bir **prompt** ve **max_length** parametrelerini içerir. API, bu isteğe yanıt olarak belirttiğimiz özelliklere göre bir metin döndürür.

Kod Kullanımı: Inference (Çıktı Alma)

Bu kod, verilen prompt'a göre modelden bir yanıt üretir ve yanıtı JSON formatında döndürür.

Endpoint: '/generate' **,**

Metod: 'POST' **,**

İstek Gövdesi (JSON):

```
{
      "prompt": "Açıklanacak görevi girin.",
      "max_length": 150
}
```

Yanıt (JSON):

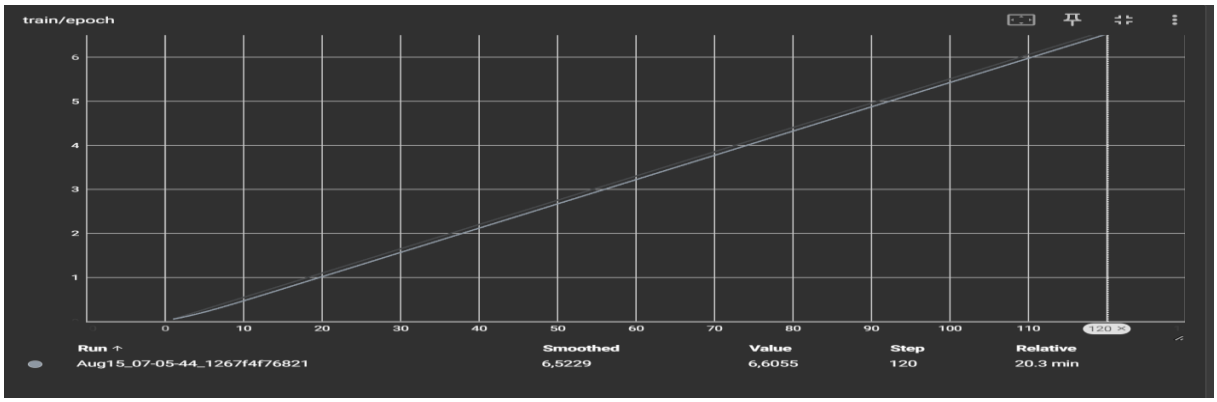
```
{
      "generated_text": Üretilen metin burada olacak.",
}
```

Açıklama:

- **prompt:** Modelin yanıt üretmesi için verilen talimat.
- **max_length:** Modelin üreteceği maksimum yeni token sayısını belirtir. Yanıtın uzunluğunu sınırlar.

