Basic Instructions

1. Enter your Name, UID and Link to Google Drive in the provided space.
2. Submit the assignment to Gradescope.

Final Submission Deadline: May 2, 5:00pm

Late Submission Deadline: May 4, 5:00pm

Name: **Sumedh Koppula**
UID: **117386066**

Link to Google Drive :
https://drive.google.com/file/d/1ta6aqyCjJe2LsPpuibmVFRHWo5prbSlA/view?usp=sharing

In this assignment, you will learn how to use transformers to generate text. Specifically, you will implement very small GPT model. It will predict streams of characters to attempt to form nice sounding sentences.

You will complete 5 exercises, described in detail later on in this notebook.

```
import torch
print("Pytorch version：")
print(torch.__version__)
print("CUDA Version: ")
print(torch.version.cuda)
print("cuDNN version is :")
print(torch.backends.cudnn.version())
```

```
Pytorch version：
2.0.0+cu117
CUDA Version:
11.7
cuDNN version is :
8500
```

```
import os
import time
import math
import pickle
from contextlib import nullcontext

import numpy as np
import torch
from torch.distributed import init_process_group,
destroy_process_group

import os
import pickle
import requests
import numpy as np

import math
import inspect
from dataclasses import dataclass

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

DATA PREPARATION

```
# download the tiny shakespeare dataset
if not os.path.exists('data'):
  os.makedirs('data')
if not os.path.exists('data/shakespeare'):
  os.makedirs('data/shakespeare')
data_root = 'data/shakespeare'
input_file_path = os.path.join(data_root, 'input.txt')
if not os.path.exists(input_file_path):
    data_url = 
'https://raw.githubusercontent.com/learn2phoenix/CMSC472_HW6/main/
input.txt'
    with open(input_file_path, 'w') as f:
        f.write(requests.get(data_url).text)

with open(input_file_path, 'r') as f:
    data = f.read()
print(f"length of dataset in characters: {len(data):,}")

# get all the unique characters that occur in this text
chars = sorted(list(set(data)))
vocab_size = len(chars)
print("all the unique characters:", ''.join(chars))
print(f"vocab size: {vocab_size:,}")

# create a mapping from characters to integers
stoi = { ch:i for i,ch in enumerate(chars) }
itos = { i:ch for i,ch in enumerate(chars) }
def encode(s):
    return [stoi[c] for c in s] # encoder: take a string, output a 
list of integers
def decode(l):
    return ''.join([itos[i] for i in l]) # decoder: take a list of 
integers, output a string

# create the train and test splits
n = len(data)
```

```python
train_data = data[:int(n*0.9)]
val_data = data[int(n*0.9):]

# encode both to integers
train_ids = encode(train_data)
val_ids = encode(val_data)
print(f"train has {len(train_ids):,} tokens")
print(f"val has {len(val_ids):,} tokens")

# export to bin files
train_ids = np.array(train_ids, dtype=np.uint16)
val_ids = np.array(val_ids, dtype=np.uint16)
train_ids.tofile(os.path.join(data_root, 'train.bin'))
val_ids.tofile(os.path.join(data_root, 'val.bin'))

# save the meta information as well, to help us encode/decode later
meta = {
    'vocab_size': vocab_size,
    'itos': itos,
    'stoi': stoi,
}
with open(f'{data_root}/meta.pkl', 'wb') as f:
    pickle.dump(meta, f)
```

```
length of dataset in characters: 1,115,395
all the unique characters:
 !$&',-.3:;?ABCDEFGHIJKLMNOPQRSTUVWXYZabcdefghijklmnopqrstuvwxyz
vocab size: 65
train has 1,003,855 tokens
val has 111,540 tokens
```

```
input_file_path
```

```
'data/shakespeare/input.txt'
```

##Complete the TODO sections in the cell below.

```python
# @torch.jit.script # good to enable when not using torch.compile,
disable when using (our default)
def new_gelu(x):
    return 0.5 * x * (1.0 + torch.tanh(math.sqrt(2.0 / math.pi) * (x +
0.044715 * torch.pow(x, 3.0))))

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, input):
        return F.layer_norm(input, self.weight.shape, self.weight,
self.bias, 1e-5)
```

###Exercise 1 Complete the forward function for `CausalSelfAttention` class. Most of the function is already implemented, you just have to compute the query, key and values.

```
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.dropout = config.dropout
        self.flash = hasattr(torch.nn.functional,
'scaled_dot_product_attention')
        if not self.flash:
            print("WARNING: using slow attention. Flash Attention
requires PyTorch >= 2.0")
            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()

        # TODO: you should calculate key, query, values (k, q, v) from
`x` for all heads in batch.
        # Don't forget to move head forward to be the batch dim
        # HINT: using self.c_attn and splits to have q, k, v
        # YOUR CODE BEGINS HERE
        qkv = self.c_attn(x)
        q, k, v = torch.chunk(qkv, 3, dim=-1)
        q = q.view(B, T, self.n_head, self.n_embd //
self.n_head).transpose(1, 2)
        k = k.view(B, T, self.n_head, self.n_embd //
self.n_head).transpose(1, 2)
        v = v.view(B, T, self.n_head, self.n_embd //
self.n_head).transpose(1, 2)
        # YOUR CODE ENDS HERE
        if self.flash:
            y = torch.nn.functional.scaled_dot_product_attention(q, k,
v, attn_mask=None, dropout_p=self.dropout if self.training else 0,
is_causal=True)
        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
```

Some other utility blocks are defined as:

```
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.c_proj  = nn.Linear(4 * config.n_embd, config.n_embd,
bias=config.bias)
        self.dropout = nn.Dropout(config.dropout)

    def forward(self, x):
        x = self.c_fc(x)
        x = new_gelu(x)
        x = self.c_proj(x)
        x = self.dropout(x)
        return x

class Block(nn.Module):

    def __init__(self, config):
        super().__init__()
        self.ln_1 = LayerNorm(config.n_embd, bias=config.bias)
        self.attn = CausalSelfAttention(config)
        self.ln_2 = LayerNorm(config.n_embd, bias=config.bias)
        self.mlp = MLP(config)

    def forward(self, x):
        x = x + self.attn(self.ln_1(x))
        x = x + self.mlp(self.ln_2(x))
        return x

@dataclass
class GPTConfig:
    block_size: int = 1024
    vocab_size: int = 50304
    n_layer: int = 12
    n_head: int = 12
    n_embd: int = 768
    dropout: float = 0.0
    bias: bool = True
```

Exercise 2

Complete the forward function for GPT class. Most of the function is again already implemented, you need to do forward for `self.transformer` of this class.

HINT: Read the token and position embeddings, forward through each block in loop and then forward through last layer.

```
class GPT(nn.Module):

    def __init__(self, config):
        super().__init__()
        assert config.vocab_size is not None
        assert config.block_size is not None
        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, bias=config.bias),
        ))
        self.lm_head = nn.Linear(config.n_embd, config.vocab_size,
bias=False)
        self.transformer.wte.weight = self.lm_head.weight

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

        # report number of parameters
        print("number of parameters: %.2fM" %
(self.get_num_params()/1e6,))

    def get_num_params(self, non_embedding=True):
        """
        Return the number of parameters in the model.
        remember to subtract the position embeddings for non_embedding
        The token embeddings would have received the same treatement
```

```
too, but
        for their use as weights, due to parameter sharing, in the
final layer.
        """
        n_params = sum(p.numel() for p in self.parameters())
        if non_embedding:
            n_params -= self.transformer.wpe.weight.numel()
        return n_params

    def _init_weights(self, module):
        if isinstance(module, nn.Linear):
            torch.nn.init.normal_(module.weight, mean=0.0, std=0.02)
            if module.bias is not None:
                torch.nn.init.zeros_(module.bias)
        elif isinstance(module, nn.Embedding):
            torch.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, f"Cannot forward sequence
of length {t}, block size is only {self.config.block_size}"
        pos = torch.arange(0, t, dtype=torch.long,
device=device).unsqueeze(0) # shape (1, t)

        # TODO: write the forward for the GPT model and assign output
to x. HINT: Refer to definition for self.transformer
        # YOUR CODE BEGINS HERE
        x = self.transformer["wte"](idx) + self.transformer["wpe"]
(pos)
        x = self.transformer["drop"](x)

        for block in self.transformer["h"]:
            x = block(x)

        x = self.transformer["ln_f"](x)
        # YOUR CODE ENDS HERE

        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)
        else:
            logits = self.lm_head(x[:, [-1], :])
            loss = None

        return logits, loss

    def crop_block_size(self, block_size):
        assert block_size <= self.config.block_size
        self.config.block_size = block_size
        self.transformer.wpe.weight =
nn.Parameter(self.transformer.wpe.weight[:block_size])
        for block in self.transformer.h:
            if hasattr(block.attn, 'bias'):
                block.attn.bias =
block.attn.bias[:,:,:block_size,:block_size]

    @classmethod
    def from_pretrained(cls, model_type, override_args=None):
        assert model_type in {'gpt2', 'gpt2-medium', 'gpt2-large',
'gpt2-xl'}
        override_args = override_args or {}
        assert all(k == 'dropout' for k in override_args)
        from transformers import GPT2LMHeadModel
        print("loading weights from pretrained gpt: %s" % model_type)


        config_args = {
            'gpt2':         dict(n_layer=12, n_head=12, n_embd=768),
# 124M params
            'gpt2-medium':  dict(n_layer=24, n_head=16, n_embd=1024),
# 350M params
            'gpt2-large':   dict(n_layer=36, n_head=20, n_embd=1280),
# 774M params
            'gpt2-xl':      dict(n_layer=48, n_head=25, n_embd=1600),
# 1558M params
        }[model_type]
        print("forcing vocab_size=50257, block_size=1024, bias=True")
        config_args['vocab_size'] = 50257
        config_args['block_size'] = 1024
        config_args['bias'] = True
        if 'dropout' in override_args:
            print(f"overriding dropout rate to
{override_args['dropout']}")
            config_args['dropout'] = override_args['dropout']
        config = GPTConfig(**config_args)
        model = GPT(config)
        sd = model.state_dict()
        sd_keys = sd.keys()
        sd_keys = [k for k in sd_keys if not k.endswith('.attn.bias')]

        model_hf = GPT2LMHeadModel.from_pretrained(model_type)
        sd_hf = model_hf.state_dict()

        sd_keys_hf = sd_hf.keys()
        sd_keys_hf = [k for k in sd_keys_hf if not
k.endswith('.attn.masked_bias')]
        sd_keys_hf = [k for k in sd_keys_hf if not
```

```
k.endswith('.attn.bias')]
        transposed = ['attn.c_attn.weight', 'attn.c_proj.weight',
'mlp.c_fc.weight', 'mlp.c_proj.weight']
        assert len(sd_keys_hf) == len(sd_keys), f"mismatched keys:
{len(sd_keys_hf)} != {len(sd_keys)}"
        for k in sd_keys_hf:
            if any(k.endswith(w) for w in transposed):
                assert sd_hf[k].shape[::-1] == sd[k].shape
                with torch.no_grad():
                    sd[k].copy_(sd_hf[k].t())
            else:
                assert sd_hf[k].shape == sd[k].shape
                with torch.no_grad():
                    sd[k].copy_(sd_hf[k])

        return model

    def configure_optimizers(self, weight_decay, learning_rate, betas,
device_type):
        decay = set()
        no_decay = set()
        whitelist_weight_modules = (torch.nn.Linear, )
        blacklist_weight_modules = (torch.nn.LayerNorm, LayerNorm,
torch.nn.Embedding)
        for mn, m in self.named_modules():
            for pn, p in m.named_parameters():
                fpn = '%s.%s' % (mn, pn) if mn else pn
                if pn.endswith('bias'):
                    no_decay.add(fpn)
                elif pn.endswith('weight') and isinstance(m,
whitelist_weight_modules):
                    decay.add(fpn)
                elif pn.endswith('weight') and isinstance(m,
blacklist_weight_modules):
                    no_decay.add(fpn)

        decay.remove('lm_head.weight')

        param_dict = {pn: p for pn, p in self.named_parameters()}
        inter_params = decay & no_decay
        union_params = decay | no_decay
        assert len(inter_params) == 0, "parameters %s made it into
both decay/no_decay sets!" % (str(inter_params), )
        assert len(param_dict.keys() - union_params) == 0, "parameters
%s were not separated into either decay/no_decay set!" \
                                                          %

(str(param_dict.keys() - union_params), )

        optim_groups = [
            {"params": [param_dict[pn] for pn in sorted(list(decay))],
"weight_decay": weight_decay},
            {"params": [param_dict[pn] for pn in
sorted(list(no_decay))], "weight_decay": 0.0},
        ]
        use_fused = (device_type == 'cuda') and ('fused' in
inspect.signature(torch.optim.AdamW).parameters)
        print(f"using fused AdamW: {use_fused}")
        extra_args = dict(fused=True) if use_fused else dict()
        optimizer = torch.optim.AdamW(optim_groups, lr=learning_rate,
betas=betas, **extra_args)

        return optimizer

    def estimate_mfu(self, fwdbwd_per_iter, dt):
        N = self.get_num_params()
        cfg = self.config
        L, H, Q, T = cfg.n_layer, cfg.n_head, cfg.n_embd//cfg.n_head,
cfg.block_size
        flops_per_token = 6*N + 12*L*H*Q*T
        flops_per_fwdbwd = flops_per_token * T
        flops_per_iter = flops_per_fwdbwd * fwdbwd_per_iter
        flops_achieved = flops_per_iter * (1.0/dt)
        flops_promised = 312e12
        mfu = flops_achieved / flops_promised
        return mfu

    @torch.no_grad()
    def generate(self, idx, max_new_tokens, temperature=1.0,
top_k=None):
        """
        Take a conditioning sequence of indices idx (LongTensor of
shape (b,t)) and complete
        the sequence max_new_tokens times, feeding the predictions
back into the model each time.
        Most likely you'll want to make sure to be in model.eval()
mode of operation for this.
        """
        for _ in range(max_new_tokens):
            # if the sequence context is growing too long we must crop
it at block_size
            idx_cond = idx if idx.size(1) <= self.config.block_size
else idx[:, -self.config.block_size:]
            # forward the model to get the logits for the index in the
sequence
            logits, _ = self(idx_cond)
            # pluck the logits at the final step and scale by desired
temperature
            logits = logits[:, -1, :] / temperature
            # optionally crop the logits to only the top k options
            if top_k is not None:
                v, _ = torch.topk(logits, min(top_k, logits.size(-1)))
                logits[logits < v[:, [-1]]] = -float('Inf')
            # apply softmax to convert logits to (normalized)
probabilities
            probs = F.softmax(logits, dim=-1)
            # sample from the distribution
            idx_next = torch.multinomial(probs, num_samples=1)
            # append sampled index to the running sequence and
continue
            idx = torch.cat((idx, idx_next), dim=1)

        return idx
```

TRAINING

```
## TRAIN CONFIG
out_dir = 'out-shakespeare-char'
out_dir_email = 'out-enron-email'
eval_interval = 250
log_interval = 10
eval_iters = 200
eval_only = False
always_save_checkpoint = False
# data
dataset = 'shakespeare'
dataset_email = 'enron_data'
gradient_accumulation_steps = 1
batch_size = 64
block_size = 256
# model
n_layer = 6
n_head = 6
n_embd = 384
dropout = 0.2
bias =  False
# adamw optimizer
learning_rate = 1e-3
max_iters = 5000
weight_decay = 1e-1
beta1 = 0.9
beta2 = 0.99
grad_clip = 1.0
decay_lr = True
warmup_iters = 100
lr_decay_iters = 5000
min_lr = 1e-4
# system
device = 'cuda'
dtype = 'float16'
compile = False
```

```
seed_offset = 0
ddp_world_size = 1
tokens_per_iter = gradient_accumulation_steps * ddp_world_size *
batch_size * block_size
print(f"tokens per iteration will be: {tokens_per_iter:,}")
```

```
tokens per iteration will be: 16,384
```

Exercise 3

1. Complete the TODO sections in the cell below and train the model on the shakespeare data. Complete the `get_lr` function. You should implement your learning rate schedule here. Your learning rate schedule should involve linear warmup and cosine decay.

Train the model. For training, you should get loss below 2.0 in roughly 10 minutes. You should not need to run for any longer than 20 minutes (on colab GPU) to get nice results. If you're just testing things out, consider training for only a minute or so at a time, and just confirming that loss decreases. You should only need to train from start to finish 1 time-when you're ready to submit.

