

Introduction

Let us build a Small Language Model (SLM) from scratch. We will try to keep the parameter size to 50-60 million.

Our goal is to generate creative and coherent text based on the input data.

Step 1: Import the Dataset

TinyStories is a synthetic dataset of short stories that only contain words that a typical 3 to 4-year-olds usually understand, generated by GPT-3.5 and GPT-4. We can get it from HuggingFace.

```
!pip install datasets
```

```
Requirement already satisfied: datasets in /usr/local/lib/python3.11/dist-packages (4.0.0)
Requirement already satisfied: filelock in /usr/local/lib/python3.11/dist-packages (from datasets) (3.18.0)
Requirement already satisfied: numpy>=1.17 in /usr/local/lib/python3.11/dist-packages (from datasets) (2.0.2)
Requirement already satisfied: pyarrow>=15.0.0 in /usr/local/lib/python3.11/dist-packages (from datasets) (18.1.0)
Requirement already satisfied: dill<0.3.9,>=0.3.0 in /usr/local/lib/python3.11/dist-packages (from datasets) (0.3.8)
Requirement already satisfied: pandas in /usr/local/lib/python3.11/dist-packages (from datasets) (2.2.2)
Requirement already satisfied: requests>=2.32.2 in /usr/local/lib/python3.11/dist-packages (from datasets) (2.32.3)
Requirement already satisfied: tqdm>=4.66.3 in /usr/local/lib/python3.11/dist-packages (from datasets) (4.67.1)
Requirement already satisfied: xxhash in /usr/local/lib/python3.11/dist-packages (from datasets) (3.5.0)
Requirement already satisfied: multiprocess<0.70.17 in /usr/local/lib/python3.11/dist-packages (from datasets) (0.70.16)
Requirement already satisfied: fsspec<=2025.3.0,>=2023.1.0 in /usr/local/lib/python3.11/dist-packages (from fsspec[http]<=2025.3.0,
Requirement already satisfied: huggingface-hub>=0.24.0 in /usr/local/lib/python3.11/dist-packages (from datasets) (0.34.4)
Requirement already satisfied: packaging in /usr/local/lib/python3.11/dist-packages (from datasets) (25.0)
Requirement already satisfied: pyyaml>=5.1 in /usr/local/lib/python3.11/dist-packages (from datasets) (6.0.2)
Requirement already satisfied: aiohttp!=4.0.0a0,!4.0.0a1 in /usr/local/lib/python3.11/dist-packages (from fsspec[http]<=2025.3.0,>
Requirement already satisfied: typing-extensions>=3.7.4.3 in /usr/local/lib/python3.11/dist-packages (from huggingface-hub>=0.24.0->
Requirement already satisfied: hf-xet<2.0.0,>=1.1.3 in /usr/local/lib/python3.11/dist-packages (from huggingface-hub>=0.24.0->dataset
Requirement already satisfied: charset-normalizer<4,>=2 in /usr/local/lib/python3.11/dist-packages (from requests>=2.32.2->datasets
Requirement already satisfied: idna<4,>=2.5 in /usr/local/lib/python3.11/dist-packages (from requests>=2.32.2->datasets) (3.10)
Requirement already satisfied: urllib3<3,>=1.21.1 in /usr/local/lib/python3.11/dist-packages (from requests>=2.32.2->datasets) (2.5
Requirement already satisfied: certifi>=2017.4.17 in /usr/local/lib/python3.11/dist-packages (from requests>=2.32.2->datasets) (2025
Requirement already satisfied: python-dateutil>=2.8.2 in /usr/local/lib/python3.11/dist-packages (from pandas->datasets) (2.9.0.post
Requirement already satisfied: pytz>=2020.1 in /usr/local/lib/python3.11/dist-packages (from pandas->datasets) (2025.2)
Requirement already satisfied: tzdata>=2022.7 in /usr/local/lib/python3.11/dist-packages (from pandas->datasets) (2025.2)
Requirement already satisfied: aiohappyeyeballs>=2.5.0 in /usr/local/lib/python3.11/dist-packages (from aiohttp!=4.0.0a0,!4.0.0a1->
Requirement already satisfied: aiosignal>=1.4.0 in /usr/local/lib/python3.11/dist-packages (from aiohttp!=4.0.0a0,!4.0.0a1->fsspec
Requirement already satisfied: attrs>=17.3.0 in /usr/local/lib/python3.11/dist-packages (from aiohttp!=4.0.0a0,!4.0.0a1->fsspec[htt
Requirement already satisfied: frozenlist>=1.1.1 in /usr/local/lib/python3.11/dist-packages (from aiohttp!=4.0.0a0,!4.0.0a1->fsspec
Requirement already satisfied: multidict<7.0,>=4.5 in /usr/local/lib/python3.11/dist-packages (from aiohttp!=4.0.0a0,!4.0.0a1->fsspec
Requirement already satisfied: propcache>=0.2.0 in /usr/local/lib/python3.11/dist-packages (from aiohttp!=4.0.0a0,!4.0.0a1->fsspec
Requirement already satisfied: yarl<2.0,>=1.17.0 in /usr/local/lib/python3.11/dist-packages (from aiohttp!=4.0.0a0,!4.0.0a1->fsspec