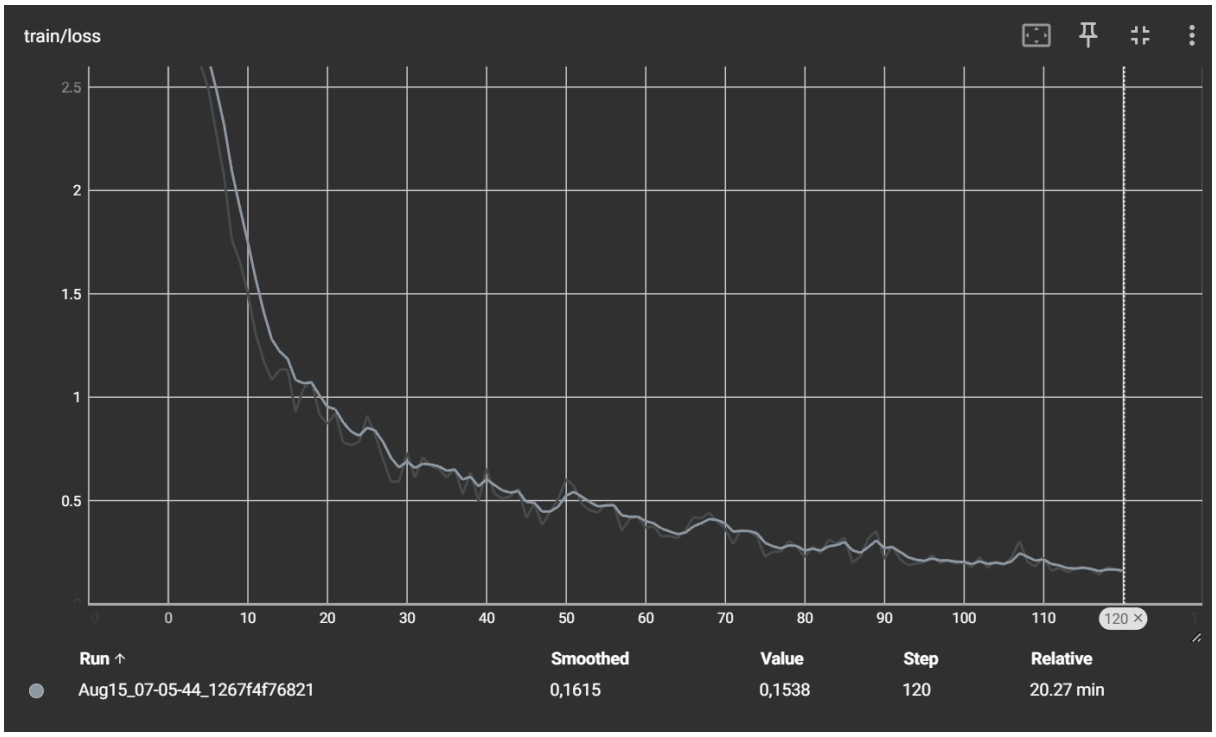
Bu kod, verilen prompt'a göre modelden bir yanıt üretir ve yanıtı JSON formatında döndürür.

8-TensorBoard ile Eğitim Sonuçlarını İzleme



1. train/epoch Grafiği

- **Açıklama:** Bu grafik, modelin eğitim süreci boyunca her epoch'ta alınan ortalama eğitim kaybını (training loss) gösterir. Yatay eksende epoch sayısı, dikey eksende ise kayıp değeri yer alır. Grafikte kayıp değerlerinin epoch'lar ilerledikçe nasıl azaldığını görebilirsiniz. Bu azalma, modelin veriyi daha iyi öğrenmesi ve performansının iyileşmesi anlamına gelir.



