Pre-Train a GPT from Scratch to be an ExpertGPT using PyTorch

Step 1: Import libraries and setup constants

In [1]:
```
import numpy as np
import tiktoken
import torch
import torch.nn as nn
from torch.nn import functional as F
```

Step 2: Load data

I have scraped a bunch of recipes from allrecipes.com and saved them in a text file.

In [2]:
```
input_file = r"data\allrecipes_data.txt"

with open(input_file, 'r', encoding='utf-8') as f:
    text = f.readlines()

print("length of data in characters:", sum(len(line) for line in text))
```

```
length of data in characters: 2038449
```

Step 3: Setup vocab and encode/decode functions

In [3]:
```
def setup_vocab_size(text, tiktoken=False):
        """Setup vocabulary size based on encoding method.
        Args:
                text (list of str): List of text lines.
                tiktoken (bool): Whether to use tiktoken for vocabulary.
        Returns:
                int: Vocabulary size.
        """
        if tiktoken:
                vocab_size = ENCODER.n_vocab

                print("vocab size based on tiktoken GPT-2 encoding:", vocab_size
        else:
                the_chars = sorted(list(set(" ".join(text))))
                vocab_size = len(the_chars)

                print("vocab size based on unique characters:", vocab_size)
                print('chars:', ''.join(the_chars))

        return vocab_size
```

In [4]:
```
def encode(text, main_text=None, max_len=128, tiktoken=False):
        """Encode text to a sequence of integers.
        Args:
                text (list of str): List of text lines to encode.
                max_len (int): Maximum length of each encoded line.
```

```
                tiktoken (bool): Whether to use tiktoken for encoding.
        Returns:
                torch.Tensor: Tensor of encoded integers.
        """
        if tiktoken:
                all_tokens = []
                for line in text:
                        tokens = ENCODER.encode(line.strip())
                        # truncate long recipes
                        tokens = tokens[:max_len]
                        # add separator between recipes
                        all_tokens.extend(tokens + [ENCODER.eot_token])
        else:
                all_chars = sorted(list(set(" ".join(main_text))))
                stoi = {ch: i for i, ch in enumerate(all_chars)}

                if text is None:
                        text = main_text

                all_tokens = []
                for line in text:
                        tokens = [stoi[ch] for ch in line.strip()]
                        # truncate long recipes
                        tokens = tokens[:max_len]
                        # add separator between recipes
                        all_tokens.extend(tokens + [stoi[' ']])

        return torch.tensor(all_tokens, dtype=torch.long)


def decode(tokens, text=None, tiktoken=False):
        """Decode a sequence of integers back to text.
        Args:
                tokens (list or torch.Tensor): Sequence of integers to decode.
                text (str): Original text (required if not using tiktoken).
                tiktoken (bool): Whether to use tiktoken for decoding.
        Returns:
                str: Decoded text.
        """
        if tiktoken:
                text = ENCODER.decode(tokens)
                text = text.replace('<|endoftext|>', '\n')
                return text
        else:
                all_chars = sorted(list(set(" ".join(text))))
                itos = {i: ch for i, ch in enumerate(all_chars)}
                text = ''.join([itos[token] for token in tokens])
                return text.replace(' ', '\n')
```

Step 4: Split data into train and validation sets

In [5]:

```
def get_batch(data, train_ratio=0.9, train=True):
    n = int(train_ratio * len(data))
    train_data = data[:n]
    val_data = data[n:]

    if train:
        print(f"Train data has {len(train_data)} tokens")
```

```
        data_ = train_data
    else:
        print(f"Validation data has {len(val_data)} tokens")
        data_ = val_data

    max_start = len(data_) - BLOCK_SIZE
    if max_start <= 0:
        raise ValueError(
            f"Data too small for block_size={BLOCK_SIZE}. Reduce block_size.")

    ix = torch.randint(max_start, (BATCH_SIZE,))
    x = torch.stack([data_[i: i+BLOCK_SIZE] for i in ix])
    y = torch.stack([data_[i+1: i+1+BLOCK_SIZE] for i in ix])

    x, y = x.to(DEVICE), y.to(DEVICE)
    return x, y
```

Step 5: Create NN Architecture

In [6]:
```
class Head(nn.Module):
    def __init__(self, head_size):
        super().__init__()
        self.key = nn.Linear(N_EMBED, head_size, bias=False)
        self.query = nn.Linear(N_EMBED, head_size, bias=False)
        self.value = nn.Linear(N_EMBED, head_size, bias=False)

        tril_def = torch.tril(torch.ones(BLOCK_SIZE, BLOCK_SIZE))
        self.register_buffer('tril', tril_def)
        self.dropout = nn.Dropout(DROPOUT_VAL)

    def forward(self, x):
        B, T, E = x.shape
        k = self.key(x)
        q = self.query(x)

        head_size = k.size(-1)
        wei = q @ k.transpose(-2, -1) * head_size ** -0.5
        wei = wei.masked_fill(self.tril[:T, :T] == 0, float('-inf'))
        wei = F.softmax(wei, dim=-1)
        wei = self.dropout(wei)

        v = self.value(x)
        out = wei @ v
        return out
```

In [7]:
```
class FeedForward(nn.Module):
    def __init__(self, n_embd):
        super().__init__()
        self.net = nn.Sequential(
            nn.Linear(n_embd, 4 * n_embd),
            nn.ReLU(),
            nn.Linear(4 * n_embd, n_embd),
            nn.Dropout(DROPOUT_VAL),
        )

    def forward(self, x):
        return self.net(x)
```