Requirement already satisfied: six>=1.5 in /usr/local/lib/python3.11/dist-packages (from python-dateutil>=2.8.2->pandas->datasets)
```

```
pip install -U datasets
```


```
Requirement already satisfied: datasets in /usr/local/lib/python3.11/dist-packages (4.0.0)
Requirement already satisfied: filelock in /usr/local/lib/python3.11/dist-packages (from datasets) (3.18.0)
Requirement already satisfied: numpy>=1.17 in /usr/local/lib/python3.11/dist-packages (from datasets) (2.0.2)
Requirement already satisfied: pyarrow>=15.0.0 in /usr/local/lib/python3.11/dist-packages (from datasets) (18.1.0)
Requirement already satisfied: dill<0.3.9,>=0.3.0 in /usr/local/lib/python3.11/dist-packages (from datasets) (0.3.8)
Requirement already satisfied: pandas in /usr/local/lib/python3.11/dist-packages (from datasets) (2.2.2)
Requirement already satisfied: requests>=2.32.2 in /usr/local/lib/python3.11/dist-packages (from datasets) (2.32.3)
Requirement already satisfied: tqdm>=4.66.3 in /usr/local/lib/python3.11/dist-packages (from datasets) (4.67.1)
Requirement already satisfied: xxhash in /usr/local/lib/python3.11/dist-packages (from datasets) (3.5.0)
Requirement already satisfied: multiprocess<0.70.17 in /usr/local/lib/python3.11/dist-packages (from datasets) (0.70.16)
Requirement already satisfied: fsspec<=2025.3.0,>=2023.1.0 in /usr/local/lib/python3.11/dist-packages (from fsspec[http]<=2025.3.0,
Requirement already satisfied: huggingface-hub>=0.24.0 in /usr/local/lib/python3.11/dist-packages (from datasets) (0.34.3)
Requirement already satisfied: packaging in /usr/local/lib/python3.11/dist-packages (from datasets) (25.0)
Requirement already satisfied: pyyaml>=5.1 in /usr/local/lib/python3.11/dist-packages (from datasets) (6.0.2)
Requirement already satisfied: aiohttp!=4.0.0a0,!4.0.0a1 in /usr/local/lib/python3.11/dist-packages (from fsspec[http]<=2025.3.0,>
Requirement already satisfied: typing-extensions>=3.7.4.3 in /usr/local/lib/python3.11/dist-packages (from huggingface-hub>=0.24.0->
Requirement already satisfied: hf-xet<2.0.0,>=1.1.3 in /usr/local/lib/python3.11/dist-packages (from huggingface-hub>=0.24.0->dataset
Requirement already satisfied: charset-normalizer<4,>=2 in /usr/local/lib/python3.11/dist-packages (from requests>=2.32.2->datasets
Requirement already satisfied: idna<4,>=2.5 in /usr/local/lib/python3.11/dist-packages (from requests>=2.32.2->datasets) (3.10)
Requirement already satisfied: urllib3<3,>=1.21.1 in /usr/local/lib/python3.11/dist-packages (from requests>=2.32.2->datasets) (2.5
Requirement already satisfied: certifi>=2017.4.17 in /usr/local/lib/python3.11/dist-packages (from requests>=2.32.2->datasets) (2025
Requirement already satisfied: python-dateutil>=2.8.2 in /usr/local/lib/python3.11/dist-packages (from pandas->datasets) (2.9.0.post
Requirement already satisfied: pytz>=2020.1 in /usr/local/lib/python3.11/dist-packages (from pandas->datasets) (2025.2)
Requirement already satisfied: tzdata>=2022.7 in /usr/local/lib/python3.11/dist-packages (from pandas->datasets) (2025.2)
Requirement already satisfied: aiohappyeyeballs>=2.5.0 in /usr/local/lib/python3.11/dist-packages (from aiohttp!=4.0.0a0,!4.0.0a1->
Requirement already satisfied: aiosignal>=1.4.0 in /usr/local/lib/python3.11/dist-packages (from aiohttp!=4.0.0a0,!4.0.0a1->fsspec
Requirement already satisfied: attrs>=17.3.0 in /usr/local/lib/python3.11/dist-packages (from aiohttp!=4.0.0a0,!4.0.0a1->fsspec[htt
Requirement already satisfied: frozenlist>=1.1.1 in /usr/local/lib/python3.11/dist-packages (from aiohttp!=4.0.0a0,!4.0.0a1->fsspec
Requirement already satisfied: multidict<7.0,>=4.5 in /usr/local/lib/python3.11/dist-packages (from aiohttp!=4.0.0a0,!4.0.0a1->fsspec
Requirement already satisfied: propcache>=0.2.0 in /usr/local/lib/python3.11/dist-packages (from aiohttp!=4.0.0a0,!4.0.0a1->fsspec
Requirement already satisfied: yarl<2.0,>=1.17.0 in /usr/local/lib/python3.11/dist-packages (from aiohttp!=4.0.0a0,!4.0.0a1->fsspec
Requirement already satisfied: six>=1.5 in /usr/local/lib/python3.11/dist-packages (from python-dateutil>=2.8.2->pandas->datasets)
```

```

from datasets import load_dataset

ds = load_dataset("roneneldan/TinyStories")