```
def get_lr(it):
    if it > lr_decay_iters:
        return min_lr
    # TODO: Implement the learning rate schedule and return lr for the
iteration
    # 1: include linear warmup
    # 2: implement cosine decay for after warmup (use warmup_iters
from your hyperparams)
    # YOUR CODE BEGINS HERE
    if it < warmup_iters:
        decay = it / warmup_iters
    else:
        decay = 0.5 * (1 + math.cos(math.pi * (it - warmup_iters) /
(lr_decay_iters - warmup_iters)))
    assert 0 <= decay <= 1
    coefficient = decay
    # YOUR CODE ENDS HERE
    return min_lr + coefficient * (learning_rate - min_lr)

os.makedirs(out_dir, exist_ok=True)
torch.manual_seed(1337 + seed_offset)
torch.backends.cuda.matmul.allow_tf32 = True
torch.backends.cudnn.allow_tf32 = True
device_type = 'cuda' if 'cuda' in device else 'cpu'
ptdtype = {'float32': torch.float32, 'float16': torch.float16}[dtype]
ctx = nullcontext() if device_type == 'cpu' else
torch.amp.autocast(device_type=device_type, dtype=ptdtype)

data_dir = os.path.join('data', dataset)
train_data = np.memmap(os.path.join(data_dir, 'train.bin'),
dtype=np.uint16, mode='r')
val_data = np.memmap(os.path.join(data_dir, 'val.bin'),
dtype=np.uint16, mode='r')
def get_batch(split):
    data = train_data if split == 'train' else val_data
    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.int
64)) for i in ix])
    if device_type == 'cuda':
        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

iter_num = 0
best_val_loss = 1e9

meta_path = os.path.join(data_dir, 'meta.pkl')
meta_vocab_size = None
if os.path.exists(meta_path):
    with open(meta_path, 'rb') as f:
        meta = pickle.load(f)
    meta_vocab_size = meta['vocab_size']
    print(f"found vocab_size = {meta_vocab_size} (inside
{meta_path})")

# model init
model_args = dict(n_layer=n_layer, n_head=n_head, n_embd=n_embd,
block_size=block_size,
                  bias=bias, vocab_size=None, dropout=dropout)
model_args['vocab_size'] = meta_vocab_size
gptconf = GPTConfig(**model_args)
model = GPT(gptconf)
if block_size < model.config.block_size:
    model.crop_block_size(block_size)
    model_args['block_size'] = block_size
model.to(device)

scaler = torch.cuda.amp.GradScaler(enabled=(dtype == 'float16'))

# optimizer
optimizer = model.configure_optimizers(weight_decay, learning_rate,
(beta1, beta2), device_type)
checkpoint = None
```

```python
# compile the model
if compile:
    print("compiling the model... (takes a ~minute)")
    unoptimized_model = model
    model = torch.compile(model, backend='triton') # requires PyTorch
2.0

@torch.no_grad()
def estimate_loss():
    out = {}
    model.eval()
    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

# training loop
X, Y = get_batch('train') # fetch the very first batch
t0 = time.time()
local_iter_num = 0 # number of iterations in the lifetime of this
process
raw_model = model
running_mfu = -1.0
for iter_num in range(max_iters):
    lr = get_lr(iter_num) if decay_lr else learning_rate
    for param_group in optimizer.param_groups:
        param_group['lr'] = lr

    if iter_num % eval_interval == 0:
        losses = estimate_loss()
        print(f"step {iter_num}: train loss {losses['train']:.4f}, val
loss {losses['val']:.4f}")
        if losses['val'] < best_val_loss or always_save_checkpoint:
            best_val_loss = losses['val']
            if iter_num > 0:
                checkpoint = {
                    'model': raw_model.state_dict(),
                    'optimizer': optimizer.state_dict(),
                    'model_args': model_args,
                    'iter_num': iter_num,
                    'best_val_loss': best_val_loss
                }
                print(f"saving checkpoint to {out_dir}")
                torch.save(checkpoint, os.path.join(out_dir,
'ckpt.pt'))
    if iter_num == 0 and eval_only:
        break

    for micro_step in range(gradient_accumulation_steps):
        with ctx:
            logits, loss = model(X, Y)
            loss = loss / gradient_accumulation_steps
        X, Y = get_batch('train')
        scaler.scale(loss).backward()
    if grad_clip != 0.0:
        torch.nn.utils.clip_grad_norm_(model.parameters(), grad_clip)
    scaler.step(optimizer)
    scaler.update()
    optimizer.zero_grad(set_to_none=True)

    # timing and logging
    t1 = time.time()
    dt = t1 - t0
    t0 = t1
    if iter_num % log_interval == 0:
        lossf = loss.item() * gradient_accumulation_steps
        if local_iter_num >= 5: # let the training loop settle a bit
            mfu = raw_model.estimate_mfu(batch_size *
gradient_accumulation_steps, dt)
            running_mfu = mfu if running_mfu == -1.0 else
0.9*running_mfu + 0.1*mfu
        print(f"iter {iter_num}: loss {lossf:.4f}, time
{dt*1000:.2f}ms, mfu {running_mfu*100:.2f}%")
    local_iter_num += 1
```

```
found vocab_size = 65 (inside data/shakespeare/meta.pkl)
number of parameters: 10.65M
using fused AdamW: True
step 0: train loss 4.2875, val loss 4.2826
iter 0: loss 4.2559, time 26404.08ms, mfu -100.00%
iter 10: loss 3.1613, time 181.93ms, mfu 2.05%
iter 20: loss 2.7856, time 181.93ms, mfu 2.05%
iter 30: loss 2.6532, time 183.13ms, mfu 2.05%
iter 40: loss 2.5717, time 181.93ms, mfu 2.05%
iter 50: loss 2.5366, time 182.99ms, mfu 2.05%
iter 60: loss 2.5172, time 181.94ms, mfu 2.05%
iter 70: loss 2.5206, time 182.29ms, mfu 2.05%
iter 80: loss 2.4905, time 182.11ms, mfu 2.05%
iter 90: loss 2.4793, time 181.78ms, mfu 2.05%
iter 100: loss 2.4800, time 181.94ms, mfu 2.05%
iter 110: loss 2.4873, time 182.17ms, mfu 2.05%
iter 120: loss 2.4659, time 183.49ms, mfu 2.04%
iter 130: loss 2.4458, time 182.22ms, mfu 2.04%
iter 140: loss 2.4543, time 183.19ms, mfu 2.04%
iter 150: loss 2.4504, time 183.49ms, mfu 2.04%
iter 160: loss 2.4567, time 182.00ms, mfu 2.04%
iter 170: loss 2.4535, time 182.29ms, mfu 2.04%
iter 180: loss 2.4411, time 182.10ms, mfu 2.04%
iter 190: loss 2.4347, time 182.19ms, mfu 2.04%
iter 200: loss 2.4271, time 182.21ms, mfu 2.04%
iter 210: loss 2.4088, time 183.85ms, mfu 2.04%
iter 220: loss 2.4106, time 183.82ms, mfu 2.04%
iter 230: loss 2.4307, time 182.90ms, mfu 2.04%
iter 240: loss 2.4107, time 182.24ms, mfu 2.04%
step 250: train loss 2.3809, val loss 2.4105
saving checkpoint to out-shakespeare-char
iter 250: loss 2.4024, time 26510.42ms, mfu 1.84%
iter 260: loss 2.3928, time 182.18ms, mfu 1.86%
iter 270: loss 2.4040, time 182.10ms, mfu 1.88%
iter 280: loss 2.4118, time 182.69ms, mfu 1.89%
iter 290: loss 2.3968, time 183.49ms, mfu 1.91%
iter 300: loss 2.3730, time 181.90ms, mfu 1.92%
iter 310: loss 2.3788, time 181.87ms, mfu 1.93%
iter 320: loss 2.3640, time 182.26ms, mfu 1.95%
iter 330: loss 2.3529, time 183.32ms, mfu 1.95%
iter 340: loss 2.3663, time 182.34ms, mfu 1.96%
iter 350: loss 2.3519, time 183.23ms, mfu 1.97%
iter 360: loss 2.3305, time 182.77ms, mfu 1.98%
iter 370: loss 2.3331, time 182.44ms, mfu 1.98%
iter 380: loss 2.3078, time 181.77ms, mfu 1.99%
iter 390: loss 2.3258, time 183.06ms, mfu 1.99%
iter 400: loss 2.3274, time 181.73ms, mfu 2.00%
iter 410: loss 2.2860, time 182.30ms, mfu 2.00%
iter 420: loss 2.3032, time 183.02ms, mfu 2.01%
iter 430: loss 2.2584, time 183.55ms, mfu 2.01%
iter 440: loss 2.2375, time 182.25ms, mfu 2.01%
iter 450: loss 2.2449, time 182.80ms, mfu 2.02%
iter 460: loss 2.1925, time 182.86ms, mfu 2.02%
iter 470: loss 2.2173, time 182.41ms, mfu 2.02%
iter 480: loss 2.2067, time 185.87ms, mfu 2.02%
iter 490: loss 2.1668, time 181.50ms, mfu 2.02%
step 500: train loss 2.0697, val loss 2.1519
saving checkpoint to out-shakespeare-char
iter 500: loss 2.1579, time 26503.83ms, mfu 1.82%
iter 510: loss 2.1412, time 183.10ms, mfu 1.84%
iter 520: loss 2.1273, time 182.06ms, mfu 1.86%
iter 530: loss 2.1286, time 183.06ms, mfu 1.88%
iter 540: loss 2.1408, time 182.17ms, mfu 1.90%
iter 550: loss 2.0856, time 183.21ms, mfu 1.91%
iter 560: loss 2.1191, time 182.26ms, mfu 1.92%
iter 570: loss 2.1009, time 181.74ms, mfu 1.94%
iter 580: loss 2.0688, time 181.87ms, mfu 1.95%
iter 590: loss 2.0423, time 182.14ms, mfu 1.96%
iter 600: loss 2.0382, time 181.81ms, mfu 1.97%
iter 610: loss 2.0241, time 182.53ms, mfu 1.97%
iter 620: loss 2.0157, time 182.39ms, mfu 1.98%
iter 630: loss 2.0091, time 182.08ms, mfu 1.99%
iter 640: loss 1.9844, time 182.19ms, mfu 1.99%
iter 650: loss 1.9958, time 183.23ms, mfu 2.00%
iter 660: loss 1.9854, time 183.49ms, mfu 2.00%
iter 670: loss 1.9328, time 182.19ms, mfu 2.01%
iter 680: loss 1.9534, time 182.34ms, mfu 2.01%
iter 690: loss 1.9321, time 181.80ms, mfu 2.01%
iter 700: loss 1.9100, time 182.90ms, mfu 2.02%
iter 710: loss 1.9026, time 182.24ms, mfu 2.02%
iter 720: loss 1.8703, time 182.52ms, mfu 2.02%
iter 730: loss 1.9007, time 182.00ms, mfu 2.02%
iter 740: loss 1.8711, time 182.15ms, mfu 2.03%
step 750: train loss 1.7652, val loss 1.9186
saving checkpoint to out-shakespeare-char
iter 750: loss 1.8595, time 26604.04ms, mfu 1.82%
iter 760: loss 1.8759, time 183.85ms, mfu 1.84%
iter 770: loss 1.8488, time 184.75ms, mfu 1.86%
iter 780: loss 1.8343, time 182.77ms, mfu 1.88%
iter 790: loss 1.8415, time 183.74ms, mfu 1.89%
iter 800: loss 1.8403, time 183.65ms, mfu 1.91%
iter 810: loss 1.8143, time 182.61ms, mfu 1.92%
iter 820: loss 1.8196, time 183.75ms, mfu 1.93%
iter 830: loss 1.7727, time 183.76ms, mfu 1.94%
iter 840: loss 1.7906, time 183.23ms, mfu 1.95%
iter 850: loss 1.7670, time 183.90ms, mfu 1.96%
iter 860: loss 1.7428, time 182.49ms, mfu 1.97%
iter 870: loss 1.7772, time 183.70ms, mfu 1.97%
iter 880: loss 1.7481, time 182.53ms, mfu 1.98%
iter 890: loss 1.7215, time 182.53ms, mfu 1.99%
iter 900: loss 1.7372, time 184.97ms, mfu 1.99%
iter 910: loss 1.7552, time 184.62ms, mfu 1.99%
iter 920: loss 1.7382, time 182.51ms, mfu 2.00%
iter 930: loss 1.7348, time 183.16ms, mfu 2.00%
iter 940: loss 1.7136, time 183.62ms, mfu 2.00%
iter 950: loss 1.7114, time 182.96ms, mfu 2.01%
iter 960: loss 1.6945, time 184.14ms, mfu 2.01%
iter 970: loss 1.6825, time 183.97ms, mfu 2.01%
iter 980: loss 1.6777, time 182.36ms, mfu 2.01%
iter 990: loss 1.6683, time 182.54ms, mfu 2.02%
step 1000: train loss 1.5831, val loss 1.7665
saving checkpoint to out-shakespeare-char
iter 1000: loss 1.6923, time 26595.71ms, mfu 1.82%
iter 1010: loss 1.6807, time 183.42ms, mfu 1.84%
iter 1020: loss 1.6278, time 182.65ms, mfu 1.86%
iter 1030: loss 1.6606, time 182.83ms, mfu 1.88%
iter 1040: loss 1.6386, time 182.38ms, mfu 1.89%
iter 1050: loss 1.6706, time 183.29ms, mfu 1.91%
iter 1060: loss 1.6305, time 183.02ms, mfu 1.92%
iter 1070: loss 1.6478, time 183.92ms, mfu 1.93%
iter 1080: loss 1.6709, time 182.63ms, mfu 1.94%
iter 1090: loss 1.6688, time 187.71ms, mfu 1.95%
iter 1100: loss 1.5898, time 183.57ms, mfu 1.95%
iter 1110: loss 1.6179, time 182.70ms, mfu 1.96%
iter 1120: loss 1.5894, time 182.63ms, mfu 1.97%
iter 1130: loss 1.6350, time 183.24ms, mfu 1.98%
iter 1140: loss 1.5946, time 183.53ms, mfu 1.98%
iter 1150: loss 1.6044, time 183.21ms, mfu 1.99%
iter 1160: loss 1.6063, time 183.18ms, mfu 1.99%
iter 1170: loss 1.5884, time 184.36ms, mfu 1.99%
iter 1180: loss 1.6010, time 184.60ms, mfu 2.00%
iter 1190: loss 1.5642, time 183.53ms, mfu 2.00%
iter 1200: loss 1.5921, time 183.62ms, mfu 2.00%
iter 1210: loss 1.5891, time 182.79ms, mfu 2.01%
iter 1220: loss 1.5868, time 182.86ms, mfu 2.01%
iter 1230: loss 1.5773, time 182.68ms, mfu 2.01%
iter 1240: loss 1.5728, time 183.87ms, mfu 2.01%
step 1250: train loss 1.4759, val loss 1.6771
saving checkpoint to out-shakespeare-char
iter 1250: loss 1.5635, time 26544.92ms, mfu 1.81%
iter 1260: loss 1.5790, time 183.10ms, mfu 1.84%
iter 1270: loss 1.5611, time 183.99ms, mfu 1.86%
iter 1280: loss 1.5537, time 182.64ms, mfu 1.87%
iter 1290: loss 1.5667, time 182.82ms, mfu 1.89%
iter 1300: loss 1.5500, time 183.18ms, mfu 1.90%
iter 1310: loss 1.5643, time 182.92ms, mfu 1.92%
iter 1320: loss 1.5938, time 182.94ms, mfu 1.93%
iter 1330: loss 1.5327, time 182.89ms, mfu 1.94%
iter 1340: loss 1.5875, time 183.56ms, mfu 1.95%
iter 1350: loss 1.5447, time 183.46ms, mfu 1.96%
iter 1360: loss 1.5128, time 182.52ms, mfu 1.97%
iter 1370: loss 1.5373, time 182.51ms, mfu 1.97%
iter 1380: loss 1.5033, time 183.76ms, mfu 1.98%
iter 1390: loss 1.5306, time 184.11ms, mfu 1.98%
iter 1400: loss 1.5038, time 182.91ms, mfu 1.99%
iter 1410: loss 1.5042, time 182.51ms, mfu 1.99%
iter 1420: loss 1.5054, time 183.59ms, mfu 2.00%
iter 1430: loss 1.5295, time 184.11ms, mfu 2.00%
iter 1440: loss 1.4742, time 182.85ms, mfu 2.00%
iter 1450: loss 1.5133, time 182.71ms, mfu 2.01%
iter 1460: loss 1.4958, time 183.32ms, mfu 2.01%
iter 1470: loss 1.5022, time 182.74ms, mfu 2.01%
iter 1480: loss 1.4835, time 182.60ms, mfu 2.02%
iter 1490: loss 1.5221, time 182.88ms, mfu 2.02%
step 1500: train loss 1.4006, val loss 1.6183
saving checkpoint to out-shakespeare-char
iter 1500: loss 1.5172, time 26527.16ms, mfu 1.82%
iter 1510: loss 1.4883, time 184.32ms, mfu 1.84%
iter 1520: loss 1.4731, time 182.91ms, mfu 1.86%
iter 1530: loss 1.4916, time 183.59ms, mfu 1.88%
iter 1540: loss 1.5160, time 183.13ms, mfu 1.89%
iter 1550: loss 1.4665, time 182.44ms, mfu 1.91%
iter 1560: loss 1.4882, time 184.41ms, mfu 1.92%
iter 1570: loss 1.4592, time 183.69ms, mfu 1.93%
iter 1580: loss 1.4700, time 182.83ms, mfu 1.94%
iter 1590: loss 1.4535, time 183.49ms, mfu 1.95%
iter 1600: loss 1.4642, time 182.27ms, mfu 1.96%
iter 1610: loss 1.4456, time 182.57ms, mfu 1.97%
iter 1620: loss 1.4393, time 182.46ms, mfu 1.97%
iter 1630: loss 1.4358, time 183.71ms, mfu 1.98%
iter 1640: loss 1.4717, time 182.39ms, mfu 1.99%
iter 1650: loss 1.4509, time 182.20ms, mfu 1.99%
iter 1660: loss 1.4154, time 183.04ms, mfu 2.00%
iter 1670: loss 1.4428, time 183.46ms, mfu 2.00%
iter 1680: loss 1.4171, time 182.41ms, mfu 2.00%
iter 1690: loss 1.4408, time 182.71ms, mfu 2.01%
iter 1700: loss 1.4757, time 182.90ms, mfu 2.01%
iter 1710: loss 1.4159, time 182.56ms, mfu 2.01%
iter 1720: loss 1.4448, time 182.79ms, mfu 2.02%
iter 1730: loss 1.4054, time 184.01ms, mfu 2.02%
iter 1740: loss 1.3997, time 183.48ms, mfu 2.02%
step 1750: train loss 1.3459, val loss 1.5727
saving checkpoint to out-shakespeare-char
iter 1750: loss 1.4217, time 26574.05ms, mfu 1.82%
iter 1760: loss 1.4438, time 182.55ms, mfu 1.84%
iter 1770: loss 1.4349, time 182.96ms, mfu 1.86%
iter 1780: loss 1.4427, time 183.19ms, mfu 1.88%
iter 1790: loss 1.3843, time 182.88ms, mfu 1.89%
iter 1800: loss 1.4529, time 183.70ms, mfu 1.91%
iter 1810: loss 1.4097, time 182.56ms, mfu 1.92%
iter 1820: loss 1.4282, time 182.26ms, mfu 1.93%
iter 1830: loss 1.4330, time 182.32ms, mfu 1.94%
iter 1840: loss 1.4215, time 182.52ms, mfu 1.95%
iter 1850: loss 1.4301, time 182.33ms, mfu 1.96%
iter 1860: loss 1.4274, time 183.08ms, mfu 1.97%
iter 1870: loss 1.4018, time 182.38ms, mfu 1.98%
iter 1880: loss 1.3919, time 187.94ms, mfu 1.98%
iter 1890: loss 1.4102, time 182.56ms, mfu 1.98%
iter 1900: loss 1.3930, time 182.51ms, mfu 1.99%
iter 1910: loss 1.3836, time 183.85ms, mfu 1.99%
iter 1920: loss 1.3911, time 182.77ms, mfu 2.00%
iter 1930: loss 1.4125, time 182.36ms, mfu 2.00%
iter 1940: loss 1.3908, time 182.77ms, mfu 2.01%
iter 1950: loss 1.4154, time 182.72ms, mfu 2.01%
iter 1960: loss 1.3923, time 182.57ms, mfu 2.01%
iter 1970: loss 1.3901, time 182.58ms, mfu 2.02%
iter 1980: loss 1.4014, time 183.80ms, mfu 2.02%
iter 1990: loss 1.3923, time 182.80ms, mfu 2.02%
step 2000: train loss 1.3063, val loss 1.5454
```