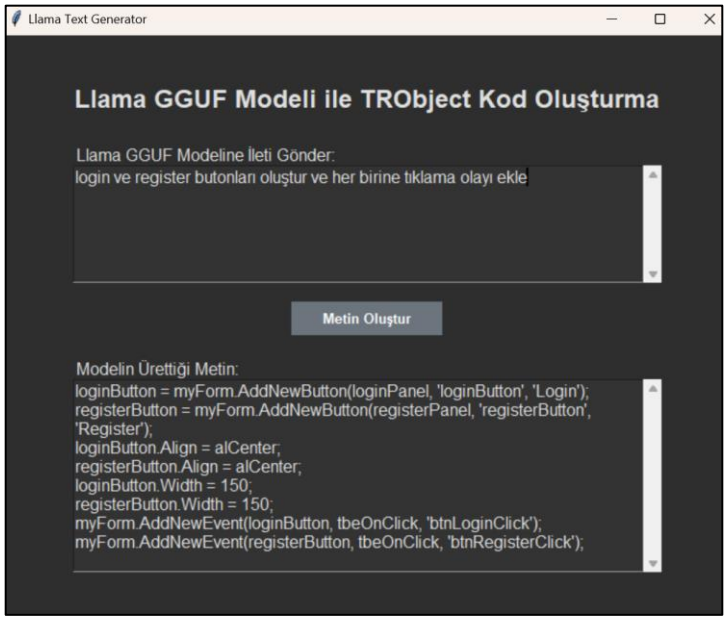
2. train/loss Grafiği

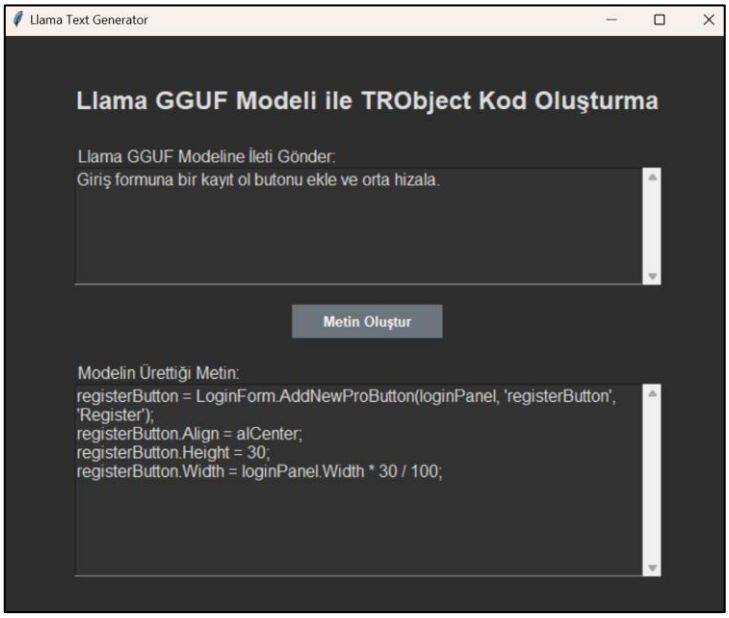
- **Açıklama:** Bu grafik, modelin eğitim sırasında her iterasyonda hesaplanan kayıp değerlerinin zaman içindeki değişimini gösterir. Yatay eksende iterasyon sayısı (batch sayısı), dikey eksende ise kayıp değeri bulunur. Grafik, eğitim sürecinde kayıp değerlerinin genellikle azaldığını, ancak belirli noktalarda dalgalanmalar olabileceğini gösterir. Bu dalgalanmalar, modelin veriye uyum sağlama sürecini yansıtır.