```

 /usr/local/lib/python3.11/dist-packages/huggingface_hub/utils/_auth.py:94: UserWarning:
 The secret `HF_TOKEN` does not exist in your Colab secrets.
 To authenticate with the Hugging Face Hub, create a token in your settings tab (<https://huggingface.co/settings/tokens>), set it as :
 You will be able to reuse this secret in all of your notebooks.
 Please note that authentication is recommended but still optional to access public models or datasets.
 warnings.warn(

README.md:	1.06k/? [00:00<00:00, 113kB/s]	
(...)-00000-of-00004-2d5a1467fff1081b.parquet:	100%	249M/249M [00:01<00:00, 252MB/s]
(...)-00001-of-00004-5852b56a2bd28fd9.parquet:	100%	248M/248M [00:00<00:00, 272MB/s]
(...)-00002-of-00004-a26307300439e943.parquet:	100%	246M/246M [00:00<00:00, 272MB/s]
(...)-00003-of-00004-d243063613e5a057.parquet:	100%	248M/248M [00:00<00:00, 272MB/s]
(...)-00000-of-00001-869c898b519ad725.parquet:	100%	9.99M/9.99M [00:00<00:00, 211MB/s]
Generating train split:	100%	2119719/2119719 [00:06<00:00, 347903.85 examples/s]
Generating validation split:	100%	21990/21990 [00:00<00:00, 301590.62 examples/s]

✓ Step 2: Tokenize the Dataset

In this step, we will do the following:

- (1) Tokenize the dataset into tokenIDs.
- (2) Create a file called "train.bin" and "validation.bin" where we will store the tokenIDs from the entire dataset.
- (3) We make sure the tokenIDs are stored on a disk, rather than on the RAM for efficient computations.

```

!pip install tiktoken
import tiktoken
import os
import numpy as np
from tqdm.auto import tqdm

enc = tiktoken.get_encoding("gpt2")

# Some functions from https://github.com/karpathy/nanoGPT/blob/master/data/openwebtext/prepare.py

def process(example):
    ids = enc.encode_ordinary(example['text']) # encode_ordinary ignores any special tokens
    out = {'ids': ids, 'len': len(ids)}
    return out

if not os.path.exists("train.bin"):
    tokenized = ds.map(
        process,
        remove_columns=['text'],
        desc="tokenizing the splits",
        num_proc=8,
    )

# concatenate all the ids in each dataset into one large file we can use for training
for split, dset in tokenized.items():
    arr_len = np.sum(dset['len'], dtype=np.uint64)
    filename = f'{split}.bin'
    dtype = np.uint16 # (can do since enc.max_token_value == 50256 is < 2**16)
    arr = np.memmap(filename, dtype=dtype, mode='w+', shape=(arr_len,))
    total_batches = 1024

    idx = 0
    for batch_idx in tqdm(range(total_batches), desc=f'writing {filename}'):
        # Batch together samples for faster write
        batch = dset.shard(num_shards=total_batches, index=batch_idx, contiguous=True).with_format('numpy')
        arr_batch = np.concatenate(batch['ids'])
        # Write into mmap
        arr[idx : idx + len(arr_batch)] = arr_batch
        idx += len(arr_batch)
    arr.flush()