```
saving checkpoint to out-shakespeare-char
iter 2000: loss 1.4013, time 26605.51ms, mfu 1.82%
iter 2010: loss 1.3753, time 182.45ms, mfu 1.84%
iter 2020: loss 1.3787, time 182.64ms, mfu 1.86%
iter 2030: loss 1.3791, time 182.90ms, mfu 1.88%
iter 2040: loss 1.3431, time 182.39ms, mfu 1.89%
iter 2050: loss 1.3341, time 183.19ms, mfu 1.91%
iter 2060: loss 1.3766, time 183.53ms, mfu 1.92%
iter 2070: loss 1.3378, time 182.64ms, mfu 1.93%
iter 2080: loss 1.3540, time 183.92ms, mfu 1.94%
iter 2090: loss 1.3667, time 182.80ms, mfu 1.95%
iter 2100: loss 1.3760, time 183.12ms, mfu 1.96%
iter 2110: loss 1.3120, time 183.19ms, mfu 1.97%
iter 2120: loss 1.3700, time 182.77ms, mfu 1.97%
iter 2130: loss 1.3855, time 182.89ms, mfu 1.98%
iter 2140: loss 1.3301, time 187.54ms, mfu 1.98%
iter 2150: loss 1.3259, time 182.57ms, mfu 1.99%
iter 2160: loss 1.3305, time 183.28ms, mfu 1.99%
iter 2170: loss 1.3447, time 182.76ms, mfu 2.00%
iter 2180: loss 1.3297, time 182.74ms, mfu 2.00%
iter 2190: loss 1.3523, time 182.64ms, mfu 2.00%
iter 2200: loss 1.3461, time 190.33ms, mfu 2.00%
iter 2210: loss 1.3585, time 184.10ms, mfu 2.00%
iter 2220: loss 1.3075, time 183.16ms, mfu 2.01%
iter 2230: loss 1.3476, time 182.75ms, mfu 2.01%
iter 2240: loss 1.3678, time 182.83ms, mfu 2.01%
step 2250: train loss 1.2592, val loss 1.5160
saving checkpoint to out-shakespeare-char
iter 2250: loss 1.3157, time 26603.06ms, mfu 1.81%
iter 2260: loss 1.3619, time 184.02ms, mfu 1.83%
iter 2270: loss 1.3344, time 182.92ms, mfu 1.85%
iter 2280: loss 1.3554, time 182.60ms, mfu 1.87%
iter 2290: loss 1.3689, time 182.44ms, mfu 1.89%
iter 2300: loss 1.3391, time 183.28ms, mfu 1.90%
iter 2310: loss 1.3172, time 182.52ms, mfu 1.92%
iter 2320: loss 1.3537, time 183.22ms, mfu 1.93%
iter 2330: loss 1.2910, time 182.45ms, mfu 1.94%
iter 2340: loss 1.3519, time 182.45ms, mfu 1.95%
iter 2350: loss 1.3175, time 182.93ms, mfu 1.96%
iter 2360: loss 1.3415, time 182.59ms, mfu 1.97%
iter 2370: loss 1.3341, time 183.49ms, mfu 1.97%
iter 2380: loss 1.3266, time 183.74ms, mfu 1.98%
iter 2390: loss 1.3455, time 184.06ms, mfu 1.98%
iter 2400: loss 1.3336, time 183.23ms, mfu 1.99%
iter 2410: loss 1.3410, time 182.57ms, mfu 1.99%
iter 2420: loss 1.3229, time 182.93ms, mfu 2.00%
iter 2430: loss 1.3285, time 182.82ms, mfu 2.00%
iter 2440: loss 1.3407, time 183.43ms, mfu 2.01%
iter 2450: loss 1.3113, time 183.04ms, mfu 2.01%
iter 2460: loss 1.3052, time 183.22ms, mfu 2.01%
iter 2470: loss 1.3342, time 182.96ms, mfu 2.01%
iter 2480: loss 1.3446, time 182.31ms, mfu 2.02%
iter 2490: loss 1.3474, time 183.75ms, mfu 2.02%
step 2500: train loss 1.2367, val loss 1.5044
saving checkpoint to out-shakespeare-char
iter 2500: loss 1.3279, time 26655.16ms, mfu 1.82%
iter 2510: loss 1.3387, time 182.45ms, mfu 1.84%
iter 2520: loss 1.3131, time 182.45ms, mfu 1.86%
iter 2530: loss 1.3159, time 182.28ms, mfu 1.88%
iter 2540: loss 1.3058, time 182.31ms, mfu 1.89%
iter 2550: loss 1.3051, time 182.04ms, mfu 1.91%
iter 2560: loss 1.3249, time 183.75ms, mfu 1.92%
iter 2570: loss 1.3370, time 183.81ms, mfu 1.93%
iter 2580: loss 1.3093, time 184.15ms, mfu 1.94%
iter 2590: loss 1.3433, time 182.85ms, mfu 1.95%
iter 2600: loss 1.3235, time 183.23ms, mfu 1.96%
iter 2610: loss 1.3094, time 183.16ms, mfu 1.97%
iter 2620: loss 1.3141, time 183.75ms, mfu 1.97%
iter 2630: loss 1.2928, time 183.52ms, mfu 1.98%
iter 2640: loss 1.3196, time 182.77ms, mfu 1.98%
iter 2650: loss 1.3035, time 182.69ms, mfu 1.99%
iter 2660: loss 1.3195, time 182.52ms, mfu 2.00%
iter 2670: loss 1.3246, time 182.47ms, mfu 2.00%
iter 2680: loss 1.3133, time 182.36ms, mfu 2.00%
iter 2690: loss 1.3045, time 185.68ms, mfu 2.00%
iter 2700: loss 1.2890, time 182.75ms, mfu 2.01%
iter 2710: loss 1.3046, time 182.56ms, mfu 2.01%
iter 2720: loss 1.3228, time 184.63ms, mfu 2.01%
iter 2730: loss 1.2955, time 183.86ms, mfu 2.01%
iter 2740: loss 1.3063, time 184.35ms, mfu 2.01%
step 2750: train loss 1.2176, val loss 1.4927
saving checkpoint to out-shakespeare-char
iter 2750: loss 1.2962, time 26751.71ms, mfu 1.81%
iter 2760: loss 1.3217, time 183.76ms, mfu 1.84%
iter 2770: loss 1.3020, time 184.05ms, mfu 1.85%
iter 2780: loss 1.2829, time 182.48ms, mfu 1.87%
iter 2790: loss 1.2845, time 182.97ms, mfu 1.89%
iter 2800: loss 1.2799, time 184.21ms, mfu 1.90%
iter 2810: loss 1.2844, time 183.92ms, mfu 1.92%
iter 2820: loss 1.2642, time 182.62ms, mfu 1.93%
iter 2830: loss 1.3320, time 182.71ms, mfu 1.94%
iter 2840: loss 1.2989, time 182.58ms, mfu 1.95%
iter 2850: loss 1.2840, time 183.18ms, mfu 1.96%
iter 2860: loss 1.2956, time 183.42ms, mfu 1.97%
iter 2870: loss 1.2888, time 182.89ms, mfu 1.97%
iter 2880: loss 1.3216, time 183.08ms, mfu 1.98%
iter 2890: loss 1.3202, time 183.12ms, mfu 1.98%
iter 2900: loss 1.3001, time 182.65ms, mfu 1.99%
iter 2910: loss 1.3162, time 183.58ms, mfu 1.99%
iter 2920: loss 1.2885, time 183.56ms, mfu 2.00%
iter 2930: loss 1.2881, time 183.09ms, mfu 2.00%
iter 2940: loss 1.2735, time 182.59ms, mfu 2.01%
iter 2950: loss 1.2884, time 184.29ms, mfu 2.01%
iter 2960: loss 1.3013, time 184.80ms, mfu 2.01%
iter 2970: loss 1.2714, time 183.79ms, mfu 2.01%
iter 2980: loss 1.2896, time 183.65ms, mfu 2.01%
iter 2990: loss 1.2809, time 184.24ms, mfu 2.01%
step 3000: train loss 1.2000, val loss 1.4902
saving checkpoint to out-shakespeare-char
iter 3000: loss 1.2817, time 26633.45ms, mfu 1.81%
iter 3010: loss 1.2659, time 182.69ms, mfu 1.84%
iter 3020: loss 1.2915, time 182.92ms, mfu 1.86%
iter 3030: loss 1.2793, time 183.92ms, mfu 1.87%
iter 3040: loss 1.3065, time 182.57ms, mfu 1.89%
iter 3050: loss 1.2691, time 183.76ms, mfu 1.90%
iter 3060: loss 1.2565, time 184.28ms, mfu 1.92%
iter 3070: loss 1.3215, time 184.27ms, mfu 1.93%
iter 3080: loss 1.2468, time 183.66ms, mfu 1.94%
iter 3090: loss 1.2526, time 182.94ms, mfu 1.95%
iter 3100: loss 1.2605, time 182.80ms, mfu 1.96%
iter 3110: loss 1.2872, time 183.17ms, mfu 1.96%
iter 3120: loss 1.2707, time 184.08ms, mfu 1.97%
iter 3130: loss 1.2649, time 182.85ms, mfu 1.98%
iter 3140: loss 1.2593, time 183.05ms, mfu 1.98%
iter 3150: loss 1.2713, time 183.12ms, mfu 1.99%
iter 3160: loss 1.2712, time 183.71ms, mfu 1.99%
iter 3170: loss 1.2852, time 183.26ms, mfu 2.00%
iter 3180: loss 1.2862, time 182.66ms, mfu 2.00%
iter 3190: loss 1.2956, time 182.62ms, mfu 2.00%
iter 3200: loss 1.2645, time 182.39ms, mfu 2.01%
iter 3210: loss 1.2501, time 183.73ms, mfu 2.01%
iter 3220: loss 1.2835, time 183.25ms, mfu 2.01%
iter 3230: loss 1.2733, time 183.19ms, mfu 2.01%
iter 3240: loss 1.2476, time 182.68ms, mfu 2.02%
step 3250: train loss 1.1830, val loss 1.4837
saving checkpoint to out-shakespeare-char
iter 3250: loss 1.2428, time 26707.70ms, mfu 1.82%
iter 3260: loss 1.2390, time 182.84ms, mfu 1.84%
iter 3270: loss 1.2562, time 182.90ms, mfu 1.86%
iter 3280: loss 1.2717, time 183.82ms, mfu 1.88%
iter 3290: loss 1.2819, time 182.94ms, mfu 1.89%
iter 3300: loss 1.2822, time 183.86ms, mfu 1.91%
iter 3310: loss 1.2679, time 183.17ms, mfu 1.92%
iter 3320: loss 1.2660, time 185.41ms, mfu 1.93%
iter 3330: loss 1.2589, time 194.01ms, mfu 1.93%
iter 3340: loss 1.2372, time 190.28ms, mfu 1.93%
iter 3350: loss 1.2815, time 182.57ms, mfu 1.94%
iter 3360: loss 1.2749, time 184.16ms, mfu 1.95%
iter 3370: loss 1.2804, time 183.54ms, mfu 1.96%
iter 3380: loss 1.3006, time 182.87ms, mfu 1.97%
iter 3390: loss 1.2668, time 182.75ms, mfu 1.97%
iter 3400: loss 1.2793, time 183.81ms, mfu 1.98%
iter 3410: loss 1.2537, time 182.73ms, mfu 1.98%
iter 3420: loss 1.2299, time 182.85ms, mfu 1.99%
iter 3430: loss 1.2558, time 182.45ms, mfu 1.99%
iter 3440: loss 1.2570, time 184.21ms, mfu 2.00%
iter 3450: loss 1.2595, time 183.43ms, mfu 2.00%
iter 3460: loss 1.2490, time 182.82ms, mfu 2.00%
iter 3470: loss 1.2439, time 183.12ms, mfu 2.01%
iter 3480: loss 1.2416, time 182.76ms, mfu 2.01%
iter 3490: loss 1.2351, time 183.65ms, mfu 2.01%
step 3500: train loss 1.1703, val loss 1.4720
saving checkpoint to out-shakespeare-char
iter 3500: loss 1.2596, time 26655.39ms, mfu 1.81%
iter 3510: loss 1.2267, time 183.00ms, mfu 1.84%
iter 3520: loss 1.2596, time 188.33ms, mfu 1.85%
iter 3530: loss 1.2866, time 183.11ms, mfu 1.87%
iter 3540: loss 1.2270, time 182.78ms, mfu 1.89%
iter 3550: loss 1.2612, time 182.53ms, mfu 1.90%
iter 3560: loss 1.2440, time 183.74ms, mfu 1.91%
iter 3570: loss 1.2702, time 182.87ms, mfu 1.93%
iter 3580: loss 1.2183, time 184.10ms, mfu 1.94%
iter 3590: loss 1.2765, time 182.71ms, mfu 1.95%
iter 3600: loss 1.2514, time 182.89ms, mfu 1.96%
iter 3610: loss 1.2782, time 183.95ms, mfu 1.96%
iter 3620: loss 1.2352, time 182.95ms, mfu 1.97%
iter 3630: loss 1.2213, time 183.17ms, mfu 1.98%
iter 3640: loss 1.2326, time 183.42ms, mfu 1.98%
iter 3650: loss 1.2604, time 182.89ms, mfu 1.99%
iter 3660: loss 1.2563, time 183.77ms, mfu 1.99%
iter 3670: loss 1.2331, time 183.15ms, mfu 2.00%
iter 3680: loss 1.2405, time 183.24ms, mfu 2.00%
iter 3690: loss 1.2418, time 182.68ms, mfu 2.00%
iter 3700: loss 1.2462, time 183.02ms, mfu 2.01%
iter 3710: loss 1.2803, time 182.58ms, mfu 2.01%
iter 3720: loss 1.2652, time 183.82ms, mfu 2.01%
iter 3730: loss 1.2329, time 182.39ms, mfu 2.01%
iter 3740: loss 1.2545, time 183.33ms, mfu 2.02%
step 3750: train loss 1.1573, val loss 1.4727
iter 3750: loss 1.2504, time 26346.71ms, mfu 1.82%
iter 3760: loss 1.2457, time 183.14ms, mfu 1.84%
iter 3770: loss 1.2412, time 183.65ms, mfu 1.86%
iter 3780: loss 1.2459, time 183.49ms, mfu 1.87%
iter 3790: loss 1.2433, time 182.96ms, mfu 1.89%
iter 3800: loss 1.2513, time 183.21ms, mfu 1.91%
iter 3810: loss 1.2678, time 183.51ms, mfu 1.92%
iter 3820: loss 1.2385, time 182.98ms, mfu 1.93%
iter 3830: loss 1.2608, time 184.43ms, mfu 1.94%
iter 3840: loss 1.2330, time 183.65ms, mfu 1.95%
iter 3850: loss 1.2241, time 183.70ms, mfu 1.96%
iter 3860: loss 1.2206, time 182.53ms, mfu 1.96%
iter 3870: loss 1.2194, time 183.07ms, mfu 1.97%
iter 3880: loss 1.2232, time 183.26ms, mfu 1.98%
iter 3890: loss 1.2540, time 183.13ms, mfu 1.98%
iter 3900: loss 1.2263, time 183.05ms, mfu 1.99%
iter 3910: loss 1.2473, time 182.89ms, mfu 1.99%
iter 3920: loss 1.2318, time 183.38ms, mfu 2.00%
iter 3930: loss 1.2061, time 184.19ms, mfu 2.00%
iter 3940: loss 1.2374, time 183.78ms, mfu 2.00%
iter 3950: loss 1.2561, time 183.90ms, mfu 2.01%
iter 3960: loss 1.2381, time 183.87ms, mfu 2.01%
iter 3970: loss 1.2470, time 182.96ms, mfu 2.01%
iter 3980: loss 1.2409, time 182.91ms, mfu 2.01%
iter 3990: loss 1.2321, time 182.68ms, mfu 2.02%
step 4000: train loss 1.1444, val loss 1.4727
iter 4000: loss 1.2102, time 26360.70ms, mfu 1.82%
iter 4010: loss 1.2347, time 182.66ms, mfu 1.84%
iter 4020: loss 1.2348, time 182.99ms, mfu 1.86%
iter 4030: loss 1.2245, time 182.65ms, mfu 1.88%
iter 4040: loss 1.2289, time 183.19ms, mfu 1.89%
iter 4050: loss 1.2320, time 182.97ms, mfu 1.91%
iter 4060: loss 1.2164, time 190.97ms, mfu 1.91%
iter 4070: loss 1.2221, time 183.42ms, mfu 1.92%
iter 4080: loss 1.2149, time 182.80ms, mfu 1.93%
iter 4090: loss 1.2271, time 183.48ms, mfu 1.94%
iter 4100: loss 1.2143, time 182.68ms, mfu 1.95%
iter 4110: loss 1.2616, time 182.76ms, mfu 1.96%
iter 4120: loss 1.2404, time 182.60ms, mfu 1.97%
iter 4130: loss 1.2391, time 182.62ms, mfu 1.98%
iter 4140: loss 1.2498, time 183.28ms, mfu 1.98%
iter 4150: loss 1.2313, time 182.98ms, mfu 1.99%
iter 4160: loss 1.2172, time 183.49ms, mfu 1.99%
iter 4170: loss 1.2149, time 182.72ms, mfu 2.00%
iter 4180: loss 1.2149, time 182.36ms, mfu 2.00%
iter 4190: loss 1.2592, time 183.36ms, mfu 2.00%
iter 4200: loss 1.1885, time 182.75ms, mfu 2.01%
iter 4210: loss 1.1969, time 183.78ms, mfu 2.01%
iter 4220: loss 1.2294, time 182.74ms, mfu 2.01%
iter 4230: loss 1.2331, time 183.08ms, mfu 2.02%
iter 4240: loss 1.2341, time 188.87ms, mfu 2.01%
step 4250: train loss 1.1356, val loss 1.4676
saving checkpoint to out-shakespeare-char
iter 4250: loss 1.2380, time 26682.50ms, mfu 1.81%
iter 4260: loss 1.1785, time 183.32ms, mfu 1.83%
iter 4270: loss 1.2439, time 183.57ms, mfu 1.85%
iter 4280: loss 1.2105, time 182.76ms, mfu 1.87%
iter 4290: loss 1.2316, time 183.03ms, mfu 1.89%
iter 4300: loss 1.2228, time 183.04ms, mfu 1.90%
iter 4310: loss 1.2022, time 183.11ms, mfu 1.92%
iter 4320: loss 1.2127, time 183.35ms, mfu 1.93%
iter 4330: loss 1.2329, time 183.53ms, mfu 1.94%
iter 4340: loss 1.2377, time 183.26ms, mfu 1.95%
iter 4350: loss 1.2189, time 183.01ms, mfu 1.96%
iter 4360: loss 1.2328, time 182.47ms, mfu 1.96%
iter 4370: loss 1.2190, time 184.15ms, mfu 1.97%
iter 4380: loss 1.1968, time 182.96ms, mfu 1.98%
iter 4390: loss 1.2169, time 183.75ms, mfu 1.98%
iter 4400: loss 1.2132, time 184.16ms, mfu 1.99%
iter 4410: loss 1.2108, time 183.97ms, mfu 1.99%
iter 4420: loss 1.2499, time 182.82ms, mfu 2.00%
iter 4430: loss 1.1946, time 182.83ms, mfu 2.00%
iter 4440: loss 1.1919, time 184.38ms, mfu 2.00%
iter 4450: loss 1.2329, time 182.90ms, mfu 2.01%
iter 4460: loss 1.2061, time 183.34ms, mfu 2.01%
iter 4470: loss 1.2277, time 182.30ms, mfu 2.01%
iter 4480: loss 1.2175, time 182.95ms, mfu 2.01%
iter 4490: loss 1.2148, time 183.11ms, mfu 2.02%
step 4500: train loss 1.1318, val loss 1.4733
iter 4500: loss 1.2143, time 26340.00ms, mfu 1.82%
iter 4510: loss 1.2153, time 183.50ms, mfu 1.84%
iter 4520: loss 1.1952, time 183.17ms, mfu 1.86%
iter 4530: loss 1.2177, time 183.20ms, mfu 1.87%
iter 4540: loss 1.2140, time 183.98ms, mfu 1.89%
iter 4550: loss 1.2339, time 182.68ms, mfu 1.90%
iter 4560: loss 1.2498, time 182.63ms, mfu 1.92%
iter 4570: loss 1.2251, time 183.46ms, mfu 1.93%
iter 4580: loss 1.2350, time 182.72ms, mfu 1.94%
iter 4590: loss 1.2394, time 184.04ms, mfu 1.95%
iter 4600: loss 1.2073, time 183.47ms, mfu 1.96%
iter 4610: loss 1.2318, time 182.46ms, mfu 1.97%
iter 4620: loss 1.2145, time 184.51ms, mfu 1.97%
iter 4630: loss 1.2240, time 183.83ms, mfu 1.98%
iter 4640: loss 1.2328, time 183.17ms, mfu 1.98%
iter 4650: loss 1.2378, time 183.47ms, mfu 1.99%
iter 4660: loss 1.2055, time 182.77ms, mfu 1.99%
iter 4670: loss 1.1968, time 182.45ms, mfu 2.00%
iter 4680: loss 1.2436, time 184.30ms, mfu 2.00%
iter 4690: loss 1.2327, time 183.54ms, mfu 2.00%
iter 4700: loss 1.1852, time 182.62ms, mfu 2.01%
iter 4710: loss 1.2177, time 182.63ms, mfu 2.01%
iter 4720: loss 1.2151, time 182.81ms, mfu 2.01%
iter 4730: loss 1.2130, time 183.16ms, mfu 2.02%
iter 4740: loss 1.2145, time 182.86ms, mfu 2.02%
step 4750: train loss 1.1290, val loss 1.4695
iter 4750: loss 1.1761, time 26417.54ms, mfu 1.82%
iter 4760: loss 1.2110, time 182.91ms, mfu 1.84%
iter 4770: loss 1.2008, time 183.22ms, mfu 1.86%
iter 4780: loss 1.2038, time 183.05ms, mfu 1.88%
iter 4790: loss 1.2327, time 183.40ms, mfu 1.89%
iter 4800: loss 1.2085, time 183.08ms, mfu 1.91%
iter 4810: loss 1.2417, time 182.45ms, mfu 1.92%
iter 4820: loss 1.2081, time 184.11ms, mfu 1.93%
iter 4830: loss 1.2297, time 184.19ms, mfu 1.94%
iter 4840: loss 1.2085, time 182.66ms, mfu 1.95%
iter 4850: loss 1.2202, time 184.04ms, mfu 1.96%
iter 4860: loss 1.2138, time 182.78ms, mfu 1.97%
iter 4870: loss 1.2373, time 184.38ms, mfu 1.97%
iter 4880: loss 1.2332, time 182.71ms, mfu 1.98%
iter 4890: loss 1.2057, time 182.58ms, mfu 1.98%
iter 4900: loss 1.2059, time 183.25ms, mfu 1.99%
iter 4910: loss 1.2244, time 182.63ms, mfu 1.99%
iter 4920: loss 1.2160, time 183.83ms, mfu 2.00%
iter 4930: loss 1.2225, time 182.64ms, mfu 2.00%
iter 4940: loss 1.2073, time 183.69ms, mfu 2.00%
iter 4950: loss 1.2220, time 182.88ms, mfu 2.01%
iter 4960: loss 1.1861, time 183.76ms, mfu 2.01%
iter 4970: loss 1.2046, time 183.34ms, mfu 2.01%
iter 4980: loss 1.2116, time 183.20ms, mfu 2.01%
iter 4990: loss 1.2036, time 183.95ms, mfu 2.02%
```

Exercise 4

Run inference on the model. Complete the TODO portions

1. You need to call `model.generate` in the given for loop.
2. Show 10 samples. These might not be perfectly sensible English, but they should be very Shakespeare-like. Make sure they can be read in your submitted PDF.