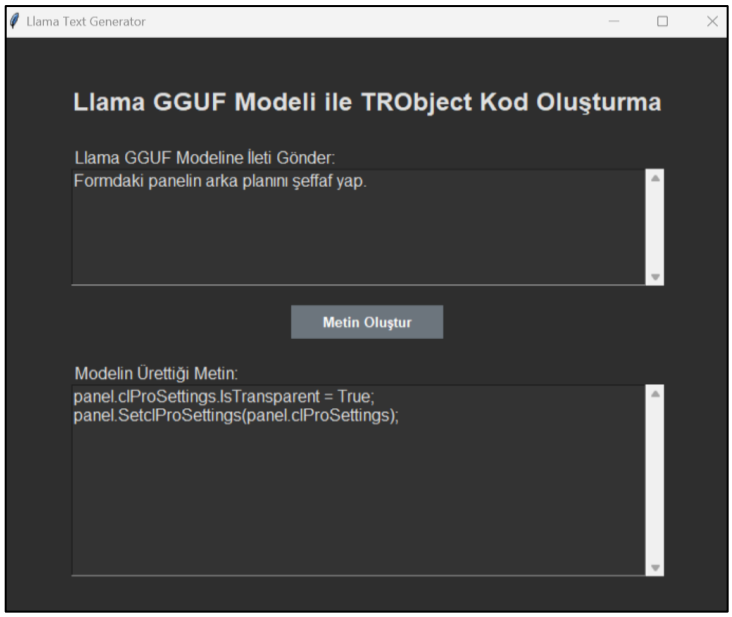
9- Modelin Ürettiği Çıktılar

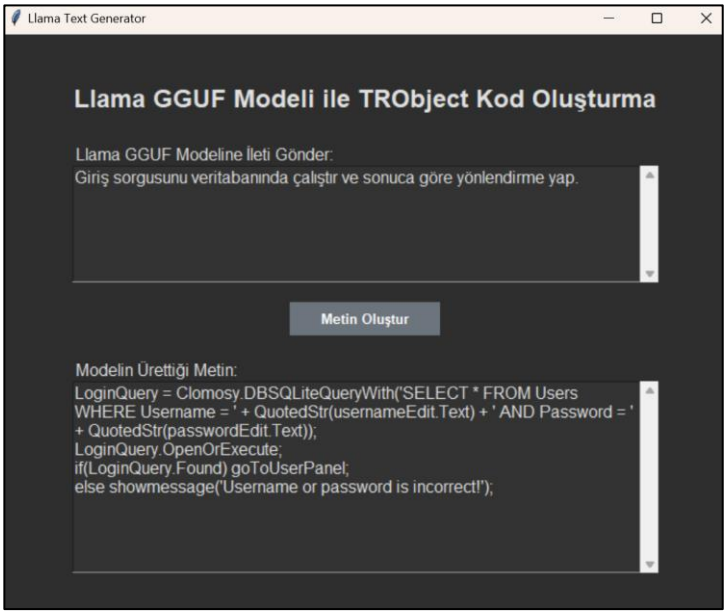
Bu bölümde, modelin çeşitli girdilere karşılık olarak ürettiği örnek çıktılar sunulmuştur. Bu örnekler, modelin metin üretme yeteneklerini, doğruluğunu ve çeşitliliğini göstermektedir. Aşağıda, farklı senaryolar için modelin verdiği yanıtlar yer almaktadır. Her bir örnek, belirli bir girdiye dayalı olarak modelin nasıl bir sonuç ürettiğini ortaya koymaktadır.

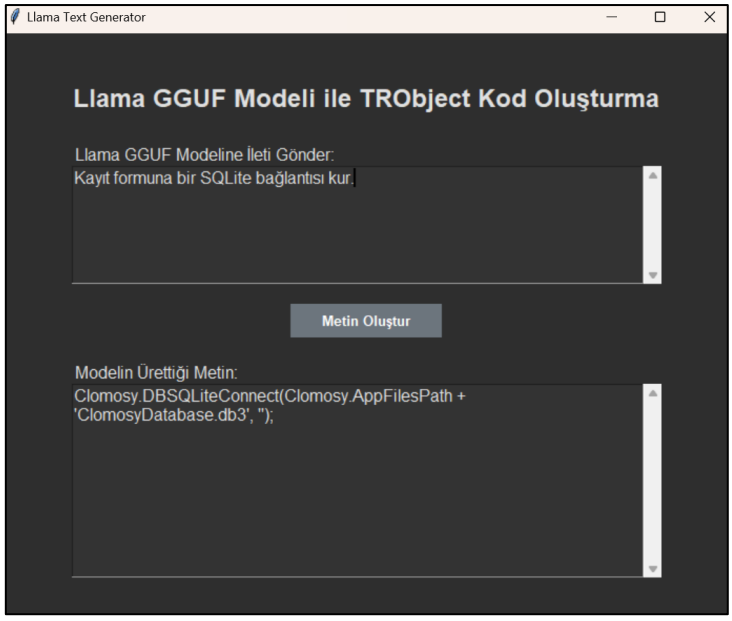
Her örnekte önce modelin aldığı girdi metni, ardından bu girdiye karşılık gelen çıktı metni sunulmuştur. Bu sayede, modelin gerçek dünyada nasıl performans gösterdiği hakkında bir fikir edinilebilir.