In [8]:
```
class MultiHeadAttention(nn.Module):
    def __init__(self, n_embd, num_heads, head_size):
        super().__init__()
        self.heads = nn.ModuleList([Head(head_size) for _ in range(num_heads)])
        self.proj = nn.Linear(n_embd, n_embd)
        self.dropout = nn.Dropout(DROPOUT_VAL)

    def forward(self, x):
        out = torch.cat([h(x) for h in self.heads], dim=-1)
        out = self.proj(out)
        out = self.dropout(out)
        return out
```

In [9]:
```
class Block(nn.Module):
    def __init__(self, n_embd, n_head):
        super().__init__()
        head_size = n_embd // n_head
        self.sa = MultiHeadAttention(n_embd, n_head, head_size)
        self.ffwd = FeedForward(n_embd)
        self.ln1 = nn.LayerNorm(n_embd)
        self.ln2 = nn.LayerNorm(n_embd)

    def forward(self, x):
        x = x + self.sa(self.ln1(x))
        x = x + self.ffwd(self.ln2(x))
        return x
```

In [10]:
```
class GPTModel(nn.Module):
    def __init__(self, vocab_size, n_embed, n_head, n_layer):
        super().__init__()
        self.token_embedding_table = nn.Embedding(vocab_size, n_embed)
        self.pos_emb_table = nn.Embedding(BLOCK_SIZE, n_embed)

        self.blocks = nn.Sequential(
            *[Block(n_embed, n_head=n_head) for _ in range(n_layer)]
        )

        self.ln_f = nn.LayerNorm(n_embed)
        self.lm_ffw_head = nn.Linear(n_embed, vocab_size)

    def forward(self, idx, targets=None):
        B, T = idx.shape
        tok_emb = self.token_embedding_table(idx)
        pos_emb = self.pos_emb_table(torch.arange(T, device=DEVICE))
        x = tok_emb + pos_emb
        x = self.blocks(x)
        x = self.ln_f(x)
        logits = self.lm_ffw_head(x)

        if targets is None:
            loss = None
        else:
            B, T, E = logits.shape
            logits = logits.view(B*T, E)
            targets = targets.view(B*T)
            loss = F.cross_entropy(logits, targets)
        return logits, loss

    def generate(self, idx, max_new_tokens):
```

```
        for _ in range(max_new_tokens):
            idx_cond = idx[:, -BLOCK_SIZE:]
            logits, loss = self(idx_cond)
            logits = logits[:, -1, :]
            probs = F.softmax(logits, dim=-1)
            idx_next = torch.multinomial(probs, num_samples=1)
            idx = torch.cat((idx, idx_next), dim=1)
        return idx
```

Step 6: Setup loss function

In [11]:
```
@torch.no_grad()
def estimate_loss(train_data, val_data):
    out = {}
    model.eval()
    for split in ['train', 'val']:
        losses = torch.zeros(EVAL_ITERS)
        for k in range(EVAL_ITERS):
            if split == 'train':
                X, Y = train_data
            else:
                X, Y = val_data
            logits, loss = model(X, Y)
            losses[k] = loss.item()
        out[split] = losses.mean()
    model.train()
    return out
```

Step 7: Setup initial params

In [12]:
```
torch.manual_seed(256)
DEVICE = 'cuda' if torch.cuda.is_available() else 'cpu'
print("Using device:", DEVICE)

BLOCK_SIZE = 40  # N tokens in sequence
BATCH_SIZE = 64
MAX_ITERS = 6000
EVAL_ITERS = 300
EVAL_INTERVAL = 500

LEARNING_RATE = 0.0003
N_EMBED = 512
N_HEAD = 8  # 8 attention heads
N_LAYER = 6  # 6 encoder layers
DROPOUT_VAL = 0.2

# other constants
ENCODER = tiktoken.get_encoding("gpt2")
```

```
Using device: cuda
```

Step 8: Train without tiktoken

In [13]:
```
VOCAB_SIZE = setup_vocab_size(text, tiktoken=False)
data = encode(text, main_text=text, tiktoken=False)
```

```
print("\nEncoded data:", data)
print("Encoded data shape: {} and dtype: {}".format(data.shape, data.dtype))
```

```
vocab size based on unique characters: 110
chars:
 !"$%&'()*,-./0123456789:;=?ABCDEFGHIJKLMNOPQRSTUVWXYZ[]abcdefghijklmnopqrstuvwxy
z ®°¼½¾áèéñûĆćč––‘’“”/™⅓⅔⅛

Encoded data: tensor([48, 65, 76,  ..., 27,  1,  1])
Encoded data shape: torch.Size([1559067]) and dtype: torch.int64
```

In [14]:
```
train_data = get_batch(data, train=True)
print("\nTrain data X shape:", train_data[0].shape)
print("Train data Y shape:", train_data[1].shape)

val_data = get_batch(data, train=False)
print("\nValidation data X shape:", val_data[0].shape)
print("Validation data Y shape:", val_data[1].shape)
```

```
Train data has 1403160 tokens

Train data X shape: torch.Size([64, 40])
Train data Y shape: torch.Size([64, 40])
Validation data has 155907 tokens

Validation data X shape: torch.Size([64, 40])
Validation data Y shape: torch.Size([64, 40])
```