```

```

Requirement already satisfied: tiktoken in /usr/local/lib/python3.11/dist-packages (0.10.0)
Requirement already satisfied: regex<=2022.1.18 in /usr/local/lib/python3.11/dist-packages (from tiktoken) (2024.11.6)
Requirement already satisfied: requests>=2.26.0 in /usr/local/lib/python3.11/dist-packages (from tiktoken) (2.32.3)
Requirement already satisfied: charset-normalizer<4,>=2 in /usr/local/lib/python3.11/dist-packages (from requests>=2.26.0->tiktoken) (3.10)
Requirement already satisfied: idna<4,>=2.5 in /usr/local/lib/python3.11/dist-packages (from requests>=2.26.0->tiktoken) (3.10)
Requirement already satisfied: urllib3<3,>=1.21.1 in /usr/local/lib/python3.11/dist-packages (from requests>=2.26.0->tiktoken) (2.5)
Requirement already satisfied: certifi>=2017.4.17 in /usr/local/lib/python3.11/dist-packages (from requests>=2.26.0->tiktoken) (2025.1.1)

tokenizing the splits (num_proc=8): 100% 2119719/2119719 [01:05<00:00, 8429.66 examples/s]

tokenizing the splits (num_proc=8): 100% 21990/21990 [00:01<00:00, 27577.46 examples/s]

writing train.bin: 100% 1024/1024 [14:06<00:00, 1.21it/s]

writing validation.bin: 100% 1024/1024 [00:11<00:00, 94.41it/s]

```

Step 3: Create Input-Output batches for the dataset

```

# Some functions from https://github.com/karpathy/nanoGPT/blob/master/train.py with slight modifications
#block size = context window
def get_batch(split):
    # We recreate np.memmap every batch to avoid a memory leak, as per
    # https://stackoverflow.com/questions/45132940/numpy-memmap-memory-usage-want-to-iterate-once/61472122#61472122
    if split == 'train':
        data = np.memmap('train.bin', dtype=np.uint16, mode='r')
    else:
        data = np.memmap('validation.bin', dtype=np.uint16, mode='r')
    ix = torch.randint(len(data) - block_size, (batch_size,))
    x = torch.stack([torch.from_numpy((data[i:i+block_size]).astype(np.int64)) for i in ix])
    y = torch.stack([torch.from_numpy((data[i+1:i+1+block_size]).astype(np.int64)) for i in ix])
    if device_type == 'cuda':
        # pin arrays x,y, which allows us to move them to GPU asynchronously (non_blocking=True)
        x, y = x.pin_memory().to(device, non_blocking=True), y.pin_memory().to(device, non_blocking=True)
    else:
        x, y = x.to(device), y.to(device)
    return x, y

```

Step 4: Define the SLM Model Architecture

```

import torch
import torch.nn as nn
import torch.nn.functional as F
import math
from dataclasses import dataclass
import numpy as np
from tqdm.auto import tqdm
from contextlib import nullcontext
import os

class LayerNorm(nn.Module):
    def __init__(self, ndim, bias):
        super().__init__()
        self.weight = nn.Parameter(torch.ones(ndim))
        self.bias = nn.Parameter(torch.zeros(ndim)) if bias else None
    def forward(self, x):
        return F.layer_norm(x, self.weight.shape, self.weight, self.bias, 1e-5)

class CausalSelfAttention(nn.Module):
    def __init__(self, config):
        super().__init__()
        assert config.n_embd % config.n_head == 0
        self.c_attn = nn.Linear(config.n_embd, 3 * config.n_embd, bias=config.bias)
        self.c_proj = nn.Linear(config.n_embd, config.n_embd, bias=config.bias)
        self.attn_dropout = nn.Dropout(config.dropout)
        self.resid_dropout = nn.Dropout(config.dropout)
        self.n_head = config.n_head
        self.n_embd = config.n_embd
        self.flash = hasattr(F, 'scaled_dot_product_attention')
        if not self.flash:
            self.register_buffer("bias", torch.tril(torch.ones(config.block_size, config.block_size))
                                .view(1, 1, config.block_size, config.block_size))

    def forward(self, x):
        B, T, C = x.size()
        q, k, v = self.c_attn(x).split(self.n_embd, dim=2)
        k = k.view(B, T, self.n_head, C // self.n_head).transpose(1, 2)
        q = q.view(B, T, self.n_head, C // self.n_head).transpose(1, 2)
        v = v.view(B, T, self.n_head, C // self.n_head).transpose(1, 2)