```
#
-----------------------------------------------------------------------------
start = "\n" # or "<|endoftext|>" or etc. Can also specify a file, use
as: "FILE:prompt.txt"
num_samples = 10 # number of samples to draw
max_new_tokens = 500 # number of tokens generated in each sample
temperature = 0.8 # 1.0 = no change, < 1.0 = less random, > 1.0 = more
random, in predictions
top_k = 200 # retain only the top_k most likely tokens, clamp others
to have 0 probability
#
-----------------------------------------------------------------------------

model.eval()

# get the absolute path of the current working directory
current_dir = os.getcwd()

# construct the relative path to the meta.pkl file
meta_path = os.path.join(current_dir, 'data', 'shakespeare',
'meta.pkl')
```

```
print(f"Loading meta from meta.pkl...")
with open(meta_path, 'rb') as f:
    meta = pickle.load(f)
stoi, itos = meta['stoi'], meta['itos']
encode = lambda s: [stoi[c] for c in s]
decode = lambda l: ''.join([itos[i] for i in l])

# encode the beginning of the prompt
if start.startswith('FILE:'):
    with open(start[5:], 'r', encoding='utf-8') as f:
        start = f.read()
start_ids = encode(start)
x = (torch.tensor(start_ids, dtype=torch.long, device=device)
[None, ...])
```

```
Loading meta from meta.pkl...
```

```
# run generation
with torch.no_grad():
    with ctx:
        for k in range(num_samples):
            #TODO: Generate the sample
            generated_text = model.generate(x, max_new_tokens,
temperature=temperature, top_k=top_k)
            # decode the generated text
            decoded_text = decode(generated_text[0].tolist())
            print(f"Sample {k+1}:\n{decoded_text}\n{'-'*80}")
```

```
Sample 1:

The state against the last death,
And but the newless of night as he would
Were adviced, and supplied the environ,
And not yield so fair as having stain'd.
Would you know he do the king; and I shall rest,
That given cried to a blood creature little hands
By being the breath of the people guests.
But, be pale-enchilated and sorrow;
And therefore, to your suitors, that I will not say.

CLARENCE:
No, by the Volsces are they not stand their heads,
To strong their and royal place of grace,
And strike
----------------------------------------------------------------------
----------
Sample 2:

Pardon for your own prison; you'll bring you, fair as
```

```
your great can be prepared of your true.

DUKE VINCENTIO:
What, for you mean? what will you are?

LUCIO:
Why, you can please you have a consul? Can you
would plead in this mother king?
The fellow I am gone?

ISABELLA:
Why, I cannot speak in the Tower. What are yours?

DUKE VINCENTIO:
You will hear you our two?

ANGELO:
Sir, perform your comfort: you know me here in a
man for foot of a mirth, if you have done swear to me,
there beholding in po
----------------------------------------------------------------------
----------
Sample 3:



Clown:
What, sir? why can thou, then? that's the cates?

Second Gentleman:
Ay, he's a word, will stay. I will not where we say.

Clown:
No, sir.