In [15]:
```
model = GPTModel(
        vocab_size=VOCAB_SIZE,
        n_embed=N_EMBED,
        n_head=N_HEAD,
        n_layer=N_LAYER,
).to(DEVICE)
optimizer = torch.optim.Adam(model.parameters(), lr=LEARNING_RATE)
```

In [16]:
```
for iter in range(MAX_ITERS):
    if iter % EVAL_INTERVAL == 0:
        losses = estimate_loss(train_data, val_data)
        print(
            f"step {iter}: train loss {losses['train']:.4f}, val loss {losses['v

    xb, yb = train_data

    logits, loss = model(xb, yb)
    optimizer.zero_grad(set_to_none=True)
    loss.backward()
    optimizer.step()
```

```
step 0: train loss 4.8732, val loss 4.8785
step 500: train loss 0.0344, val loss 6.7613
step 1000: train loss 0.0340, val loss 7.0988
step 1500: train loss 0.0334, val loss 7.4959
step 2000: train loss 0.0334, val loss 7.5826
step 2500: train loss 0.0336, val loss 7.6140
step 3000: train loss 0.0334, val loss 7.9874
step 3500: train loss 0.0334, val loss 7.7455
step 4000: train loss 0.0332, val loss 7.9353
step 4500: train loss 0.0332, val loss 7.9278
step 5000: train loss 0.0335, val loss 7.8881
step 5500: train loss 0.0331, val loss 8.4531
```

In [17]:
```
# Save the trained model
torch.save(model.state_dict(), "recipe_gpt_weights.pt")
torch.save(model, "recipe_gpt_full.pt")
```

Step 9: Train with tiktoken

In [18]:
```
VOCAB_SIZE = setup_vocab_size(text, tiktoken=True)
data = encode(text, tiktoken=True)

print("\nEncoded data:", data)
print("Encoded data shape: {} and dtype: {}".format(data.shape, data.dtype))
```

```
vocab size based on tiktoken GPT-2 encoding: 50257

Encoded data: tensor([19160,    25,  1879,  ...,  2559, 50256, 50256])
Encoded data shape: torch.Size([554535]) and dtype: torch.int64
```

In [19]:
```
train_data = get_batch(data, train=True)
print("\nTrain data X shape:", train_data[0].shape)
print("Train data Y shape:", train_data[1].shape)

val_data = get_batch(data, train=False)
print("\nValidation data X shape:", val_data[0].shape)
print("Validation data Y shape:", val_data[1].shape)
```

```
Train data has 499081 tokens

Train data X shape: torch.Size([64, 40])
Train data Y shape: torch.Size([64, 40])
Validation data has 55454 tokens

Validation data X shape: torch.Size([64, 40])
Validation data Y shape: torch.Size([64, 40])
```

In [20]:
```
model = GPTModel(
        vocab_size=VOCAB_SIZE,
        n_embed=N_EMBED,
        n_head=N_HEAD,
        n_layer=N_LAYER,
).to(DEVICE)
optimizer = torch.optim.Adam(model.parameters(), lr=LEARNING_RATE)
```

In [21]:
```
for iter in range(MAX_ITERS):
    if iter % EVAL_INTERVAL == 0:
        losses = estimate_loss(train_data, val_data)
        print(
```

```
            f"step {iter}: train loss {losses['train']:.4f}, val loss {losses['v

    xb, yb = train_data

    logits, loss = model(xb, yb)
    optimizer.zero_grad(set_to_none=True)
    loss.backward()
    optimizer.step()
```

```
step 0: train loss 10.9975, val loss 10.9803
step 500: train loss 0.0101, val loss 6.5823
step 1000: train loss 0.0095, val loss 6.9171
step 1500: train loss 0.0095, val loss 6.8513
step 2000: train loss 0.0093, val loss 7.1076
step 2500: train loss 0.0093, val loss 7.2503
step 3000: train loss 0.0092, val loss 7.3446
step 3500: train loss 0.0092, val loss 7.4494
step 4000: train loss 0.0092, val loss 7.5193
step 4500: train loss 0.0092, val loss 7.6105
step 5000: train loss 0.0092, val loss 7.6700
step 5500: train loss 0.0092, val loss 7.7287
```

In [22]:
```
# Save the trained model
torch.save(model.state_dict(), "recipe_gpt_weights_tiktoken.pt")
torch.save(model, "recipe_gpt_full_tiktoken.pt")
```

Evaluate saved models

Evaluate model without tiktoken

In [13]:
```
VOCAB_SIZE = setup_vocab_size(text, tiktoken=False)
data = encode(text, main_text=text, tiktoken=False)

print("\nEncoded data:", data)
print("Encoded data shape: {} and dtype: {}".format(data.shape, data.dtype))
```

```
vocab size based on unique characters: 110
chars:
 !"$%&'()*,-./0123456789:;=?ABCDEFGHIJKLMNOPQRSTUVWXYZ[]abcdefghijklmnopqrstuvwxy
z ®°¼½¾áèéñûĆćč––‘’“”/™⅓⅔⅛

Encoded data: tensor([48, 65, 76,  ..., 27,  1,  1])
Encoded data shape: torch.Size([1559067]) and dtype: torch.int64
```