```

```

    if self.flash:
        y = F.scaled_dot_product_attention(q, k, v, attn_mask=None, dropout_p=self.attn_dropout.p if self.training else 0.0, is_cau:
    else:
        att = (q @ k.transpose(-2, -1)) * (1.0 / math.sqrt(k.size(-1)))
        att = att.masked_fill(self.bias[:, :, :T, :T] == 0, float('-inf'))
        att = F.softmax(att, dim=-1)
        att = self.attn_dropout(att)
        y = att @ v

    y = y.transpose(1, 2).contiguous().view(B, T, C)
    y = self.resid_dropout(self.c_proj(y))
    return y

class MLP(nn.Module):
    def __init__(self, config):
        super().__init__()
        self.c_fc = nn.Linear(config.n_embd, 4 * config.n_embd, bias=config.bias)
        self.gelu = nn.GELU()
        self.c_proj = nn.Linear(4 * config.n_embd, config.n_embd, bias=config.bias)
        self.dropout = nn.Dropout(config.dropout)
    def forward(self, x):
        return self.dropout(self.c_proj(self.gelu(self.c_fc(x))))

class Block(nn.Module):
    def __init__(self, config):
        super().__init__()
        self.ln1 = LayerNorm(config.n_embd, config.bias)
        self.attn = CausalSelfAttention(config)
        self.ln2 = LayerNorm(config.n_embd, config.bias)
        self.mlp = MLP(config)
    def forward(self, x):
        x = x + self.attn(self.ln1(x))
        x = x + self.mlp(self.ln2(x))
        return x

@dataclass
class GPTConfig:
    block_size: int
    vocab_size: int
    n_layer: int
    n_head: int
    n_embd: int
    dropout: float = 0.0
    bias: bool = True

class GPT(nn.Module):
    def __init__(self, config):
        super().__init__()
        self.config = config
        self.transformer = nn.ModuleDict(dict(
            wte=nn.Embedding(config.vocab_size, config.n_embd),
            wpe=nn.Embedding(config.block_size, config.n_embd),
            drop=nn.Dropout(config.dropout),
            h=nn.ModuleList([Block(config) for _ in range(config.n_layer)]),
            ln_f=LayerNorm(config.n_embd, config.bias),
        ))
        self.lm_head = nn.Linear(config.n_embd, config.vocab_size, bias=False)
        self.transformer.wte.weight = self.lm_head.weight # weight tying

        self.apply(self._init_weights)
        for pn, p in self.named_parameters():
            if pn.endswith('c_proj.weight'):
                nn.init.normal_(p, mean=0.0, std=0.02 / math.sqrt(2 * config.n_layer))

    def _init_weights(self, module):
        if isinstance(module, nn.Linear):
            nn.init.normal_(module.weight, mean=0.0, std=0.02)
            if module.bias is not None:
                nn.init.zeros_(module.bias)
        elif isinstance(module, nn.Embedding):
            nn.init.normal_(module.weight, mean=0.0, std=0.02)

    def forward(self, idx, targets=None):
        device = idx.device
        b, t = idx.size()
        assert t <= self.config.block_size
        pos = torch.arange(0, t, dtype=torch.long, device=device)

        tok_emb = self.transformer.wte(idx)
        pos_emb = self.transformer.wpe(pos)
        x = self.transformer.drop(tok_emb + pos_emb)

```

```

    for block in self.transformer.h:
        x = block(x)
    x = self.transformer.ln_f(x)

    if targets is not None:
        logits = self.lm_head(x)
        loss = F.cross_entropy(logits.view(-1, logits.size(-1)), targets.view(-1), ignore_index=-1)
        return logits, loss
    else:
        logits = self.lm_head(x[:, [-1], :])
        return logits, None

@torch.no_grad()
def generate(self, idx, max_new_tokens, temperature=1.0, top_k=None):
    """
    Generate tokens given a conditioning sequence.
    idx: Tensor of shape (B, T)
    """
    for _ in range(max_new_tokens):
        idx_cond = idx if idx.size(1) <= self.config.block_size else idx[:, -self.config.block_size:]
        logits, _ = self(idx_cond)
        logits = logits[:, -1, :] / temperature
        if top_k is not None:
            v, _ = torch.topk(logits, min(top_k, logits.size(-1)))
            logits[logits < v[:, [-1]]] = -float('Inf')
        probs = F.softmax(logits, dim=-1)
        idx_next = torch.multinomial(probs, num_samples=1)
        idx = torch.cat((idx, idx_next), dim=1)
    return idx

config = GPTConfig(
    vocab_size=50257,      # use the tokenizer's vocab size
    block_size=128,       # or whatever context size you're training with
    n_layer=6,
    n_head=6,
    n_embd=384,
    dropout=0.1,
    bias=True
)

model = GPT(config)