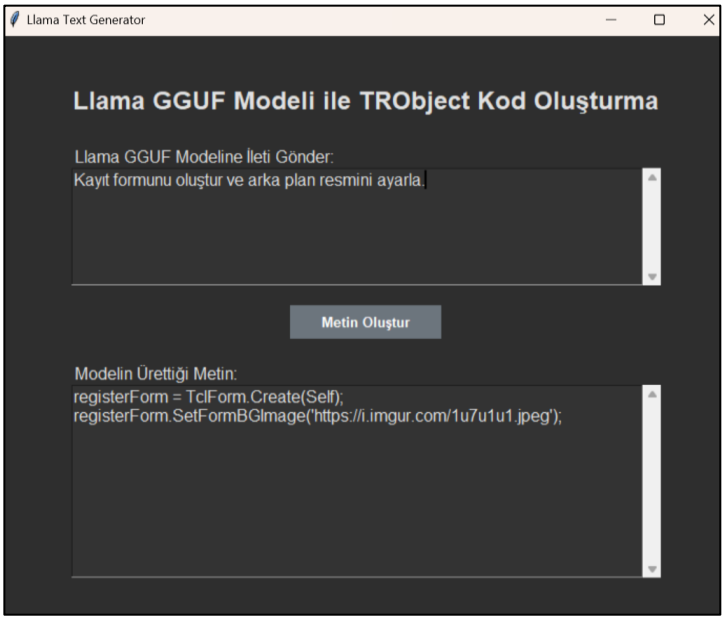

Llama Text Generator
Llama GGUF Modeli ile TRObject Kod Oluşturma
Llama GGUF Modeline İleti Gönder:
login ve register butonları oluştur ve her birine tıklama olayı ekle
Metin Oluştur
Modelin Ürettiği Metin:
loginButton = myForm.AddNewButton(loginPanel, 'loginButton', 'Login');
registerButton = myForm.AddNewButton(registerPanel, 'registerButton',
'Register');
loginButton.Align = alCenter;
registerButton.Align = alCenter;
loginButton.Width = 150;
registerButton.Width = 150;
myForm.AddNewEvent(loginButton, tbeOnClick, 'btnLoginClick');
myForm.AddNewEvent(registerButton, tbeOnClick, 'btnRegisterClick');


Llama Text Generator
Llama GGUF Modeli ile TRObject Kod Oluşturma
Llama GGUF Modeline İleti Gönder:
Giriş formuna bir kayıt ol butonu ekle ve orta hizala.
Metin Oluştur
Modelin Ürettiği Metin:
registerButton = LoginForm.AddNewProButton(loginPanel, 'registerButton',
'Register');
registerButton.Align = alCenter;
registerButton.Height = 30;
registerButton.Width = loginPanel.Width * 30 / 100;


Llama Text Generator
Llama GGUF Modeli ile TRObject Kod Oluşturma
Llama GGUF Modeline İleti Gönder:
Formdaki panelin arka planını şeffaf yap.
Metin Oluştur
Modelin Ürettiği Metin:
panel.clProSettings.IsTransparent = True;
panel.SetclProSettings(panel.clProSettings);


Llama Text Generator
Llama GGUF Modeli ile TRObject Kod Oluşturma
Llama GGUF Modeline İleti Gönder:
Giriş sorgusunu veritabanında çalıştır ve sonuca göre yönlendirme yap.
Metin Oluştur
Modelin Ürettiği Metin:
LoginQuery = Clomosy.DBSQLiteQueryWith('SELECT * FROM Users
WHERE Username = ' + QuotedStr(usernameEdit.Text) + ' AND Password = '
+ QuotedStr(passwordEdit.Text));
LoginQuery.OpenOrExecute;
if(LoginQuery.Found) goToUserPanel;
else showmessage('Username or password is incorrect!');


Llama Text Generator
Llama GGUF Modeli ile TRObject Kod Oluşturma
Llama GGUF Modeline İleti Gönder:
Kayıt formuna bir SQLite bağlantısı kur.
Metin Oluştur
Modelin Ürettiği Metin:
Clomosy.DBSQLiteConnect(Clomosy.AppFilesPath +
'ClomosyDatabase.db3', '');


Llama Text Generator
Llama GGUF Modeli ile TRObject Kod Oluşturma
Llama GGUF Modeline İleti Gönder:
Kayıt formunu oluştur ve arka plan resmini ayarla.
Metin Oluştur
Modelin Ürettiği Metin:
registerForm = TclForm.Create(Self);
registerForm.SetFormBGImage('https://i.imgur.com/1u7u1u1.jpeg');