In [14]:
```
model_path = "recipe_gpt_full.pt"
model = torch.load(model_path, weights_only=False, map_location=DEVICE)
model.eval()
```

Out[14]:
```
GPTModel(
  (token_embedding_table): Embedding(110, 512)
  (pos_emb_table): Embedding(40, 512)
  (blocks): Sequential(
    (0): Block(
      (sa): MultiHeadAttention(
        (heads): ModuleList(
          (0-7): 8 x Head(
            (key): Linear(in_features=512, out_features=64, bias=False)
            (query): Linear(in_features=512, out_features=64, bias=False)
            (value): Linear(in_features=512, out_features=64, bias=False)
            (dropout): Dropout(p=0.2, inplace=False)
          )
        )
        (proj): Linear(in_features=512, out_features=512, bias=True)
        (dropout): Dropout(p=0.2, inplace=False)
      )
      (ffwd): FeedForward(
        (net): Sequential(
          (0): Linear(in_features=512, out_features=2048, bias=True)
          (1): ReLU()
          (2): Linear(in_features=2048, out_features=512, bias=True)
          (3): Dropout(p=0.2, inplace=False)
        )
      )
      (ln1): LayerNorm((512,), eps=1e-05, elementwise_affine=True)
      (ln2): LayerNorm((512,), eps=1e-05, elementwise_affine=True)
    )
    (1): Block(
      (sa): MultiHeadAttention(
        (heads): ModuleList(
          (0-7): 8 x Head(
            (key): Linear(in_features=512, out_features=64, bias=False)
            (query): Linear(in_features=512, out_features=64, bias=False)
            (value): Linear(in_features=512, out_features=64, bias=False)
            (dropout): Dropout(p=0.2, inplace=False)
          )
        )
        (proj): Linear(in_features=512, out_features=512, bias=True)
        (dropout): Dropout(p=0.2, inplace=False)
      )
      (ffwd): FeedForward(
        (net): Sequential(
          (0): Linear(in_features=512, out_features=2048, bias=True)
          (1): ReLU()
          (2): Linear(in_features=2048, out_features=512, bias=True)
          (3): Dropout(p=0.2, inplace=False)
        )
      )
      (ln1): LayerNorm((512,), eps=1e-05, elementwise_affine=True)
      (ln2): LayerNorm((512,), eps=1e-05, elementwise_affine=True)
    )
    (2): Block(
      (sa): MultiHeadAttention(
        (heads): ModuleList(
          (0-7): 8 x Head(
            (key): Linear(in_features=512, out_features=64, bias=False)
            (query): Linear(in_features=512, out_features=64, bias=False)
            (value): Linear(in_features=512, out_features=64, bias=False)
            (dropout): Dropout(p=0.2, inplace=False)
```

```
        )
      )
      (proj): Linear(in_features=512, out_features=512, bias=True)
      (dropout): Dropout(p=0.2, inplace=False)
    )
    (ffwd): FeedForward(
      (net): Sequential(
        (0): Linear(in_features=512, out_features=2048, bias=True)
        (1): ReLU()
        (2): Linear(in_features=2048, out_features=512, bias=True)
        (3): Dropout(p=0.2, inplace=False)
      )
    )
    (ln1): LayerNorm((512,), eps=1e-05, elementwise_affine=True)
    (ln2): LayerNorm((512,), eps=1e-05, elementwise_affine=True)
  )
  (3): Block(
    (sa): MultiHeadAttention(
      (heads): ModuleList(
        (0-7): 8 x Head(
          (key): Linear(in_features=512, out_features=64, bias=False)
          (query): Linear(in_features=512, out_features=64, bias=False)
          (value): Linear(in_features=512, out_features=64, bias=False)
          (dropout): Dropout(p=0.2, inplace=False)
        )
      )
      (proj): Linear(in_features=512, out_features=512, bias=True)
      (dropout): Dropout(p=0.2, inplace=False)
    )
    (ffwd): FeedForward(
      (net): Sequential(
        (0): Linear(in_features=512, out_features=2048, bias=True)
        (1): ReLU()
        (2): Linear(in_features=2048, out_features=512, bias=True)
        (3): Dropout(p=0.2, inplace=False)
      )
    )
    (ln1): LayerNorm((512,), eps=1e-05, elementwise_affine=True)
    (ln2): LayerNorm((512,), eps=1e-05, elementwise_affine=True)
  )
  (4): Block(
    (sa): MultiHeadAttention(
      (heads): ModuleList(
        (0-7): 8 x Head(
          (key): Linear(in_features=512, out_features=64, bias=False)
          (query): Linear(in_features=512, out_features=64, bias=False)
          (value): Linear(in_features=512, out_features=64, bias=False)
          (dropout): Dropout(p=0.2, inplace=False)
        )
      )
      (proj): Linear(in_features=512, out_features=512, bias=True)
      (dropout): Dropout(p=0.2, inplace=False)
    )
    (ffwd): FeedForward(
      (net): Sequential(
        (0): Linear(in_features=512, out_features=2048, bias=True)
        (1): ReLU()
        (2): Linear(in_features=2048, out_features=512, bias=True)
        (3): Dropout(p=0.2, inplace=False)
      )
```

```
      )
      (ln1): LayerNorm((512,), eps=1e-05, elementwise_affine=True)
      (ln2): LayerNorm((512,), eps=1e-05, elementwise_affine=True)
    )
    (5): Block(
      (sa): MultiHeadAttention(
        (heads): ModuleList(
          (0-7): 8 x Head(
            (key): Linear(in_features=512, out_features=64, bias=False)
            (query): Linear(in_features=512, out_features=64, bias=False)
            (value): Linear(in_features=512, out_features=64, bias=False)
            (dropout): Dropout(p=0.2, inplace=False)
          )
        )
        (proj): Linear(in_features=512, out_features=512, bias=True)
        (dropout): Dropout(p=0.2, inplace=False)
      )
      (ffwd): FeedForward(
        (net): Sequential(
          (0): Linear(in_features=512, out_features=2048, bias=True)
          (1): ReLU()
          (2): Linear(in_features=2048, out_features=512, bias=True)
          (3): Dropout(p=0.2, inplace=False)
        )
      )
      (ln1): LayerNorm((512,), eps=1e-05, elementwise_affine=True)
      (ln2): LayerNorm((512,), eps=1e-05, elementwise_affine=True)
    )
  )
  (ln_f): LayerNorm((512,), eps=1e-05, elementwise_affine=True)
  (lm_ffw_head): Linear(in_features=512, out_features=110, bias=True)
)
```