```

▼ Step 5: Define the loss function

```

def estimate_loss(model):
    out = {}
    model.eval()
    with torch.inference_mode():
        for split in ['train', 'val']:
            losses = torch.zeros(eval_iters)
            for k in range(eval_iters):
                X, Y = get_batch(split)
                with ctx:
                    logits, loss = model(X, Y)
                losses[k] = loss.item()
            out[split] = losses.mean()
    model.train()
    return out

```

▼ Step 6: Define SLM Training Configuration Part 1

```

# Training Config
import torch
from contextlib import nullcontext

learning_rate = 1e-4 #more stable training, earlier 1e-4
max_iters = 20000 #increase from 25000
warmup_steps = 1000 #smoother initial train, earlier 100
min_lr = 5e-4 #lower rate, earlier 5e-4
eval_iters = 500 # increased from 100
batch_size = 32 # changed from 16, better gradient estimate
block_size = 128 #changed from 64, capture longer range dependencies

gradient_accumulation_steps = 32 # reduced from 50

```

```

device = "cuda" if torch.cuda.is_available() else "cpu"
device_type = 'cuda' if 'cuda' in device else 'cpu' # for later use in torch.autocast
# note: float16 data type will automatically use a GradScaler

# How to use autocast https://wandb.ai/wandb_fc/tips/reports/How-To-Use-Autocast-in-PyTorch--VmlldzoymTk4NTky
#dtype = 'bfloat16' if torch.cuda.is_available() and torch.cuda.is_bf16_supported() else 'float16' # 'float32', 'bfloat16', or 'float16'
dtype = 'bfloat16' if torch.cuda.is_available() and torch.cuda.is_bf16_supported() else 'float16' # 'float32', 'bfloat16', or 'float16'
ptdtype = {'float32': torch.float32, 'bfloat16': torch.bfloat16, 'float16': torch.float16}[dtype]

ctx = nullcontext() if device_type == 'cpu' else torch.amp.autocast(device_type=device_type, dtype=ptdtype)

torch.set_default_device(device)
torch.manual_seed(42)

↩ <torch._C.Generator at 0x7d538c0d5d90>

```

✓ Step 7: Define SLM Training Configuration Part 2

```

from torch.optim.lr_scheduler import LinearLR, SequentialLR, CosineAnnealingLR

##PUT IN WEIGHT DECAY, CHANGED BETA2 to 0.95
optimizer = torch.optim.AdamW(model.parameters(), lr=learning_rate, betas=(0.9, 0.95), weight_decay=0.1, eps=1e-9) #weight decay for re

scheduler_warmup = LinearLR(optimizer, total_iters = warmup_steps) #Implement linear warmup
scheduler_decay = CosineAnnealingLR(optimizer, T_max = max_iters - warmup_steps, eta_min = min_lr) #Implement lr decay
scheduler = SequentialLR(optimizer, schedulers=[scheduler_warmup, scheduler_decay], milestones=[warmup_steps]) #Switching from warmup to

# https://stackoverflow.com/questions/72534859/is-gradscaler-necessary-with-mixed-precision-training-with-pytorch
scaler = torch.cuda.amp.GradScaler(enabled=(dtype == 'float16'))

↩ /tmp/ipython-input-2132813893.py:11: FutureWarning: `torch.cuda.amp.GradScaler(args...)` is deprecated. Please use `torch.amp.GradSc
scaler = torch.cuda.amp.GradScaler(enabled=(dtype == 'float16'))

