# **Model Training Documentation**

This documentation explains the steps to train a language model using the provided Python scripts. The model is designed to process and generate Uzbek text using a custom tokenizer and a GPT-2-based architecture. We tried to train with at least 5 epochs with a full dataset, but training time was 48 hours and we did not have the time and resources to train the model. So we ended up training 2 epochs with only 5000 samples of data which was almost nothing. Even though the result was a very inaccurate model, it was working better than expected. After training we forgot to take a screenshot of the training, but we saw the eval\_loss being 10.09 percent and accuracy around 60 percent.

**Students**: Bokiev Nodirbek 20212075, Davronov Javokhir 20202548

## **Table of Contents**

1. **Tokenizer Training**
2. **Dataset Tokenization**
3. **Model Training**
4. **Evaluation and Text Generation**

## **1. Tokenizer Training**

The script tokenizer.py is used to train and save a custom tokenizer using a Byte-Pair Encoding (BPE) model. It processes Uzbek text data to create a tokenizer tailored to the dataset.

tokenizer.py:

from tokenizers import Tokenizer, models, trainers, pre\_tokenizers, processors

from datasets import load\_dataset

import os

# Paths

TOKENIZER\_SAVE\_PATH = "./tokenizer"

def train\_and\_save\_tokenizer():

# Load the 'lat' split

dataset = load\_dataset("tahrirchi/uz-books", split="lat")

# Extract all text

def get\_texts():

for sample in dataset:

yield sample["text"]

# Initialize the tokenizer

tokenizer = Tokenizer(models.BPE())

tokenizer.pre\_tokenizer = pre\_tokenizers.Whitespace()

# Train the tokenizer

trainer = trainers.BpeTrainer(special\_tokens=["<s>", "</s>", "<pad>", "<unk>", "<mask>"])

tokenizer.train\_from\_iterator(get\_texts(), trainer=trainer)

# Post-processing for GPT-like models

tokenizer.post\_processor = processors.TemplateProcessing(

single="<s> $A </s>",

pair="<s> $A </s> </s> $B </s>",

special\_tokens=[("<s>", 0), ("</s>", 1)]

)

# Save the tokenizer

os.makedirs(TOKENIZER\_SAVE\_PATH, exist\_ok=True)

tokenizer.save(f"{TOKENIZER\_SAVE\_PATH}/tokenizer.json")

print(f"Tokenizer saved at {TOKENIZER\_SAVE\_PATH}/tokenizer.json")

if \_\_name\_\_ == "\_\_main\_\_":

train\_and\_save\_tokenizer()

### **Code Overview**

* **Dataset Loading**: Loads the lat split of the dataset tahrirchi/uz-books from Hugging Face.
* **Tokenizer Initialization**: Uses BPE for efficient tokenization.
* **Training**: Trains the tokenizer on the text data, with special tokens like <s>, </s>, <pad>, <unk>, and <mask>.
* **Post-Processing**: Adds a template for GPT-like input-output processing.
* **Saving**: Saves the trained tokenizer to ./tokenizer/tokenizer.json.

### **Run Instructions**

python tokenizer.py

**2. Dataset Tokenization**

The script dataset.py tokenizes the dataset using the trained tokenizer and prepares it for model training.

dataset.py:

from transformers import PreTrainedTokenizerFast

from datasets import load\_dataset

import os

# Paths

TOKENIZER\_PATH = "./tokenizer/tokenizer.json"

OUTPUT\_DATASET\_PATH = "./tokenized\_dataset"

def tokenize\_dataset():

# Load tokenizer

tokenizer = PreTrainedTokenizerFast(tokenizer\_file=TOKENIZER\_PATH)

# Add a padding token if not already present

if tokenizer.pad\_token is None:

tokenizer.add\_special\_tokens({'pad\_token': '[PAD]'})

# Load the 'lat' split

dataset = load\_dataset("tahrirchi/uz-books", split="lat")

# Tokenization function

def tokenize\_function(examples):

# Tokenize and add padding

tokenized = tokenizer(

examples["text"],

truncation=True,

max\_length=512,

padding="max\_length"

)

tokenized["labels"] = tokenized["input\_ids"].copy() # Add labels

return tokenized

# Tokenize the dataset with high parallelism

tokenized\_dataset = dataset.map(

tokenize\_function,

batched=True,

batch\_size=1000, # Process 1000 examples at a time

num\_proc=12, # Use 12 parallel processes (adjust based on CPU threads)

remove\_columns=["text"]

)

# Save the tokenized dataset

os.makedirs(OUTPUT\_DATASET\_PATH, exist\_ok=True)

tokenized\_dataset.save\_to\_disk(OUTPUT\_DATASET\_PATH)

print(f"Tokenized dataset saved at {OUTPUT\_DATASET\_PATH}")

if \_\_name\_\_ == "\_\_main\_\_":

tokenize\_dataset()

### **Code Overview**

* **Tokenizer Loading**: Loads the tokenizer from ./tokenizer/tokenizer.json.
* **Padding Token**: Ensures a padding token ([PAD]) exists.
* **Dataset Loading**: Loads the lat split of the tahrirchi/uz-books dataset.
* **Tokenization**: Processes the text into token IDs with truncation and padding.
* **Parallel Processing**: Utilizes up to 12 processes for faster tokenization.
* **Saving**: Saves the tokenized dataset to ./tokenized\_dataset.