In [15]:

```
sos_context = torch.zeros((1, 1), dtype=torch.long, device=DEVICE)
generated_text = model.generate(sos_context, max_new_tokens=500)[0].tolist()
print(decode(generated_text, text=text, tiktoken=False))
```

```
eded
Instructions:
1.
To
make
the
dippin
f
S
S
-
1/4
Stun
a
owit
f
wil
fr

5
1/4
1/4
on
tu
-
1/4
om
Swixtu
1/4
o
-
1/4
1/4
1/4
-
1/4
n
o
1/4
on
25
-
1/25
-
1/4
f-
1/2cuplacooondonary
3/4
f-
1/4
-
1/4
Stlablalablililand
S-
1/25
jucoon
1/2coon
poond-
1/4
St.
```

```
1/5
oooupfr
-
1/25
-
1/4
Tion
S
wblatlalatlablat
ovxtucoved
-
1/4
-
1/4
-
1/4
Stlat
ond
-
1/4
wixtuy
-
1/4
1/4
on
on
p-
3/4
Powixtutur
S
S
on
on
1/4
r
wixtur
slatur
tuc
on
f
-
1/4
1/4
latlablaco==================================================
```

In [16]:
```
new_context = torch.tensor(encode(["Chicken recipe"], main_text=text, tiktoken=F
generated_text = model.generate(new_context, max_new_tokens=500)[0].tolist()
print(decode(generated_text, text=text, tiktoken=False))
```

```
C:\Users\ikath\AppData\Local\Temp\ipykernel_11680\3269294349.py:1: UserWarning: T
o copy construct from a tensor, it is recommended to use sourceTensor.detach().cl
one() or sourceTensor.detach().clone().requires_grad_(True), rather than torch.te
nsor(sourceTensor).
  new_context = torch.tensor(encode(["Chicken recipe"], main_text=text, tiktoken=
False), dtype=torch.long, device=DEVICE).view(1, -1)
```

```
Chicken
recipe
ene
meant
oven
wil
sn
for
-
3
s
r
for
-
1/4
1/4
k
fr
poon
onatila
pat
fr
25
on
win
fo
Swixtur
satutlat
on
ow
satlatur
ove
Stu
1/4
St
fr
5
1/2t.
fr
/2e
Stlat
o
f
-
1/4
1/4
-
1/4
Sm-
ablalat
Swixtur
her
-
1/4
-
1/2con
1/4
1/4
1/4
```

```
1/4
lalablat
ond
-
1.
Sud
1/4
fr
-
1/4
St.
Stu-
1/4
1/2lalion
1/4
S
-
1/4
1/4
-
1/4
1/4
-
1/2ck
on
-
1/4
-
1/4
1/4
on
-
1/4
1/4
1/4
-
1/4
1/4
on
-
1/25
frdd
-
1/4
ju
-
1/4
-
1/4
1/4
1/4
filalalilau-
1/2con
-
1/4
1/4
f-
1/4
on
on
```

```
-
1/4
1/4
1/4
f-
1/4
fr
1/4
xtutalalatlaco
```

In [17]:
```
new_context = torch.tensor(encode(["Mushroom"], main_text=text, tiktoken=False),
generated_text = model.generate(new_context, max_new_tokens=500)[0].tolist()
print(decode(generated_text, text=text, tiktoken=False))
```

```
C:\Users\ikath\AppData\Local\Temp\ipykernel_11680\2734706773.py:1: UserWarning: T
o copy construct from a tensor, it is recommended to use sourceTensor.detach().cl
one() or sourceTensor.detach().clone().requires_grad_(True), rather than torch.te
nsor(sourceTensor).
  new_context = torch.tensor(encode(["Mushroom"], main_text=text, tiktoken=Fals
e), dtype=torch.long, device=DEVICE).view(1, -1)
```

```
Mushroom
t-
1/2tablemponfreretablaconlesStu
on
-
1
oxt
ddixtur
-
1/4
-
1/4
Studr
Stuch-
1/¼ephow
1/4
on
on
onextu
elatu
laton
fr

fr
-
1
-
1/23.
fr
fr
-
1/4
Stheatut
25
f-
1/4
fr
1/25
-
1/25
y
1/4
fre
-
1.
wilat
ou
on
lilabl
on
oud
-
1
he
-
1
hichep
on
-
1/4
```

```
1/25-
fr
fr
-
ablacon
wicon
-
1/4
-
1/2cheon
ply
-
1/4
-
1/4
-
1/25ond
-
3
one
on
-
1/4
-
1/4
-
1/4
f-
1/4
f-
1/4
fro
1/4
-
1/2conddlatudplathe
-
-
fr
1/2labled
-
-
3/4
on
S
lated
-
1/4
1/4
Stud
-
1/2cked
-
-
3/4
ooon
-
1/4
1/4
-
1/4
```

```
-
1/4
sababablab
```