```

✓ Step 8: Pre-train the SLM

```

best_val_loss = float('inf')
best_model_params_path = "best_model_params.pt"
train_loss_list, validation_loss_list = [], []

# Ensure model is on the correct device
model = model.to(device)

# In your training loop
for epoch in tqdm(range(max_iters)):
    if epoch % eval_iters == 0 and epoch != 0:
        # Ensure estimate_loss uses the correct device
        losses = estimate_loss(model)
        print(f"Epoch {epoch}: train loss {losses['train']:.4f}, val loss {losses['val']:.4f}")
        print(f"The current learning rate: {optimizer.param_groups[0]['lr']:.5f}")
        train_loss_list += [losses['train']]
        validation_loss_list += [losses['val']]

        if losses['val'] < best_val_loss:
            best_val_loss = losses['val']
            torch.save(model.state_dict(), best_model_params_path)

# Ensure X and y are on the correct device
X, y = get_batch("train")
X, y = X.to(device), y.to(device)

with ctx:
    logits, loss = model(X, y)
    loss = loss / gradient_accumulation_steps
    scaler.scale(loss).backward()

if ((epoch + 1) % gradient_accumulation_steps == 0) or (epoch + 1 == max_iters):
    torch.nn.utils.clip_grad_norm_(model.parameters(), max_norm=0.5)
    scaler.step(optimizer)
    scaler.update()
    optimizer.zero_grad(set_to_none=True)
    scheduler.step()

```

100% 20000/20000 [19:55<00:00, 34.78it/s]

```

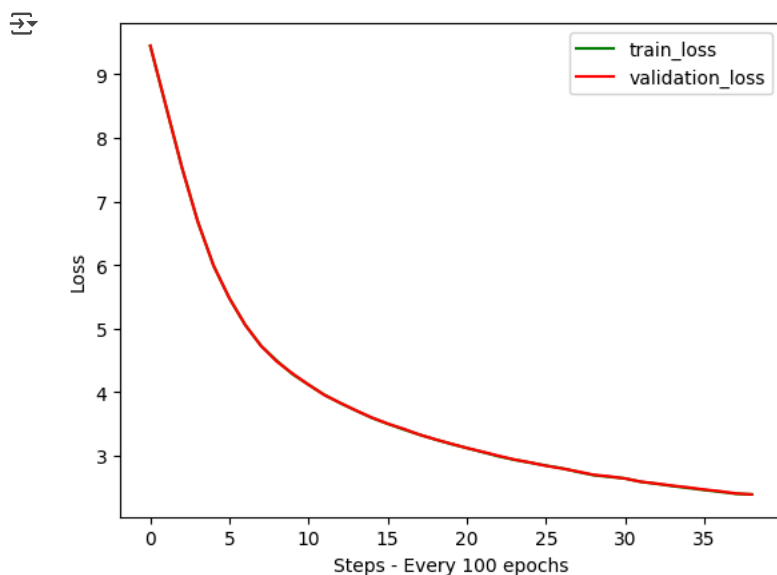
/usr/local/lib/python3.11/dist-packages/torch/optim/lr_scheduler.py:227: UserWarning: Detected call of `lr_scheduler.step()` before
warnings.warn(
Epoch 500: train loss 9.4434, val loss 9.4478
The current learning rate: 0.00007
/usr/local/lib/python3.11/dist-packages/torch/optim/lr_scheduler.py:243: UserWarning: The epoch parameter in `scheduler.step()` was
warnings.warn(EPOCH_DEPRECATION_WARNING, UserWarning)
Epoch 1000: train loss 8.4843, val loss 8.4880
The current learning rate: 0.00010
Epoch 1500: train loss 7.5319, val loss 7.5295
The current learning rate: 0.00010
Epoch 2000: train loss 6.6829, val loss 6.6796
The current learning rate: 0.00010
Epoch 2500: train loss 5.9868, val loss 5.9874
The current learning rate: 0.00011
Epoch 3000: train loss 5.4754, val loss 5.4753
The current learning rate: 0.00011
Epoch 3500: train loss 5.0568, val loss 5.0569
The current learning rate: 0.00012
Epoch 4000: train loss 4.7278, val loss 4.7268
The current learning rate: 0.00012
Epoch 4500: train loss 4.4928, val loss 4.4798
The current learning rate: 0.00013
Epoch 5000: train loss 4.2813, val loss 4.2910
The current learning rate: 0.00014
Epoch 5500: train loss 4.1174, val loss 4.1172
The current learning rate: 0.00015
Epoch 6000: train loss 3.9580, val loss 3.9559
The current learning rate: 0.00016
Epoch 6500: train loss 3.8283, val loss 3.8322
The current learning rate: 0.00018
Epoch 7000: train loss 3.7105, val loss 3.7125
The current learning rate: 0.00019
Epoch 7500: train loss 3.5964, val loss 3.6009
The current learning rate: 0.00020
Epoch 8000: train loss 3.5030, val loss 3.5054
The current learning rate: 0.00022
Epoch 8500: train loss 3.4146, val loss 3.4242
The current learning rate: 0.00024
Epoch 9000: train loss 3.3326, val loss 3.3330
The current learning rate: 0.00025
Epoch 9500: train loss 3.2573, val loss 3.2588
The current learning rate: 0.00027
Epoch 10000: train loss 3.1863, val loss 3.1901
The current learning rate: 0.00028
Epoch 10500: train loss 3.1217, val loss 3.1232
The current learning rate: 0.00030
Epoch 11000: train loss 3.0579, val loss 3.0628
The current learning rate: 0.00032
Epoch 11500: train loss 2.9917, val loss 3.0002
The current learning rate: 0.00033
Epoch 12000: train loss 2.9377, val loss 2.9414
The current learning rate: 0.00035
Epoch 12500: train loss 2.8922, val loss 2.8947
The current learning rate: 0.00036
Epoch 13000: train loss 2.8471, val loss 2.8424
The current learning rate: 0.00038
Epoch 13500: train loss 2.8041, val loss 2.8015
The current learning rate: 0.00040
Epoch 14000: train loss 2.7478, val loss 2.7539
The current learning rate: 0.00041
Epoch 14500: train loss 2.6959, val loss 2.7014
The current learning rate: 0.00042
Epoch 15000: train loss 2.6693, val loss 2.6762
The current learning rate: 0.00044
Epoch 15500: train loss 2.6446, val loss 2.6427
The current learning rate: 0.00045
Epoch 16000: train loss 2.5898, val loss 2.5931
The current learning rate: 0.00046
Epoch 16500: train loss 2.5605, val loss 2.5623
The current learning rate: 0.00047
Epoch 17000: train loss 2.5234, val loss 2.5302
The current learning rate: 0.00048
Epoch 17500: train loss 2.4941, val loss 2.5027
The current learning rate: 0.00048
Epoch 18000: train loss 2.4619, val loss 2.4696
The current learning rate: 0.00049
Epoch 18500: train loss 2.4342, val loss 2.4425
The current learning rate: 0.00049
Epoch 19000: train loss 2.4027, val loss 2.4082
The current learning rate: 0.00050
Epoch 19500: train loss 2.3933, val loss 2.3935
The current learning rate: 0.00050