AUTOLYCUS:
I pray, may be the senators a buart: I have not the
man words for the cunning for soundly should be
any that thought which, whose goes report on the ears: he are not
begun and first violengeance. Here's a bawd of his party choose
sings, the poor own prince-hostes; the volume to his
bigger than sentences: his discovery him like
me such a sub
----------------------------------------------------------------------
----------
Sample 4:



ANGELO:
O thou comest that is a well-half as merry
```

```
woman of the mistresses of the sense of the charmion.

ISABELLA:
Yes, so that are set now after how it now
The pruns of the man thereof this blood and speech sing
Contends to the sulluship in the earth
Of what may I see it seem to be speak,
I know that I is so many rich with you.

DUKE VINCENTIO:
I'll not take to see alone.

HASTINGS:
No, thou wilt stand to the news of my mind,
And to this precious eyes can to expose your strength.

DUKE VINCEN
----------------------------------------------------------------------
----------
Sample 5:

I'll not shame the bosom of a chamber.

AUTOLYCUS:
The poor is so; and we did such as the king
As if I do leave him to part.

Clown:
It is my careful true find upon him; he have been
seen it.

Clown:
What a hath a self-of a tribune, he shall be queen
brief the one and house, he did such a present of
this gentleman that was revenge; but I'll know
his favour hand and honour to his body, and but
what he is a state, I heard it for
him, you will not have well been a good world of her
brother: I wish
----------------------------------------------------------------------
----------
Sample 6:

That his father died by so bloody in Rome,
And thou didst repose the end heaven of thy bed,
In the bloody royalty of thy watch,
Having not a day day's for the blood
Of my foe death dead of love to the banishment?
```

```
EDWARD:
To make him to thee, or what he was the grace
But what then? But what are slain a country's grace?

KING EDWARD IV:
If he confinent himself, or else what slain we may.

LADY GREY:
What, are you say it the heart of York:
For York doth trembling in my fortune's house,
That may be
----------------------------------------------------------------------
----------
Sample 7:

If you do not speak: and let it speak again.

First Senator:
A poor children, sir, I shall maech you see at this!

Second Murderer:
My lord, 'tis no other recoice for me.

Clown:
I know not a single back and slaughter is present.

Clown:
I am a word of that she has heard that done but he did;
and the state still you good to be player.

Second Murderer:
Why, to-morrow; rash my knocks will I die;
And let me the house of all this issue of the daughter
Than every own strange and that in the sins.

S
----------------------------------------------------------------------
----------
Sample 8:


DUCHESS OF YORK:
Why, had I done, my lord?

DUCHESS OF YORK:
Though the father France, what desire,
For thy tongue to make my soul? What means have you blamed?

KING RICHARD III:
```

```
How now, boy, if thou canst be contented
To what is not long his conqueror?
Why lords Thereford? what says the queen?

QUEEN MARGARET:
Ay, brave my head, and what did I repute my great?

KING RICHARD III:
Then, thou shalt die not such a modeless?

QUEEN MARGARET:
I have founded to Angelo,
Is rash better thee and more p
----------------------------------------------------------------------
----------
Sample 9:

When he shall be Rome this stars, they lived
his spatch ancient and one reported by some
suitors and stock enemies.

COMINIUS:
So, unknown the great princes
Why, by his officers, who well not had been
March at the matter.

MENENIUS:
What?
Thou wilt do?

SICINIUS:
Be reason'd the powerful plague?

MENENIUS:
Tranio, let's see him: he is born so four
obsequires and a cheek-time
Freder his patience, I speak, and so fellow have you
Becomes to perform him.

MENENIUS:
O my lord,
His prince, though not
----------------------------------------------------------------------
----------
Sample 10:

From us before we have sent the wind of world.

First Lord:
```

```
So hath cured my son of behaving kingdom's nature?

KING RICHARD III:
Within the forest request brother may be gone.
Ah, an as he new my prepared man?

CLARENCE:
When I have heard this time I did? O that all him!
Who is the father drops to a Jelory's kingdom?

WARWICK:
Stay, and until thou under where I seem to be Henry?

QUEEN MARGARET:
Then I have thee to justice of Warwick's day,
And what a shallow was the sun mine are that shed
That
----------------------------------------------------------------------
----------
```

Exercise 5: Train the model on a new dataset and show results.

This exercise is mostly about making sure you can find and preprocess text, as well as checking that you understand the above code well enough to reuse it.

1. Find some text data. Use our Shakespeare file as reference. You will want a similar amount of text data. Don't go overboard- a big text file will just make things take too long.
2. Perform any preprocessing necessary to get the text ready for the model. Use the preprocessing code we provide as reference.
3. Train the model on your text.
4. Generate and print 10 samples from the model trained on your text.

You may want to implement the functions below, using the code in the previous cells. Or not! It's up to you. You just need to write code that can train a model to generate text from some non-Shakespeare data. The generated text is the main deliverable that most of the grade will be based on. Make sure it displays prominently in your submitted PDF.

```python
import os
import requests
import tarfile
import email
from email import policy
import numpy as np
import pickle

def download_data():
    """
    - Downloads the Enron Email dataset
    - saves it to the data folder
    - preprocesses the dataset
    - saves the preprocessed dataset to input.txt, train.bin, val.bin,
meta.pkl
    - saves the meta data to meta.pkl
    - Provides the statistics of the dataset
    """

    # Download and extract the dataset
    url = "https://www.cs.cmu.edu/~./enron/enron_mail_20150507.tar.gz"
    filename = "enron_mail_20150507.tgz"
    folder = "maildir" # Manually considering only 51 users data for
simplicity

    if not os.path.exists(filename):
        response = requests.get(url)
        open(filename, 'wb').write(response.content)

    if not os.path.exists(folder):
        with tarfile.open(filename, "r:gz") as tar:
            tar.extractall()

    # Preprocess the dataset
    def extract_text_from_email(email_file):
        with open(email_file, 'r', encoding='utf-8', errors='ignore')
as f:
            msg = email.message_from_file(f, policy=policy.default)
            return
msg.get_body(preferencelist=('plain')).get_content()

    def preprocess_enron_emails(data_folder):
        texts = []
        for root, _, files in os.walk(data_folder):
            for file in files:
                email_path = os.path.join(root, file)
                try:
                    email_text = extract_text_from_email(email_path)
                    texts.append(email_text)
                except Exception as e:
                    print(f"Error processing {email_path}: {e}")
        return "\n".join(texts)

    def encode(s):
        return [stoi[c] for c in s] # encoder: take a string, output a
list of integers

    def decode(l):
        return ''.join([itos[i] for i in l]) # decoder: take a list of
integers, output a string
```

```
    email_data = preprocess_enron_emails(folder)

    if not os.path.exists('data/enron_data'):
        os.makedirs('data/enron_data')

    # save the data to a file as input.txt
    with open('data/enron_data/input.txt', 'w', encoding='utf-8') as
f:
        f.write(email_data)

    # save the meta information as well, to help us encode/decode
later
    data_root = 'data/enron_data'

    # Prepare the dataset
    data = email_data
    print(f"length of dataset in characters: {len(data):,}")

    # get all the unique characters that occur in this text
    chars = sorted(list(set(data)))
    vocab_size = len(chars)
    print("all the unique characters:", ''.join(chars))
    print(f"vocab size: {vocab_size:,}")

    # create a mapping from characters to integers
    stoi = { ch:i for i,ch in enumerate(chars) }
    itos = { i:ch for i,ch in enumerate(chars) }

    # create the train and test splits
    n = len(data)
    train_data = data[:int(n*0.9)]
    val_data = data[int(n*0.9):]

    # encode both to integers
    train_ids = encode(train_data)
    val_ids = encode(val_data)
    print(f"train has {len(train_ids):,} tokens")
    print(f"val has {len(val_ids):,} tokens")

    # export to bin files
    train_ids = np.array(train_ids, dtype=np.uint16)
    val_ids = np.array(val_ids, dtype=np.uint16)
    train_ids.tofile(os.path.join(data_root, 'train.bin'))
    val_ids.tofile(os.path.join(data_root, 'val.bin'))

    # save the meta information as well, to help us encode/decode
later
    meta = {
        'vocab_size': vocab_size,
        'itos': itos,
        'stoi': stoi,
    }
    with open(f'{data_root}/meta.pkl', 'wb') as f:
        pickle.dump(meta, f)


def train_model():
    """Train the model that is defined in Exercise 1 on your train
data"""

    # Email generation model
    os.makedirs(out_dir_email, exist_ok=True)
    torch.manual_seed(1337 + seed_offset)
    torch.backends.cuda.matmul.allow_tf32 = True
    torch.backends.cudnn.allow_tf32 = True
    device_type = 'cuda' if 'cuda' in device else 'cpu'
    ptdtype = {'float32': torch.float32, 'float16': torch.float16}
[dtype]
    ctx = nullcontext() if device_type == 'cpu' else
torch.amp.autocast(device_type=device_type, dtype=ptdtype)

    # load the email dataset and create batches
    data_dir = os.path.join('data', dataset_email)
    train_data = np.memmap(os.path.join(data_dir, 'train.bin'),
dtype=np.uint16, mode='r')
    val_data = np.memmap(os.path.join(data_dir, 'val.bin'),
dtype=np.uint16, mode='r')
    def get_batch(split):
        data = train_data if split == 'train' else val_data
        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.int
64)) for i in ix])
        if device_type == 'cuda':
            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

    iter_num = 0
    best_val_loss = 1e9

    meta_path = os.path.join(data_dir, 'meta.pkl')
    meta_vocab_size = None
    if os.path.exists(meta_path):
        with open(meta_path, 'rb') as f:
            meta = pickle.load(f)
        meta_vocab_size = meta['vocab_size']
        print(f"found vocab_size = {meta_vocab_size} (inside
{meta_path})")

    # model init
    model_args = dict(n_layer=n_layer, n_head=n_head, n_embd=n_embd,
block_size=block_size,
                      bias=bias, vocab_size=None, dropout=dropout)
    model_args['vocab_size'] = meta_vocab_size
    gptconf = GPTConfig(**model_args)
    model = GPT(gptconf)
    if block_size < model.config.block_size:
        model.crop_block_size(block_size)
        model_args['block_size'] = block_size
    model.to(device)

    scaler = torch.cuda.amp.GradScaler(enabled=(dtype == 'float16'))

    # optimizer
    optimizer = model.configure_optimizers(weight_decay,
learning_rate, (beta1, beta2), device_type)
    checkpoint = None

    # compile the model
    if compile:
        print("compiling the model... (takes a ~minute)")
        unoptimized_model = model
        model = torch.compile(model, backend='triton') # requires
PyTorch 2.0

    @torch.no_grad()
    def estimate_loss():
        out = {}
        model.eval()
        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

    # training loop
    X, Y = get_batch('train') # fetch the very first batch
    t0 = time.time()
    local_iter_num = 0 # number of iterations in the lifetime of this 
process
    raw_model = model
    running_mfu = -1.0
    for iter_num in range(max_iters):
        lr = get_lr(iter_num) if decay_lr else learning_rate
        for param_group in optimizer.param_groups:
            param_group['lr'] = lr

        if iter_num % eval_interval == 0:
            losses = estimate_loss()
            print(f"step {iter_num}: train loss {losses['train']:.4f}, 
val loss {losses['val']:.4f}")
            if losses['val'] < best_val_loss or 
always_save_checkpoint:
                best_val_loss = losses['val']
                if iter_num > 0:
                    checkpoint = {
                        'model': raw_model.state_dict(),
                        'optimizer': optimizer.state_dict(),
                        'model_args': model_args,
                        'iter_num': iter_num,
                        'best_val_loss': best_val_loss
                    }
                    print(f"saving checkpoint to {out_dir_email}")
                    torch.save(checkpoint, os.path.join(out_dir_email, 
'emailGPT_ckpt.pt'))
        if iter_num == 0 and eval_only:
            break

        for micro_step in range(gradient_accumulation_steps):
            with ctx:
                logits, loss = model(X, Y)
                loss = loss / gradient_accumulation_steps
            X, Y = get_batch('train')
            scaler.scale(loss).backward()
        if grad_clip != 0.0:
            torch.nn.utils.clip_grad_norm_(model.parameters(), 
grad_clip)
        scaler.step(optimizer)
        scaler.update()
        optimizer.zero_grad(set_to_none=True)

        # timing and logging
        t1 = time.time()
        dt = t1 - t0
        t0 = t1
        if iter_num % log_interval == 0:
            lossf = loss.item() * gradient_accumulation_steps
            if local_iter_num >= 5: # let the training loop settle a 
```

```
bit
                mfu = raw_model.estimate_mfu(batch_size *
gradient_accumulation_steps, dt)
                running_mfu = mfu if running_mfu == -1.0 else
0.9*running_mfu + 0.1*mfu
            print(f"iter {iter_num}: loss {lossf:.4f}, time
{dt*1000:.2f}ms, mfu {running_mfu*100:.2f}%")
        local_iter_num += 1

    return model


def eval_model(model):
    """Runs inference of the trained model on your test data"""

    #
----------------------------------------------------------------------
-------
    start = "\n" # or "<|endoftext|>" or etc. Can also specify a file,
use as: "FILE:prompt.txt"
    num_samples = 11 # number of samples to draw
    max_new_tokens = 1000 # number of tokens generated in each sample
    temperature = 0.8 # 1.0 = no change, < 1.0 = less random, > 1.0 =
more random, in predictions
    top_k = 200 # retain only the top_k most likely tokens, clamp
others to have 0 probability
    #
----------------------------------------------------------------------
-------

    model.eval()

    # get the absolute path of the current working directory
    current_dir = os.getcwd()

    # construct the relative path to the meta.pkl file
    meta_path = os.path.join(current_dir, 'data', 'enron_data',
'meta.pkl')

    print(f"Loading meta from meta.pkl...")
    with open(meta_path, 'rb') as f:
        meta = pickle.load(f)
    stoi, itos = meta['stoi'], meta['itos']
    encode = lambda s: [stoi[c] for c in s]
    decode = lambda l: ''.join([itos[i] for i in l])

    # encode the beginning of the prompt
    if start.startswith('FILE:'):
        with open(start[5:], 'r', encoding='utf-8') as f:
            start = f.read()
    start_ids = encode(start)
    x = (torch.tensor(start_ids, dtype=torch.long, device=device)
[None, ...])
    # run generation
    with torch.no_grad():
        with ctx:
            for k in range(num_samples):
                #TODO: Generate the sample
                generated_text = model.generate(x, max_new_tokens,
temperature=temperature, top_k=top_k)
                # decode the generated text
                decoded_text = decode(generated_text[0].tolist())
                print(f"Sample {k+1}:\n{decoded_text}\n{'-'*80}")
```

```
# Download the data and preprocess it for training and evaluation of
the model
download_data()
```

```
length of dataset in characters: 337,409,058
all the unique characters:
 !"#$%&'()*+,-./0123456789:;<=>?
@ABCDEFGHIJKLMNOPQRSTUVWXYZ[\]_`abcdefghijklmnopqrstuvwxyz{|}~□
vocab size: 117
train has 303,668,152 tokens
val has 33,740,906 tokens
```

```
# Train the model
# expected loss should start from -ln(1/117) = 4.762
trained_model = train_model()
```