In [18]:
```
new_context = torch.tensor(encode(["Salt"], main_text=text, tiktoken=False), dty
generated_text = model.generate(new_context, max_new_tokens=500)[0].tolist()
print(decode(generated_text, text=text, tiktoken=False))
```

```
C:\Users\ikath\AppData\Local\Temp\ipykernel_11680\1049793302.py:1: UserWarning: T
o copy construct from a tensor, it is recommended to use sourceTensor.detach().cl
one() or sourceTensor.detach().clone().requires_grad_(True), rather than torch.te
nsor(sourceTensor).
  new_context = torch.tensor(encode(["Salt"], main_text=text, tiktoken=False), dt
ype=torch.long, device=DEVICE).view(1, -1)
```

```
Salt
in
the
preheated
oven
for
25
minutes
on
f-
1/4
oveg
f
-
1
-
1/23/4
-
1/23/2lablaondon
f-
1/4
f-
1/4
ond
f-
plablap-
1/4
fr
filary
1/4
fr
wil
5
-
1/4
wi-
1/4
1/4
1/4
1/4
on
Sm
ow
-
1/4
-
1/4
-
1/2tatlatlat.
jucher
on
1/4
-
1/4
1/25-
1/25
Stud
SBlaton
f-
3/4
```

```
on
1/4
2chon
onlatlat.
oved
S
-
1/4
on
25
f-
1/4
fr
Stus
her
fr
atalacor
1/4
f-
1/4
fr
1/4
-
1/2lablablabuhed
-
1/4
-
1/2½
xtudpond
-
1.
onat.
fre
fr
Stu
-
1.
Stud
S
at.
he
f
S
Sto
S
fr
1/4
fr

d
Stuila
e
f
-
1/25
on
1/4
pouc
on
-.
```

```
oory
Stato
```

Evaluate tiktoken

In [19]:
```
model_path_tiktoken = "recipe_gpt_full_tiktoken.pt"
model_tiktoken = torch.load(model_path_tiktoken, weights_only=False, map_locatio
model_tiktoken.eval()
```