### **Run Instructions**

python dataset.py

## **3. Model Training**

The script train.py trains a GPT-2-like model using the tokenized dataset.

train.py:

from transformers import GPT2Config, GPT2LMHeadModel, Trainer, TrainingArguments

from datasets import load\_from\_disk

# Paths

TOKENIZED\_DATASET\_PATH = "./tokenized\_dataset"

OUTPUT\_MODEL\_PATH = "./trained\_model"

def train\_model():

# Load tokenized dataset

dataset = load\_from\_disk(TOKENIZED\_DATASET\_PATH)

# Split dataset into train and validation sets

dataset = dataset.train\_test\_split(test\_size=0.1)

# Model configuration

config = GPT2Config(

vocab\_size=50257,

n\_positions=1024,

n\_ctx=1024,

n\_embd=492,

n\_layer=12,

n\_head=12,

)

model = GPT2LMHeadModel(config)

# Training arguments

training\_args = TrainingArguments(

output\_dir=OUTPUT\_MODEL\_PATH,

overwrite\_output\_dir=True,

num\_train\_epochs=2, # Fewer epochs

per\_device\_train\_batch\_size=64, # Optimize batch size

gradient\_accumulation\_steps=4,

warmup\_steps=500,

learning\_rate=5e-4,

fp16=True,

evaluation\_strategy="epoch",

save\_strategy="epoch",

logging\_dir="./logs",

save\_total\_limit=2,

)

# Initialize Trainer

trainer = Trainer(

model=model,

args=training\_args,

train\_dataset=dataset["train"],

eval\_dataset=dataset["test"],

)

trainer.train()

# Save the model

trainer.save\_model(OUTPUT\_MODEL\_PATH)

print(f"Model saved at {OUTPUT\_MODEL\_PATH}")

if \_\_name\_\_ == "\_\_main\_\_":

train\_model()

### **Code Overview**

* **Dataset Loading**: Loads the tokenized dataset from ./tokenized\_dataset.
* **Dataset Splitting**: Splits the dataset into training (90%) and validation (10%).
* **Model Configuration**: Sets up GPT-2 with 492 embedding dimensions, 12 layers, and 12 attention heads.
* **Training Arguments**:
  + Batch size: 64 (with gradient accumulation for effective size of 256).
  + Learning rate: 0.0005.
  + Epochs: 2 (can be adjusted for larger datasets).
  + Evaluation and saving after every epoch.
  + Mixed precision (fp16) for faster training.
* **Trainer Initialization**: Uses Hugging Face's Trainer to manage training.
* **Model Saving**: Saves the trained model to ./trained\_model.

### **Run Instructions**

python train.py

## **4. Evaluation and Text Generation**

The script evaluate.py evaluates the trained model and generates Uzbek text based on a given prompt.

evaluate.py:

from transformers import GPT2LMHeadModel

from tokenizers import Tokenizer

import torch

# Load the custom tokenizer

tokenizer = Tokenizer.from\_file("./tokenizer/tokenizer.json")

# Load the GPT model

model = GPT2LMHeadModel.from\_pretrained('./trained\_model')

# Move model to GPU if available

device = "cuda" if torch.cuda.is\_available() else "cpu"

model.to(device)

# Define the Uzbek prompt

prompt = "Sen kimsan?"

# Tokenize the input

inputs = tokenizer.encode(prompt).ids # Use custom tokenizer to encode the prompt

# Convert to tensor and move to device

inputs = torch.tensor(inputs).unsqueeze(0).to(device)

# Create attention mask

attention\_mask = torch.ones(inputs.shape, device=device)

# Generate text with sampling and other parameters

output = model.generate(

inputs,

max\_length=100,

num\_return\_sequences=1,

temperature=0.7,

attention\_mask=attention\_mask,

do\_sample=True,

top\_k=50,

top\_p=0.95,

repetition\_penalty=2.0

)

# Convert tensor to list before decoding

generated\_text = tokenizer.decode(output[0].cpu().tolist(), skip\_special\_tokens=True)

# Print the generated text

print(generated\_text)

### **Code Overview**

* **Tokenizer Loading**: Loads the custom tokenizer from ./tokenizer/tokenizer.json.
* **Model Loading**: Loads the trained GPT model from ./trained\_model.
* **Prompt**: Accepts a user-defined Uzbek text prompt.
* **Text Generation**:
  + Uses top-k sampling and nucleus sampling (top-p) for diversity.
  + Applies a repetition penalty to reduce repetitive outputs.
  + Limits the output to 100 tokens.
* **Output**: Decodes and prints the generated text.

### **Run Instructions**

python evaluate.py

## **Key File Locations**

| **File/Directory** | **Description** |
| --- | --- |
| ./tokenizer | Contains the saved tokenizer. |
| ./tokenized\_dataset | Tokenized dataset for model training. |
| ./trained\_model | Trained GPT-2 model files. |
| ./logs | Training logs and metrics. |

## 

## 

## **Example Workflow**

**Train the Tokenizer**:  
 python tokenizer.py

**Tokenize the Dataset**:  
 python dataset.py

**Train the Model**:  
 python train.py

**Evaluate the Model**:  
 python evaluate.py

## **Notes**

* **Dataset Requirements**: Ensure the tahrirchi/uz-books dataset is available on Hugging Face.
* **Hardware**: GPU is recommended for model training and evaluation.
* **Parallelism**: Adjust num\_proc in dataset.py based on available CPU cores.

By following these steps, you can train and evaluate a language model for Uzbek text using custom tokenization and GPT-2-based architecture.

Screenshots from the process