```
found vocab_size = 117 (inside data/enron_data/meta.pkl)
number of parameters: 10.67M
using fused AdamW: True
step 0: train loss 4.7417, val loss 4.7410
iter 0: loss 4.7292, time 26248.32ms, mfu -100.00%
iter 10: loss 3.5903, time 182.76ms, mfu 2.04%
iter 20: loss 3.1500, time 183.13ms, mfu 2.04%
iter 30: loss 2.9429, time 183.48ms, mfu 2.04%
iter 40: loss 3.0150, time 182.58ms, mfu 2.04%
iter 50: loss 2.8258, time 182.75ms, mfu 2.04%
iter 60: loss 2.7966, time 183.83ms, mfu 2.04%
iter 70: loss 2.8596, time 183.04ms, mfu 2.04%
iter 80: loss 2.7788, time 183.06ms, mfu 2.04%
iter 90: loss 2.7965, time 181.96ms, mfu 2.04%
iter 100: loss 2.7575, time 182.55ms, mfu 2.04%
iter 110: loss 2.7725, time 182.67ms, mfu 2.04%
iter 120: loss 2.8021, time 182.69ms, mfu 2.04%
iter 130: loss 2.6917, time 183.08ms, mfu 2.04%
iter 140: loss 2.6610, time 183.54ms, mfu 2.04%
iter 150: loss 2.7197, time 182.84ms, mfu 2.04%
iter 160: loss 2.6077, time 182.19ms, mfu 2.04%
iter 170: loss 2.5853, time 182.25ms, mfu 2.04%
iter 180: loss 2.6258, time 182.37ms, mfu 2.04%
iter 190: loss 2.6928, time 182.19ms, mfu 2.04%
iter 200: loss 2.6156, time 182.52ms, mfu 2.04%
iter 210: loss 2.6507, time 183.20ms, mfu 2.04%
iter 220: loss 2.6042, time 182.78ms, mfu 2.04%
iter 230: loss 2.5460, time 181.93ms, mfu 2.04%
iter 240: loss 2.6254, time 183.56ms, mfu 2.04%
step 250: train loss 2.5949, val loss 2.5532
saving checkpoint to out-enron-email
iter 250: loss 2.6451, time 26486.10ms, mfu 1.84%
iter 260: loss 2.6283, time 184.17ms, mfu 1.86%
iter 270: loss 2.6253, time 182.86ms, mfu 1.88%
iter 280: loss 2.5444, time 183.56ms, mfu 1.89%
iter 290: loss 2.6021, time 182.77ms, mfu 1.91%
iter 300: loss 2.5186, time 184.19ms, mfu 1.92%
iter 310: loss 2.6165, time 183.28ms, mfu 1.93%
iter 320: loss 2.4184, time 182.83ms, mfu 1.94%
iter 330: loss 2.4992, time 182.66ms, mfu 1.95%
iter 340: loss 2.5768, time 183.35ms, mfu 1.96%
iter 350: loss 2.5184, time 182.84ms, mfu 1.97%
iter 360: loss 2.5773, time 182.85ms, mfu 1.98%
iter 370: loss 2.5187, time 183.41ms, mfu 1.98%
iter 380: loss 2.5687, time 182.79ms, mfu 1.99%
iter 390: loss 2.6203, time 182.96ms, mfu 1.99%
iter 400: loss 2.5547, time 182.58ms, mfu 2.00%
iter 410: loss 2.4362, time 183.62ms, mfu 2.00%
iter 420: loss 2.5128, time 182.91ms, mfu 2.01%
iter 430: loss 2.4964, time 183.00ms, mfu 2.01%
iter 440: loss 2.4568, time 182.70ms, mfu 2.01%
iter 450: loss 2.4861, time 182.81ms, mfu 2.02%
iter 460: loss 2.5007, time 182.98ms, mfu 2.02%
iter 470: loss 2.3988, time 182.74ms, mfu 2.02%
iter 480: loss 2.4657, time 182.95ms, mfu 2.02%
iter 490: loss 2.4345, time 183.40ms, mfu 2.02%
step 500: train loss 2.3967, val loss 2.3440
saving checkpoint to out-enron-email
iter 500: loss 2.4603, time 26476.10ms, mfu 1.82%
iter 510: loss 2.4392, time 183.38ms, mfu 1.84%
iter 520: loss 2.4819, time 182.76ms, mfu 1.86%
iter 530: loss 2.4396, time 184.24ms, mfu 1.88%
iter 540: loss 2.3798, time 190.83ms, mfu 1.89%
iter 550: loss 2.4098, time 184.08ms, mfu 1.90%
iter 560: loss 2.4034, time 182.96ms, mfu 1.92%
iter 570: loss 2.3709, time 183.64ms, mfu 1.93%
iter 580: loss 2.4751, time 182.46ms, mfu 1.94%
iter 590: loss 2.4001, time 184.29ms, mfu 1.95%
iter 600: loss 2.2948, time 187.30ms, mfu 1.95%
iter 610: loss 2.3500, time 182.40ms, mfu 1.96%
iter 620: loss 2.3356, time 182.66ms, mfu 1.97%
iter 630: loss 2.3912, time 182.80ms, mfu 1.98%
iter 640: loss 2.4253, time 184.84ms, mfu 1.98%
iter 650: loss 2.3039, time 183.65ms, mfu 1.99%
iter 660: loss 2.3987, time 182.82ms, mfu 1.99%
iter 670: loss 2.3001, time 183.22ms, mfu 2.00%
iter 680: loss 2.2664, time 183.07ms, mfu 2.00%
iter 690: loss 2.2821, time 183.33ms, mfu 2.00%
iter 700: loss 2.2860, time 182.88ms, mfu 2.01%
iter 710: loss 2.3267, time 183.07ms, mfu 2.01%
iter 720: loss 2.3382, time 184.14ms, mfu 2.01%
iter 730: loss 2.2500, time 182.59ms, mfu 2.02%
iter 740: loss 2.3393, time 182.60ms, mfu 2.02%
step 750: train loss 2.1304, val loss 2.0932
saving checkpoint to out-enron-email
iter 750: loss 2.3403, time 26447.17ms, mfu 1.82%
iter 760: loss 2.2621, time 183.49ms, mfu 1.84%
iter 770: loss 2.2428, time 183.27ms, mfu 1.86%
iter 780: loss 2.1465, time 182.24ms, mfu 1.88%
iter 790: loss 2.1995, time 182.99ms, mfu 1.89%
iter 800: loss 2.2816, time 183.20ms, mfu 1.91%
iter 810: loss 2.2449, time 183.83ms, mfu 1.92%
iter 820: loss 2.1201, time 183.73ms, mfu 1.93%
iter 830: loss 2.1806, time 183.08ms, mfu 1.94%
iter 840: loss 2.1995, time 183.60ms, mfu 1.95%
iter 850: loss 2.2455, time 183.01ms, mfu 1.96%
iter 860: loss 2.1232, time 183.23ms, mfu 1.97%
iter 870: loss 2.1706, time 182.69ms, mfu 1.98%
iter 880: loss 2.2255, time 183.07ms, mfu 1.98%
iter 890: loss 2.1405, time 182.74ms, mfu 1.99%
iter 900: loss 2.0414, time 183.67ms, mfu 1.99%
iter 910: loss 2.1523, time 183.77ms, mfu 2.00%
iter 920: loss 2.1354, time 182.50ms, mfu 2.00%
iter 930: loss 2.0387, time 183.13ms, mfu 2.00%
iter 940: loss 2.1022, time 183.95ms, mfu 2.01%
iter 950: loss 2.1369, time 182.83ms, mfu 2.01%
iter 960: loss 2.0675, time 182.75ms, mfu 2.01%
iter 970: loss 2.0092, time 183.71ms, mfu 2.02%
iter 980: loss 2.0815, time 182.69ms, mfu 2.02%
iter 990: loss 2.1002, time 182.97ms, mfu 2.02%
step 1000: train loss 1.9332, val loss 1.8924
saving checkpoint to out-enron-email
iter 1000: loss 2.1650, time 26386.92ms, mfu 1.82%
iter 1010: loss 2.0918, time 182.55ms, mfu 1.84%
iter 1020: loss 1.9921, time 182.55ms, mfu 1.86%
iter 1030: loss 2.0251, time 182.77ms, mfu 1.88%
iter 1040: loss 2.0774, time 182.73ms, mfu 1.90%
iter 1050: loss 2.0580, time 184.13ms, mfu 1.91%
iter 1060: loss 2.0656, time 182.61ms, mfu 1.92%
iter 1070: loss 1.9783, time 183.58ms, mfu 1.93%
iter 1080: loss 2.0790, time 183.05ms, mfu 1.94%
iter 1090: loss 2.0300, time 182.88ms, mfu 1.95%
iter 1100: loss 1.9869, time 182.96ms, mfu 1.96%
iter 1110: loss 1.9094, time 183.54ms, mfu 1.97%
iter 1120: loss 1.9152, time 184.31ms, mfu 1.98%
iter 1130: loss 1.9671, time 183.71ms, mfu 1.98%
iter 1140: loss 1.9581, time 182.67ms, mfu 1.99%
iter 1150: loss 1.8961, time 183.59ms, mfu 1.99%
iter 1160: loss 1.8669, time 184.52ms, mfu 2.00%
iter 1170: loss 1.8886, time 182.71ms, mfu 2.00%
iter 1180: loss 1.9191, time 182.76ms, mfu 2.00%
iter 1190: loss 1.8679, time 183.25ms, mfu 2.01%
iter 1200: loss 1.8738, time 183.00ms, mfu 2.01%
iter 1210: loss 1.9253, time 183.23ms, mfu 2.01%
iter 1220: loss 1.9376, time 183.21ms, mfu 2.02%
iter 1230: loss 1.8689, time 183.72ms, mfu 2.02%
iter 1240: loss 1.9405, time 183.84ms, mfu 2.02%
step 1250: train loss 1.7685, val loss 1.7327
saving checkpoint to out-enron-email
iter 1250: loss 1.8869, time 26403.24ms, mfu 1.82%
iter 1260: loss 1.9193, time 183.45ms, mfu 1.84%
iter 1270: loss 1.9207, time 182.40ms, mfu 1.86%
iter 1280: loss 1.8736, time 183.92ms, mfu 1.88%
iter 1290: loss 1.8123, time 182.78ms, mfu 1.89%
iter 1300: loss 1.9100, time 183.22ms, mfu 1.91%
iter 1310: loss 1.8997, time 183.28ms, mfu 1.92%
iter 1320: loss 1.9676, time 183.14ms, mfu 1.93%
iter 1330: loss 1.8895, time 182.74ms, mfu 1.94%
iter 1340: loss 1.8756, time 183.46ms, mfu 1.95%
iter 1350: loss 1.8906, time 183.04ms, mfu 1.96%
iter 1360: loss 1.8714, time 183.36ms, mfu 1.97%
iter 1370: loss 1.7464, time 183.76ms, mfu 1.98%
iter 1380: loss 1.7820, time 183.39ms, mfu 1.98%
iter 1390: loss 1.8243, time 183.14ms, mfu 1.99%
iter 1400: loss 1.8424, time 182.71ms, mfu 1.99%
iter 1410: loss 1.9279, time 182.40ms, mfu 2.00%
iter 1420: loss 1.8124, time 182.58ms, mfu 2.00%
iter 1430: loss 1.8284, time 182.74ms, mfu 2.01%
iter 1440: loss 1.8566, time 184.26ms, mfu 2.01%
iter 1450: loss 1.7720, time 182.22ms, mfu 2.01%
iter 1460: loss 1.8079, time 183.55ms, mfu 2.01%
iter 1470: loss 1.7474, time 183.58ms, mfu 2.02%
iter 1480: loss 1.7318, time 183.72ms, mfu 2.02%
iter 1490: loss 1.7884, time 183.29ms, mfu 2.02%
step 1500: train loss 1.6555, val loss 1.6368
saving checkpoint to out-enron-email
iter 1500: loss 1.7698, time 26419.10ms, mfu 1.82%
iter 1510: loss 1.7418, time 184.35ms, mfu 1.84%
iter 1520: loss 1.7730, time 184.38ms, mfu 1.86%
iter 1530: loss 1.6776, time 184.35ms, mfu 1.87%
iter 1540: loss 1.7996, time 183.11ms, mfu 1.89%
iter 1550: loss 1.7371, time 183.96ms, mfu 1.90%
iter 1560: loss 1.6778, time 183.33ms, mfu 1.92%
iter 1570: loss 1.7458, time 182.54ms, mfu 1.93%
iter 1580: loss 1.7699, time 182.86ms, mfu 1.94%
iter 1590: loss 1.7225, time 183.85ms, mfu 1.95%
iter 1600: loss 1.6966, time 182.63ms, mfu 1.96%
iter 1610: loss 1.6240, time 183.14ms, mfu 1.97%
iter 1620: loss 1.7876, time 183.48ms, mfu 1.97%
iter 1630: loss 1.7084, time 182.53ms, mfu 1.98%
iter 1640: loss 1.6565, time 184.35ms, mfu 1.99%
iter 1650: loss 1.7768, time 183.49ms, mfu 1.99%
iter 1660: loss 1.7324, time 182.61ms, mfu 2.00%
iter 1670: loss 1.6698, time 183.12ms, mfu 2.00%
iter 1680: loss 1.7663, time 183.25ms, mfu 2.00%
iter 1690: loss 1.7754, time 182.87ms, mfu 2.01%
iter 1700: loss 1.7458, time 183.76ms, mfu 2.01%
iter 1710: loss 1.6828, time 183.23ms, mfu 2.01%
iter 1720: loss 1.6581, time 183.39ms, mfu 2.01%
iter 1730: loss 1.6932, time 182.54ms, mfu 2.02%
iter 1740: loss 1.7063, time 182.69ms, mfu 2.02%
step 1750: train loss 1.5891, val loss 1.5610
saving checkpoint to out-enron-email
iter 1750: loss 1.6834, time 26440.38ms, mfu 1.82%
iter 1760: loss 1.6579, time 182.71ms, mfu 1.84%
iter 1770: loss 1.7010, time 182.81ms, mfu 1.86%
iter 1780: loss 1.7466, time 183.86ms, mfu 1.88%
iter 1790: loss 1.7405, time 182.78ms, mfu 1.90%
iter 1800: loss 1.6717, time 183.47ms, mfu 1.91%
iter 1810: loss 1.7150, time 184.59ms, mfu 1.92%
iter 1820: loss 1.7851, time 182.31ms, mfu 1.93%
iter 1830: loss 1.6924, time 182.20ms, mfu 1.94%
iter 1840: loss 1.6129, time 183.81ms, mfu 1.95%
iter 1850: loss 1.7337, time 182.39ms, mfu 1.96%
iter 1860: loss 1.6738, time 183.37ms, mfu 1.97%
iter 1870: loss 1.6764, time 183.54ms, mfu 1.98%
iter 1880: loss 1.5892, time 182.76ms, mfu 1.98%
iter 1890: loss 1.7490, time 182.92ms, mfu 1.99%
iter 1900: loss 1.6216, time 183.01ms, mfu 1.99%
iter 1910: loss 1.6769, time 182.81ms, mfu 2.00%
iter 1920: loss 1.7001, time 182.80ms, mfu 2.00%
iter 1930: loss 1.7125, time 183.32ms, mfu 2.01%
iter 1940: loss 1.6026, time 182.95ms, mfu 2.01%
iter 1950: loss 1.6274, time 184.39ms, mfu 2.01%
iter 1960: loss 1.6377, time 183.39ms, mfu 2.01%
iter 1970: loss 1.5659, time 184.57ms, mfu 2.01%
iter 1980: loss 1.5730, time 183.22ms, mfu 2.02%
iter 1990: loss 1.6230, time 182.74ms, mfu 2.02%
step 2000: train loss 1.5288, val loss 1.5154
saving checkpoint to out-enron-email
iter 2000: loss 1.5537, time 26698.84ms, mfu 1.82%
iter 2010: loss 1.6242, time 183.36ms, mfu 1.84%
iter 2020: loss 1.6975, time 183.60ms, mfu 1.86%
iter 2030: loss 1.5312, time 182.79ms, mfu 1.88%
iter 2040: loss 1.5693, time 183.63ms, mfu 1.89%
iter 2050: loss 1.6259, time 182.80ms, mfu 1.91%
iter 2060: loss 1.6493, time 183.63ms, mfu 1.92%
iter 2070: loss 1.5244, time 182.51ms, mfu 1.93%
iter 2080: loss 1.6052, time 182.63ms, mfu 1.94%
iter 2090: loss 1.6184, time 182.65ms, mfu 1.95%
iter 2100: loss 1.6004, time 182.62ms, mfu 1.96%
iter 2110: loss 1.6265, time 182.73ms, mfu 1.97%
iter 2120: loss 1.5573, time 183.23ms, mfu 1.98%
iter 2130: loss 1.5988, time 182.74ms, mfu 1.98%
iter 2140: loss 1.5975, time 182.64ms, mfu 1.99%
iter 2150: loss 1.6521, time 182.62ms, mfu 2.00%
iter 2160: loss 1.5312, time 190.81ms, mfu 1.99%
iter 2170: loss 1.5099, time 183.77ms, mfu 2.00%
iter 2180: loss 1.6060, time 182.96ms, mfu 2.00%
iter 2190: loss 1.7045, time 182.87ms, mfu 2.00%
iter 2200: loss 1.6227, time 185.28ms, mfu 2.01%
iter 2210: loss 1.5729, time 183.13ms, mfu 2.01%
iter 2220: loss 1.6373, time 183.59ms, mfu 2.01%
iter 2230: loss 1.6319, time 185.13ms, mfu 2.01%
iter 2240: loss 1.6493, time 182.69ms, mfu 2.01%
step 2250: train loss 1.4773, val loss 1.4643
saving checkpoint to out-enron-email
iter 2250: loss 1.5681, time 26470.31ms, mfu 1.81%
iter 2260: loss 1.5664, time 182.36ms, mfu 1.84%
iter 2270: loss 1.5603, time 183.97ms, mfu 1.86%
iter 2280: loss 1.5360, time 182.93ms, mfu 1.88%
iter 2290: loss 1.5772, time 187.38ms, mfu 1.89%
iter 2300: loss 1.5077, time 182.67ms, mfu 1.90%
iter 2310: loss 1.5578, time 182.49ms, mfu 1.92%
iter 2320: loss 1.5092, time 182.74ms, mfu 1.93%
iter 2330: loss 1.6489, time 182.52ms, mfu 1.94%
iter 2340: loss 1.5765, time 182.54ms, mfu 1.95%
iter 2350: loss 1.5858, time 186.38ms, mfu 1.96%
iter 2360: loss 1.5853, time 183.25ms, mfu 1.96%
iter 2370: loss 1.6041, time 183.32ms, mfu 1.97%
iter 2380: loss 1.5245, time 184.31ms, mfu 1.98%
iter 2390: loss 1.6289, time 183.26ms, mfu 1.98%
iter 2400: loss 1.5373, time 182.83ms, mfu 1.99%
iter 2410: loss 1.5322, time 183.21ms, mfu 1.99%
iter 2420: loss 1.5932, time 182.84ms, mfu 2.00%
iter 2430: loss 1.5674, time 182.83ms, mfu 2.00%
iter 2440: loss 1.5192, time 182.35ms, mfu 2.01%
iter 2450: loss 1.5729, time 183.73ms, mfu 2.01%
iter 2460: loss 1.5736, time 184.10ms, mfu 2.01%
iter 2470: loss 1.5751, time 182.90ms, mfu 2.01%
iter 2480: loss 1.5141, time 183.67ms, mfu 2.02%
iter 2490: loss 1.5990, time 182.48ms, mfu 2.02%