```

✓ Step 9: Plot the SLM Loss Function

```
import matplotlib.pyplot as plt
train_loss_list_converted = [i.cpu().detach() for i in train_loss_list]
validation_loss_list_converted = [i.cpu().detach() for i in validation_loss_list]

plt.plot(train_loss_list_converted, 'g', label='train_loss')
plt.plot(validation_loss_list_converted, 'r', label='validation_loss')
plt.xlabel("Steps - Every 100 epochs")
plt.ylabel("Loss")
plt.legend()
plt.show()
```



✓ Step 10: Run SLM Inference on our trained model

```
#Load the model
model = GPT(config) # re-create the model with same config
device = "cuda" if torch.cuda.is_available() else "cpu"
best_model_params_path = "best_model_params.pt"
model.load_state_dict(torch.load(best_model_params_path, map_location=torch.device(device))) # load best model states
```

↩ <All keys matched successfully>

```
sentence = "Once upon a time there was a pumpkin."
context = (torch.tensor(enc.encode_ordinary(sentence)).unsqueeze(dim = 0))
y = model.generate(context, 200)
print(enc.decode(y.squeeze().tolist()))
```

↩ Once upon a time there was a pumpkin. It was very special. The pumpkin wanted to paint with its family. So one day, a family decided
2 laser soldiers traveled and drove it grow into the sky. laughed like the whole."

The pumpkin stopped and smiled and felt happy for the tube. It sounded like it. Give it on big taller than it. They packed it and b
When it was done, the pumpkin had closed. It was relaxed and cozy. There were people that flying. Little Thursday workerAlright, the
Her friend thought smellyly and said: "We can try to open it! It can be alright. We can find it".

Joe was amazed. He wanted that if they could delay, they would come back back and mattered the sign

```
sentence = "A little girl went to the woods"
context = (torch.tensor(enc.encode_ordinary(sentence)).unsqueeze(dim = 0))
y = model.generate(context, 200)
print(enc.decode(y.squeeze().tolist()))
```

↩ A little girl went to the woods and he was looking at the animals and he saw a little boy with a big smile on its face. He knew she
One day, the girl called Jeff went for a walk. He saw something pretty in a nearby old structure. It was a kind of creature and he i
Ted the factory said, "I have a hero!"; he was sure it will rise.

Mary smiled and was not getting still there before. It was a powerful tw armor! And this made him even more he dream less very safe


```
from google.colab import runtime  
runtime.unassign()
```