Out[19]:
```
GPTModel(
  (token_embedding_table): Embedding(50257, 512)
  (pos_emb_table): Embedding(40, 512)
  (blocks): Sequential(
    (0): Block(
      (sa): MultiHeadAttention(
        (heads): ModuleList(
          (0-7): 8 x Head(
            (key): Linear(in_features=512, out_features=64, bias=False)
            (query): Linear(in_features=512, out_features=64, bias=False)
            (value): Linear(in_features=512, out_features=64, bias=False)
            (dropout): Dropout(p=0.2, inplace=False)
          )
        )
        (proj): Linear(in_features=512, out_features=512, bias=True)
        (dropout): Dropout(p=0.2, inplace=False)
      )
      (ffwd): FeedForward(
        (net): Sequential(
          (0): Linear(in_features=512, out_features=2048, bias=True)
          (1): ReLU()
          (2): Linear(in_features=2048, out_features=512, bias=True)
          (3): Dropout(p=0.2, inplace=False)
        )
      )
      (ln1): LayerNorm((512,), eps=1e-05, elementwise_affine=True)
      (ln2): LayerNorm((512,), eps=1e-05, elementwise_affine=True)
    )
    (1): Block(
      (sa): MultiHeadAttention(
        (heads): ModuleList(
          (0-7): 8 x Head(
            (key): Linear(in_features=512, out_features=64, bias=False)
            (query): Linear(in_features=512, out_features=64, bias=False)
            (value): Linear(in_features=512, out_features=64, bias=False)
            (dropout): Dropout(p=0.2, inplace=False)
          )
        )
        (proj): Linear(in_features=512, out_features=512, bias=True)
        (dropout): Dropout(p=0.2, inplace=False)
      )
      (ffwd): FeedForward(
        (net): Sequential(
          (0): Linear(in_features=512, out_features=2048, bias=True)
          (1): ReLU()
          (2): Linear(in_features=2048, out_features=512, bias=True)
          (3): Dropout(p=0.2, inplace=False)
        )
      )
      (ln1): LayerNorm((512,), eps=1e-05, elementwise_affine=True)
      (ln2): LayerNorm((512,), eps=1e-05, elementwise_affine=True)
    )
    (2): Block(
      (sa): MultiHeadAttention(
        (heads): ModuleList(
          (0-7): 8 x Head(
            (key): Linear(in_features=512, out_features=64, bias=False)
            (query): Linear(in_features=512, out_features=64, bias=False)
            (value): Linear(in_features=512, out_features=64, bias=False)
            (dropout): Dropout(p=0.2, inplace=False)
```

```
        )
      )
      (proj): Linear(in_features=512, out_features=512, bias=True)
      (dropout): Dropout(p=0.2, inplace=False)
    )
    (ffwd): FeedForward(
      (net): Sequential(
        (0): Linear(in_features=512, out_features=2048, bias=True)
        (1): ReLU()
        (2): Linear(in_features=2048, out_features=512, bias=True)
        (3): Dropout(p=0.2, inplace=False)
      )
    )
    (ln1): LayerNorm((512,), eps=1e-05, elementwise_affine=True)
    (ln2): LayerNorm((512,), eps=1e-05, elementwise_affine=True)
  )
  (3): Block(
    (sa): MultiHeadAttention(
      (heads): ModuleList(
        (0-7): 8 x Head(
          (key): Linear(in_features=512, out_features=64, bias=False)
          (query): Linear(in_features=512, out_features=64, bias=False)
          (value): Linear(in_features=512, out_features=64, bias=False)
          (dropout): Dropout(p=0.2, inplace=False)
        )
      )
      (proj): Linear(in_features=512, out_features=512, bias=True)
      (dropout): Dropout(p=0.2, inplace=False)
    )
    (ffwd): FeedForward(
      (net): Sequential(
        (0): Linear(in_features=512, out_features=2048, bias=True)
        (1): ReLU()
        (2): Linear(in_features=2048, out_features=512, bias=True)
        (3): Dropout(p=0.2, inplace=False)
      )
    )
    (ln1): LayerNorm((512,), eps=1e-05, elementwise_affine=True)
    (ln2): LayerNorm((512,), eps=1e-05, elementwise_affine=True)
  )
  (4): Block(
    (sa): MultiHeadAttention(
      (heads): ModuleList(
        (0-7): 8 x Head(
          (key): Linear(in_features=512, out_features=64, bias=False)
          (query): Linear(in_features=512, out_features=64, bias=False)
          (value): Linear(in_features=512, out_features=64, bias=False)
          (dropout): Dropout(p=0.2, inplace=False)
        )
      )
      (proj): Linear(in_features=512, out_features=512, bias=True)
      (dropout): Dropout(p=0.2, inplace=False)
    )
    (ffwd): FeedForward(
      (net): Sequential(
        (0): Linear(in_features=512, out_features=2048, bias=True)
        (1): ReLU()
        (2): Linear(in_features=2048, out_features=512, bias=True)
        (3): Dropout(p=0.2, inplace=False)
      )
```

```
      )
      (ln1): LayerNorm((512,), eps=1e-05, elementwise_affine=True)
      (ln2): LayerNorm((512,), eps=1e-05, elementwise_affine=True)
    )
    (5): Block(
      (sa): MultiHeadAttention(
        (heads): ModuleList(
          (0-7): 8 x Head(
            (key): Linear(in_features=512, out_features=64, bias=False)
            (query): Linear(in_features=512, out_features=64, bias=False)
            (value): Linear(in_features=512, out_features=64, bias=False)
            (dropout): Dropout(p=0.2, inplace=False)
          )
        )
        (proj): Linear(in_features=512, out_features=512, bias=True)
        (dropout): Dropout(p=0.2, inplace=False)
      )
      (ffwd): FeedForward(
        (net): Sequential(
          (0): Linear(in_features=512, out_features=2048, bias=True)
          (1): ReLU()
          (2): Linear(in_features=2048, out_features=512, bias=True)
          (3): Dropout(p=0.2, inplace=False)
        )
      )
      (ln1): LayerNorm((512,), eps=1e-05, elementwise_affine=True)
      (ln2): LayerNorm((512,), eps=1e-05, elementwise_affine=True)
    )
  )
  (ln_f): LayerNorm((512,), eps=1e-05, elementwise_affine=True)
  (lm_ffw_head): Linear(in_features=512, out_features=50257, bias=True)
)
```

In [20]:
```
sos_context = torch.zeros((1, 1), dtype=torch.long, device=DEVICE)
generated_text = model_tiktoken.generate(sos_context, max_new_tokens=500)[0].tol
print(decode(generated_text, tiktoken=True))
```

```
!(15-ounce) canscannellini beans, rinsed and drained
- 1pintgrape tomatoes
- 1/2cupreduced sodium chicken broth
- 2teaspo, boneless chicken- ¼cupbroccoli, boneless chicken to taste
-skated oven to taste
-shaped oil
- 1tablespoonunsalted butter
- 1/2teaspo, boneless chicken; mix to taste
- 1/2teaspo, boneless chicken broth to taste
- 2teaspo, finelyickenMEN oil
- 1/2teaspo, finelywhole kernel corn, finelymerteaspo, finelymer diabetesPRESSather all ingredients.Dotdash Meredith Food Studios
- 1teaspo, finely 122 oil
-shaped oil
- ¼cupchicken bouillon, boneless chicken- ¼cupchicken bouillon
- ¼cupwholive oil
- ¼cupchicken bouillon, bonelessfinalscupshigh- ¼cup yogurt, boneless chicken thighs
- ¼cup, boneless funnel, finelyAllrecipes.Allrecipes.com/2.com/2.com/2teaspo, lemon juice, finely Ro, finely ½teaspo, add chicken thighs
-skteaspo, finely chicken broth to taste
- 1/2teaspo, finely Prop Styling: me covered, finely Guests to taste
- 1/2teaspo, finelyini beans, bonelesshigh heat oil
- 1/2teaspo, boneless chicken to taste
- ¼cupchicken bouillon with a oil
- 1/2teaspo, finelyini, finelyists An instant read thermometer to taste
- ¼cupchicken bouillon, finelyi, boneless chicken broth to taste
-shaped me, finely read thermometer inserted near the center of balls with a 9x13
-shapedgam, justheavy cream into a large chunks
- ¼cupchopped fresh cilantro and 1/2teaspo, finelymer. Gather all ingredients.Dotdash Meredith Food Studios
-shapedo, finely 2teaspo, add chicken to taste
-low with a bowl with a large pot of balls with a 9x13-recipe- ¼cupchicken bou80
-shaped
```