step 2500: train loss 1.4550, val loss 1.4464
saving checkpoint to out-enron-email
iter 2500: loss 1.5387, time 26452.90ms, mfu 1.82%
iter 2510: loss 1.5800, time 183.08ms, mfu 1.84%
iter 2520: loss 1.5019, time 188.80ms, mfu 1.85%
iter 2530: loss 1.5440, time 183.30ms, mfu 1.87%
iter 2540: loss 1.6197, time 183.51ms, mfu 1.89%
iter 2550: loss 1.5255, time 184.08ms, mfu 1.90%
iter 2560: loss 1.4846, time 183.28ms, mfu 1.92%
iter 2570: loss 1.5162, time 182.67ms, mfu 1.93%
iter 2580: loss 1.5703, time 182.32ms, mfu 1.94%
iter 2590: loss 1.5584, time 183.46ms, mfu 1.95%
iter 2600: loss 1.5102, time 182.67ms, mfu 1.96%
iter 2610: loss 1.4478, time 182.65ms, mfu 1.97%
iter 2620: loss 1.5909, time 182.82ms, mfu 1.98%
iter 2630: loss 1.6133, time 182.70ms, mfu 1.98%
iter 2640: loss 1.5091, time 182.65ms, mfu 1.99%
iter 2650: loss 1.5404, time 182.51ms, mfu 1.99%
iter 2660: loss 1.5696, time 182.93ms, mfu 2.00%
iter 2670: loss 1.5058, time 182.43ms, mfu 2.00%
iter 2680: loss 1.5499, time 182.82ms, mfu 2.01%
iter 2690: loss 1.4656, time 182.27ms, mfu 2.01%
iter 2700: loss 1.5364, time 182.28ms, mfu 2.01%
iter 2710: loss 1.5719, time 183.83ms, mfu 2.02%
iter 2720: loss 1.5615, time 182.93ms, mfu 2.02%
iter 2730: loss 1.6170, time 182.55ms, mfu 2.02%
iter 2740: loss 1.5317, time 183.17ms, mfu 2.02%
step 2750: train loss 1.4271, val loss 1.4258
saving checkpoint to out-enron-email
iter 2750: loss 1.5046, time 26442.26ms, mfu 1.82%
iter 2760: loss 1.5223, time 182.62ms, mfu 1.84%
iter 2770: loss 1.5584, time 184.14ms, mfu 1.86%
iter 2780: loss 1.5172, time 183.91ms, mfu 1.88%
iter 2790: loss 1.5634, time 182.73ms, mfu 1.90%
iter 2800: loss 1.5066, time 183.70ms, mfu 1.91%
iter 2810: loss 1.5821, time 182.78ms, mfu 1.92%
iter 2820: loss 1.4909, time 182.63ms, mfu 1.93%
iter 2830: loss 1.4311, time 183.54ms, mfu 1.94%
iter 2840: loss 1.4528, time 182.69ms, mfu 1.95%
iter 2850: loss 1.5036, time 183.07ms, mfu 1.96%
iter 2860: loss 1.5008, time 182.49ms, mfu 1.97%
iter 2870: loss 1.4744, time 183.45ms, mfu 1.98%
iter 2880: loss 1.4445, time 183.56ms, mfu 1.98%
iter 2890: loss 1.5162, time 182.55ms, mfu 1.99%
iter 2900: loss 1.4436, time 183.32ms, mfu 1.99%
iter 2910: loss 1.4622, time 183.02ms, mfu 2.00%
iter 2920: loss 1.5705, time 184.82ms, mfu 2.00%
iter 2930: loss 1.4376, time 182.89ms, mfu 2.00%
iter 2940: loss 1.4881, time 183.16ms, mfu 2.01%
iter 2950: loss 1.4437, time 182.94ms, mfu 2.01%
iter 2960: loss 1.6143, time 182.96ms, mfu 2.01%
iter 2970: loss 1.4429, time 182.58ms, mfu 2.02%
iter 2980: loss 1.4679, time 184.76ms, mfu 2.02%
iter 2990: loss 1.5671, time 188.54ms, mfu 2.01%
step 3000: train loss 1.4131, val loss 1.4127
saving checkpoint to out-enron-email
iter 3000: loss 1.5668, time 26466.10ms, mfu 1.81%
iter 3010: loss 1.5993, time 183.46ms, mfu 1.84%
iter 3020: loss 1.5505, time 182.83ms, mfu 1.86%
iter 3030: loss 1.4877, time 184.13ms, mfu 1.87%
iter 3040: loss 1.5203, time 183.21ms, mfu 1.89%
iter 3050: loss 1.6269, time 182.33ms, mfu 1.91%
iter 3060: loss 1.5871, time 184.16ms, mfu 1.92%
iter 3070: loss 1.4407, time 183.57ms, mfu 1.93%
iter 3080: loss 1.4588, time 183.34ms, mfu 1.94%
iter 3090: loss 1.4397, time 183.17ms, mfu 1.95%
iter 3100: loss 1.5132, time 183.95ms, mfu 1.96%
iter 3110: loss 1.4852, time 182.81ms, mfu 1.97%
iter 3120: loss 1.5137, time 183.60ms, mfu 1.97%
iter 3130: loss 1.3992, time 183.22ms, mfu 1.98%
iter 3140: loss 1.4876, time 182.63ms, mfu 1.99%
iter 3150: loss 1.6640, time 182.79ms, mfu 1.99%
iter 3160: loss 1.6007, time 182.78ms, mfu 2.00%
iter 3170: loss 1.4620, time 182.31ms, mfu 2.00%
iter 3180: loss 1.5903, time 183.65ms, mfu 2.00%
iter 3190: loss 1.5681, time 183.24ms, mfu 2.01%
iter 3200: loss 1.4958, time 182.64ms, mfu 2.01%
iter 3210: loss 1.4343, time 183.41ms, mfu 2.01%
iter 3220: loss 1.5142, time 187.42ms, mfu 2.01%
iter 3230: loss 1.4591, time 182.62ms, mfu 2.01%
iter 3240: loss 1.4640, time 183.06ms, mfu 2.02%
step 3250: train loss 1.3954, val loss 1.3993
saving checkpoint to out-enron-email
iter 3250: loss 1.4768, time 26393.12ms, mfu 1.82%
iter 3260: loss 1.5534, time 182.55ms, mfu 1.84%
iter 3270: loss 1.5564, time 183.65ms, mfu 1.86%
iter 3280: loss 1.4012, time 183.11ms, mfu 1.88%
iter 3290: loss 1.5274, time 183.43ms, mfu 1.89%
iter 3300: loss 1.5627, time 182.92ms, mfu 1.91%
iter 3310: loss 1.4883, time 183.19ms, mfu 1.92%
iter 3320: loss 1.4806, time 184.11ms, mfu 1.93%
iter 3330: loss 1.4670, time 182.85ms, mfu 1.94%
iter 3340: loss 1.5604, time 184.54ms, mfu 1.95%
iter 3350: loss 1.4585, time 182.93ms, mfu 1.96%
iter 3360: loss 1.5920, time 182.75ms, mfu 1.97%
iter 3370: loss 1.4993, time 183.68ms, mfu 1.97%
iter 3380: loss 1.4877, time 183.99ms, mfu 1.98%
iter 3390: loss 1.4622, time 182.96ms, mfu 1.99%
iter 3400: loss 1.5224, time 182.89ms, mfu 1.99%
iter 3410: loss 1.5438, time 183.79ms, mfu 2.00%
iter 3420: loss 1.5028, time 182.80ms, mfu 2.00%
iter 3430: loss 1.4778, time 183.86ms, mfu 2.00%
iter 3440: loss 1.4400, time 183.39ms, mfu 2.01%
iter 3450: loss 1.4524, time 187.11ms, mfu 2.00%
iter 3460: loss 1.5198, time 182.81ms, mfu 2.01%
iter 3470: loss 1.5493, time 183.08ms, mfu 2.01%
iter 3480: loss 1.4516, time 182.54ms, mfu 2.01%
iter 3490: loss 1.3329, time 186.01ms, mfu 2.01%
step 3500: train loss 1.3841, val loss 1.3877
saving checkpoint to out-enron-email
iter 3500: loss 1.4443, time 26406.98ms, mfu 1.81%
iter 3510: loss 1.5394, time 183.74ms, mfu 1.84%
iter 3520: loss 1.5174, time 183.57ms, mfu 1.86%
iter 3530: loss 1.4835, time 183.56ms, mfu 1.87%
iter 3540: loss 1.5043, time 184.18ms, mfu 1.89%
iter 3550: loss 1.4821, time 182.47ms, mfu 1.90%
iter 3560: loss 1.4261, time 184.27ms, mfu 1.92%
iter 3570: loss 1.5151, time 184.90ms, mfu 1.93%
iter 3580: loss 1.4698, time 182.31ms, mfu 1.94%
iter 3590: loss 1.4824, time 182.76ms, mfu 1.95%
iter 3600: loss 1.3849, time 182.94ms, mfu 1.96%
iter 3610: loss 1.5252, time 183.54ms, mfu 1.97%
iter 3620: loss 1.5062, time 183.12ms, mfu 1.97%
iter 3630: loss 1.4615, time 184.26ms, mfu 1.98%
iter 3640: loss 1.3672, time 182.61ms, mfu 1.98%
iter 3650: loss 1.4805, time 182.55ms, mfu 1.99%
iter 3660: loss 1.4737, time 182.40ms, mfu 2.00%
iter 3670: loss 1.3931, time 183.88ms, mfu 2.00%
iter 3680: loss 1.4379, time 183.14ms, mfu 2.00%
iter 3690: loss 1.4191, time 183.05ms, mfu 2.01%
iter 3700: loss 1.5552, time 182.61ms, mfu 2.01%
iter 3710: loss 1.4056, time 182.99ms, mfu 2.01%
iter 3720: loss 1.4432, time 182.84ms, mfu 2.02%
iter 3730: loss 1.5132, time 182.67ms, mfu 2.02%
iter 3740: loss 1.4532, time 183.45ms, mfu 2.02%
step 3750: train loss 1.3675, val loss 1.3784
saving checkpoint to out-enron-email
iter 3750: loss 1.5050, time 26490.66ms, mfu 1.82%
iter 3760: loss 1.5369, time 184.20ms, mfu 1.84%
iter 3770: loss 1.5210, time 182.98ms, mfu 1.86%
iter 3780: loss 1.4495, time 184.09ms, mfu 1.88%
iter 3790: loss 1.4524, time 188.97ms, mfu 1.89%
iter 3800: loss 1.4457, time 182.82ms, mfu 1.90%
iter 3810: loss 1.4373, time 184.29ms, mfu 1.91%
iter 3820: loss 1.3992, time 183.83ms, mfu 1.93%
iter 3830: loss 1.4736, time 184.16ms, mfu 1.94%
iter 3840: loss 1.5272, time 183.09ms, mfu 1.95%
iter 3850: loss 1.4472, time 182.65ms, mfu 1.96%
iter 3860: loss 1.5664, time 184.17ms, mfu 1.96%
iter 3870: loss 1.4858, time 183.12ms, mfu 1.97%
iter 3880: loss 1.4569, time 182.70ms, mfu 1.98%
iter 3890: loss 1.4090, time 184.31ms, mfu 1.98%
iter 3900: loss 1.4289, time 183.79ms, mfu 1.99%
iter 3910: loss 1.3996, time 183.90ms, mfu 1.99%
iter 3920: loss 1.3975, time 182.65ms, mfu 2.00%
iter 3930: loss 1.3906, time 183.38ms, mfu 2.00%
iter 3940: loss 1.4453, time 183.33ms, mfu 2.00%
iter 3950: loss 1.5796, time 184.35ms, mfu 2.01%
iter 3960: loss 1.4718, time 183.69ms, mfu 2.01%
iter 3970: loss 1.5272, time 182.58ms, mfu 2.01%
iter 3980: loss 1.5050, time 182.74ms, mfu 2.02%
iter 3990: loss 1.4916, time 183.27ms, mfu 2.02%
step 4000: train loss 1.3609, val loss 1.3740
saving checkpoint to out-enron-email
iter 4000: loss 1.4138, time 26466.85ms, mfu 1.82%
iter 4010: loss 1.3702, time 182.62ms, mfu 1.84%
iter 4020: loss 1.4293, time 184.22ms, mfu 1.86%
iter 4030: loss 1.3496, time 182.88ms, mfu 1.88%
iter 4040: loss 1.4465, time 184.08ms, mfu 1.89%
iter 4050: loss 1.4621, time 183.53ms, mfu 1.91%
iter 4060: loss 1.4274, time 182.68ms, mfu 1.92%
iter 4070: loss 1.4313, time 183.02ms, mfu 1.93%
iter 4080: loss 1.4664, time 183.70ms, mfu 1.94%
iter 4090: loss 1.3556, time 183.75ms, mfu 1.95%
iter 4100: loss 1.5216, time 183.90ms, mfu 1.96%
iter 4110: loss 1.4190, time 182.70ms, mfu 1.97%
iter 4120: loss 1.4884, time 182.87ms, mfu 1.97%
iter 4130: loss 1.4260, time 183.67ms, mfu 1.98%
iter 4140: loss 1.4665, time 182.81ms, mfu 1.99%
iter 4150: loss 1.3607, time 183.14ms, mfu 1.99%
iter 4160: loss 1.3768, time 183.74ms, mfu 2.00%
iter 4170: loss 1.4173, time 183.86ms, mfu 2.00%
iter 4180: loss 1.4822, time 182.35ms, mfu 2.00%
iter 4190: loss 1.4871, time 183.23ms, mfu 2.01%
iter 4200: loss 1.4683, time 182.39ms, mfu 2.01%
iter 4210: loss 1.3882, time 182.87ms, mfu 2.01%
iter 4220: loss 1.4950, time 184.19ms, mfu 2.02%
iter 4230: loss 1.4283, time 183.24ms, mfu 2.02%
iter 4240: loss 1.5603, time 183.27ms, mfu 2.02%
step 4250: train loss 1.3533, val loss 1.3598
saving checkpoint to out-enron-email
iter 4250: loss 1.4462, time 26464.86ms, mfu 1.82%
iter 4260: loss 1.4600, time 182.29ms, mfu 1.84%
iter 4270: loss 1.4416, time 183.01ms, mfu 1.86%
iter 4280: loss 1.4493, time 183.92ms, mfu 1.88%
iter 4290: loss 1.4018, time 182.57ms, mfu 1.89%
iter 4300: loss 1.4572, time 183.22ms, mfu 1.91%
iter 4310: loss 1.4629, time 183.96ms, mfu 1.92%
iter 4320: loss 1.4578, time 182.93ms, mfu 1.93%
iter 4330: loss 1.3955, time 183.51ms, mfu 1.94%
iter 4340: loss 1.4183, time 183.70ms, mfu 1.95%
iter 4350: loss 1.3944, time 183.32ms, mfu 1.96%
iter 4360: loss 1.4668, time 183.88ms, mfu 1.97%
iter 4370: loss 1.4718, time 184.75ms, mfu 1.97%
iter 4380: loss 1.4826, time 184.15ms, mfu 1.98%
iter 4390: loss 1.3947, time 184.13ms, mfu 1.98%
iter 4400: loss 1.4064, time 183.18ms, mfu 1.99%
iter 4410: loss 1.5385, time 184.21ms, mfu 1.99%
iter 4420: loss 1.4361, time 182.41ms, mfu 2.00%
iter 4430: loss 1.3620, time 182.53ms, mfu 2.00%
iter 4440: loss 1.5167, time 183.18ms, mfu 2.01%
iter 4450: loss 1.3869, time 183.20ms, mfu 2.01%
iter 4460: loss 1.4137, time 182.99ms, mfu 2.01%
iter 4470: loss 1.4473, time 183.40ms, mfu 2.01%
iter 4480: loss 1.3839, time 182.95ms, mfu 2.02%
iter 4490: loss 1.4121, time 184.57ms, mfu 2.02%
step 4500: train loss 1.3475, val loss 1.3534
saving checkpoint to out-enron-email
iter 4500: loss 1.4588, time 26362.13ms, mfu 1.82%
iter 4510: loss 1.4067, time 184.11ms, mfu 1.84%
iter 4520: loss 1.4125, time 182.13ms, mfu 1.86%
iter 4530: loss 1.4721, time 182.70ms, mfu 1.88%
iter 4540: loss 1.4111, time 183.60ms, mfu 1.89%
iter 4550: loss 1.4643, time 183.48ms, mfu 1.91%
iter 4560: loss 1.2722, time 182.89ms, mfu 1.92%
iter 4570: loss 1.4372, time 182.58ms, mfu 1.93%
iter 4580: loss 1.4202, time 182.86ms, mfu 1.94%
iter 4590: loss 1.4740, time 180.76ms, mfu 1.96%
iter 4600: loss 1.4670, time 180.79ms, mfu 1.97%
iter 4610: loss 1.4657, time 180.95ms, mfu 1.98%
iter 4620: loss 1.4892, time 180.52ms, mfu 1.99%
iter 4630: loss 1.4087, time 180.52ms, mfu 1.99%
iter 4640: loss 1.3623, time 181.43ms, mfu 2.00%
iter 4650: loss 1.4501, time 182.71ms, mfu 2.00%
iter 4660: loss 1.5414, time 183.47ms, mfu 2.01%
iter 4670: loss 1.4254, time 182.33ms, mfu 2.01%
iter 4680: loss 1.4481, time 181.63ms, mfu 2.02%
iter 4690: loss 1.4169, time 182.73ms, mfu 2.02%
iter 4700: loss 1.4764, time 181.28ms, mfu 2.02%
iter 4710: loss 1.4463, time 180.72ms, mfu 2.03%
iter 4720: loss 1.3424, time 180.57ms, mfu 2.03%
iter 4730: loss 1.3770, time 180.52ms, mfu 2.03%
iter 4740: loss 1.4124, time 180.72ms, mfu 2.04%
step 4750: train loss 1.3393, val loss 1.3540
iter 4750: loss 1.4003, time 26165.09ms, mfu 1.84%
iter 4760: loss 1.4782, time 182.76ms, mfu 1.86%
iter 4770: loss 1.4566, time 183.14ms, mfu 1.87%
iter 4780: loss 1.4193, time 184.72ms, mfu 1.89%
iter 4790: loss 1.3714, time 183.48ms, mfu 1.90%
iter 4800: loss 1.4879, time 183.03ms, mfu 1.92%
iter 4810: loss 1.3938, time 182.47ms, mfu 1.93%
iter 4820: loss 1.4639, time 182.80ms, mfu 1.94%
iter 4830: loss 1.4082, time 182.65ms, mfu 1.95%
iter 4840: loss 1.3797, time 182.87ms, mfu 1.96%
iter 4850: loss 1.3520, time 182.28ms, mfu 1.97%
iter 4860: loss 1.4986, time 182.70ms, mfu 1.98%
iter 4870: loss 1.3730, time 183.28ms, mfu 1.98%
iter 4880: loss 1.5202, time 182.66ms, mfu 1.99%
iter 4890: loss 1.4498, time 184.15ms, mfu 1.99%
iter 4900: loss 1.4759, time 182.67ms, mfu 2.00%
iter 4910: loss 1.4809, time 183.87ms, mfu 2.00%
iter 4920: loss 1.4513, time 182.41ms, mfu 2.01%
iter 4930: loss 1.4162, time 183.76ms, mfu 2.01%
iter 4940: loss 1.4123, time 184.46ms, mfu 2.01%
iter 4950: loss 1.3977, time 182.83ms, mfu 2.01%
iter 4960: loss 1.4300, time 182.48ms, mfu 2.02%
iter 4970: loss 1.4665, time 182.85ms, mfu 2.02%
iter 4980: loss 1.4228, time 182.87ms, mfu 2.02%
iter 4990: loss 1.4732, time 182.67ms, mfu 2.02%
```

```python
# Evaluate the model
eval_model(model=trained_model)
```

```
Loading meta from meta.pkl...
Sample 1:

 Water on August 2001, 2001 11:52 AM
To: Kolli Kewitz/ENRON@enronXgate Communications@ENRON
cc: Vicki Heas/HOU/ECT@ECT
cc: Donna Blank/HOU/ECT@ECT, Robert Schulder/Corp/Enron@Enron
cc:
Subject: Re: FW: November 1. Should receive









---Original Message----
From:      Dasovich, Mara
Sent: Monday, October 03, 2001 12:34 PM
To:   White, James D Subject:



Do we did not any help we please do not contact which may be my
sistance the
contact of personal companies as the recent of the financial
energy companies that these state's just as of a current
rebacked in Partner.  I am short the price of the explanation of the
```

```
address which
is the believe charge the call host
provide by the cost of the website. The state company of the state
will
need to expect to be that the U.S. company of a SoCal factors.  This
is go toget
continued by the service minuter, and bankruptcy manager products will
seek to the brought
many message a released its optimate allow this in utility into a
distributio
----------------------------------------------------------------------
----------
Sample 2:
```

```
Price a very contract because reported to bring you can receive the
commondate to decide.  However,
and its preveniency set up the Sidea with a file price left it with
the
contract different that can be advised to made a preferrence wanted to
estable
second such as a presentation of earlier than the U.S. will be
preferenced of lease lawyers that the assumer of the disagreement
previous engineeration.
In the RD projects to other statements to ensure their plant and any
qualifications, which
is repeable to the acquisition.
```

[IMAGE]

Mark Times
10/04/2000 09:21 PM
To: Kim Shapiro/NA/Enron@Enron
cc:
Subject: Protection Contract Password for the absource of other
entered and Crisis

George Raters

Jeff

Regards,

Jeff

Mark ------------------------ Forwarded by Tim Wilton/HOU/ECT on
11/25/2000 02:31 AM ------------------------

Please do this e-mail who confident is a serve of Indianance for

```
Christmas
Tracisco of the CW project.
----------------------------------------------------------------------
----------
Sample 3:


                      <R
FACE="http://www.rigzone.com/images/email/aspacer.gif>
<http://www.rigzone.com/images/spacer.gif>
<http://www.investments.com/images/spacer.gif>
<http://www.rigzone.com/images/spacer.gif>
<http://www.economister.com/images/spacer.gif>
<http://www.rigzone.com/images/spacer.gif>
<http://www.energy.com/images/spacer.gif>
<http://www.rigzone.com/images/spacer.gif>
<http://www.net/images/spacer.gif>
    <http://www.custom.com/images/spacer.gif>
   <http://www.ciuise.com/images/images/spacer.gif>
   <http://www.rigzone.com/images/link.adp?id=275></td>
            <td color="#FFFFFFF" face="Verdana, arial, Helvetica"
size="2" size="1">


----------------------------------------------------------------------
----------
Sample 4:



PRIVILEGRATION TO INTERNATE RIVE SHAREs ON THE favor to report on the
state delayers for "To continue the box consultants" and management as
set of our states, and is necessary provided by Chicago Commission
Secretary at PG&E considerators, residential increase by charges of
the months.  The program was parties to problem.  The state are
expected to sean of the state Enron's project account in a dot of the
Contract from Enron Democrats Industry, included its information of
Corp.  The points would be contracted to associate for the London of
Enron and Article responding due on the energy restructions.  The
American Edison is the "high-a following pipeline state last week.
This is energy hours to deal for the PPA's trading group with November
27 of the security by Ohson and other Texas.  The contact may be able
the sender information into oil version of commissioners  . . . . .
Gas Davis Brown's plant successful from
----------------------------------------------------------------------
----------
Sample 5:
```

```
----------------------------------------------------------------------
----------
Sample 6:


Power that we wanted to reply solution who were thought the Meeting
data was and
worder to fall, that we can can change. Sorry state he will be a
regulatory of password, the NFRC congression of the process on
designed in the bonds
instruction.''  It rather the state's allegal costs that the
state's utility's includes as they driver be standing was swell
because
know that the found partment shall in state before a profession
suppliers but they are residentially
proposed by the approval to be complitants to California's partnership
as
consistential law companies that CEO try Venture of big service as the

state energy government and delivery said its and providers on the
risk interparterprehensive
believe high power of cross and competiting sprices of the
information technology. The responsibility database and is the reason
round plants of our statement costs and gas-defenses from the Finance
to
receive the public power credit. A week of additional deskgreement on
the
state
----------------------------------------------------------------------
----------
Sample 7:


Even Pacific Company, Markets has also known complete to be procedured
the
monthst and response that shipping from the interest
sender to be the University and Attached the Corp. and the FERC
any other expected to the language presentation, but they would like
to be a bid
likely with the company to send that there are to expect their
employmental.

Thanks!

Please let me know for your havings could be confidential disclosure
or for month.

Have send is on the Stansfer and let me know your or options
```

```
about I will have a complete to found complete for the group,
please click here.

Confidential information Island Generational and the approval delivery
will review of its
state for other specifical prices with analyzed in the
Independent agreements to Sempra Enron Commission and SPC is out
available to a
some assets for DOP Capital Reserved.
If it is a final confirmance is in Sunday.  But it is not raised how
if works that the short has been he hearing the Transaction Commission



----------------------------------------------------------------------
----------
Sample 8:

Scott Americas Client Technology Coopertions.
Today's uninterest from $146.4 per of $0.66

With has fact Electricity for the Commission Board.  The Neutro
Company Siervised in Five
Resources Form Power Communications Shop Power Democrats will be found
to be the
rate desktation of the United Board Lavo shares of 17 megawatts.  So
far a staff cold-term
wholesale comments to sign the Electricity Management and would likely
on the
Restance Agency says volunteered are limitted activities.

Thanks,
Kate







 ----Original Message----
From:  bark T.
Sent: Tuesday, October 12, 2001 6:43 AM
To: Robert A.
Subject:



I have trying out of your gas assumer.

Hi Gary are now that is not specific this week.  I don't all all the
give the submittee of the
```

```
3-7 contract is someone who that you
are propertion to effort and this subscribe or that I don't set me
know that wanted
making to deliver the subscribe to do you prepare are significant-for
that
their projects reserve - they offer PG&E
----------------------------------------------------------------------------
----------
Sample 9:




Janes D. Sallers February 00 12:27 PM
To: skitt@enron.com
cc: James J Mara/NA/Enron@Enron Communications, James D
Steffes/HOU/EES@EES, Steven
Mara@ENRON, Maria London/Corp/Enron@ENRON, Michael
Campbe/ET&S/Enron@Enron, David McGroups/ENRON@enronXgate, Dan
Darent/FGT/Enron@ENRON, Mary Kavie
Harnes/Enron@ENRON, Jeff Dasovich/NA/Enron@Enron, Paul
Greg Stock/HOU/EES@EES, Kate Symes/PDX/ECT@ECT, Susan J
Mara/NA/Enron@ENRON, Michael
Jim Ellio/NA/Enron@Enron, Larry K Taylor/HOU/EES@EES, Mark E
Shapiro/NA/Enron@Enron, Jeff Dasovich/NA/Enron@Enron, Phillip

Comnes/NA/Enron@Enron, James D Steffes/NA/Enron@Enron, Richarley
Thomas D (E-mail)'" <rward@enronXgate cc:
Subject: Fw: Web  Drivery









Thanks,

JUST



====================================================================
O Mike Swert Systems -- For Houston
<http://www.genergyenergy.com/pma/dpal?
t=id=00000020181&ND=109572840226;

I have a set three agencies you have any quarter of disclosure, and
the stanting you fiel
----------------------------------------------------------------------
----------
Sample 10:


-------------- Forwarded by Moy Corp/Enron on 12/08/2001
11:52 AM -------------------

     Energy Sent:    Thursday, August 15, 2001 12:30 AM
To:  Candy August (E-mail)'; Williams, Christi Bob Sent:   Wednesday,
Information
     03/17/2001 01:44 AM
To: David Mean/HOU/ECT@ECT, Tana Gasse/HOU/ECT@ECT, Mary
Kenne/NA/Enron@Enron, Jeff Dasovich/NA/Enron@Enron, Mary
Bartine/Corp/Enron@ENRON, Jeff Dasovich/NA/Enron@Enron, Janet
Martinet/ENRON@enronXgate, Lee Hass/HOU/ECT@ECT
cc:
Subject: Re:


Daren't find deal total offsets of the 50rd PR at the RC points


======================================================================
===========================================================
The message of the restory for AMEDIA friends on the Profile law
information on the End to all
the company's become majority and the transference
informations could be take to send the until termination.




----------------------------------------------------------------------
----------
Sample 11:


     Enron Communications in the fundate of companies application
management in number the link parties for the having today to posciate
nomination on its section in the EnronOnline.  In the event.
```

```
Later 2-088-975-1700

Enron's high-company product to San John Enron (E-mail)'" <sroftwarn
continue="verdana, helvetica" said.  He was looking a conference
earlier the statement to imply management provider of the Thursday and
OG and their
company price since its out to the power plant system.  So very
support to much open the energy transaction
```

```
--------- Forwarded by Mary Christers/Corp/Enron on 02/2000 04:16 PM
-----------------------




"Daren H Newski, Davis <richard@enron.com>
     02/22/2000 02:28 PM
           To: James Porter/HOU/ECT@ECT, Jeff
Dasovich/NA/Enron@Enron, Tom
Announce/HOU/ECT@ECT, Mark McCubbine/HOU/ECT@ECT, James D
George/HOU/ECT@ECT, Racy Paul Paul/HOU/ECT@ECT, Andrew
Board/ET&S/Enron@ENRON, Stephen J
Michael/HOU/ECT@ECT, Paul K Smith/ENRON@enronXgate, Robert
----------------------------------------------------------------------------
----------
```