In [23]:
```
new_context = torch.tensor(encode(["Chicken recipe"], tiktoken=True), dtype=torc
generated_text = model_tiktoken.generate(new_context, max_new_tokens=500)[0].tol
print(decode(generated_text, tiktoken=True))
```

```
C:\Users\ikath\AppData\Local\Temp\ipykernel_11680\3420255904.py:1: UserWarning: To copy construct from a tensor, it is recommended to use sourceTensor.detach().clone() or sourceTensor.detach().clone().requires_grad_(True), rather than torch.tensor(sourceTensor).
  new_context = torch.tensor(encode(["Chicken recipe"], tiktoken=True), dtype=torch.long, device=DEVICE).view(1, -1)
```

```
Chicken recipe
- 1mediumonion, cut into thin wedges
- 2tablespoonsdrywhite wine
- 1teaspoonchoppedfresh thymeor1/4teaspo, addteaspo, boneless chicken to taste
- 1/2teaspo, finely inserted near the center of balls with a food processor; mix
to taste
- ½teaspo, finelyini beans, finelymersk. Gather all ingredients.Dotdash Meredith
Food Studios
-shaped Studios
-shaped oil
-sk coil Spring with a fork, finelyshaped oil
- 2teaspo, just UAE Conservation finelymediumonion, boneless chicken to taste
- 1cupgrated oven- 1 hour to taste
- ¼cupgrated oven temperature.Dotdash Meredith Food Studios
- ¼cupbroccoli florets, boneless chicken clean, finelycot a large pot of balls, f
inely crackers, finely 2teaspo, finelyencia oil
-shaped oil
-skini- ¼cupwater, finely least Proledy water to taste
- ½teaspo, finelyreal becomes Academy a large pot of balls with a Comerecipeas, b
oneless chicken; mix to taste
-shaped oil
-shaped oil
-shaped oil
-shaped oil
- 1/2.Dotdash Meredith Food Studios
- 2teaspo, finely 2teaspo, boneless chicken broth to taste
- 1/2teaspo, finelyman-skshaped yogurt with a 9x13- 1/2.Dotdash Meredith Food Stu
dios
-shaped yogurt, finelyoodoohigh heat oil
-shaped oil
-shaped oil
-shaped cheese
- 1/2cupredded Parm- 1/2teaspo, finelyAllrecipes.com/2teaspo, finelymerhigh heat
oil
- 1/2teaspo, finelyazy-sk Jonah-skteaspo, finelysk80
-shaped garlic, finely 3slic, finelyated oven until frag needed oil
-shaped read thermometer inserted near the center, add chicken-shaped oil
-recipe- ¼cupchoppedfresh thymeor theawed potatoes until reconstituted, finely Ga
ther all ingredients.Dotdash Meredith Food Studios
- ¼cupgrated ovenated oven until reconstituted, just wooden spoon until
```

In [24]:
```
new_context = torch.tensor(encode(["Mushroom"], tiktoken=True), dtype=torch.long
generated_text = model_tiktoken.generate(new_context, max_new_tokens=500)[0].tol
print(decode(generated_text, tiktoken=True))
```

```
C:\Users\ikath\AppData\Local\Temp\ipykernel_11680\2154474370.py:1: UserWarning: T
o copy construct from a tensor, it is recommended to use sourceTensor.detach().cl
one() or sourceTensor.detach().clone().requires_grad_(True), rather than torch.te
nsor(sourceTensor).
  new_context = torch.tensor(encode(["Mushroom"], tiktoken=True), dtype=torch.lon
g, device=DEVICE).view(1, -1)
```

```
Mushroom
- 1/2cupwholeroasted almonds
- 4clovesgarlic
- 1teaspoonkosher salt, plus more to taste
- 1cupgrated oven until evenlymer- ¼cupgrated oven cheese
-shaped yogurt, finelyatoes, boneless chickeno, finely 2teaspo, finely: https://w
ww.allrecipes.com/recipe- Transfer lobstercot chilies in a 9x13-sk13-recipe/baby-
skotine finelyUpgrade-skcreamteaspo,fund forms a spoon, boneless chicken broth to
taste
-shaped Dream oil
-shaped oil
- 1cupgrated oven until golden, finely 2teaspo, Add pasta
-shaped yogurt, finelyshaped oil
- 1/2teaspo, boneless chicken thighs
- ¼cupchopped tomatoes in a large chunks
- 1/2teaspo, boneless chicken thighs
- 2teaspo, finely all ingredients.Dotdash Meredith Food Studios
-shaped 3slicesfresh thymeor the refrigerator and cut into small pieces.Dotdash M
eredith Food Studios
- partnered with a fork, finely 2teaspo, finely squeeze bottleongar oil in a 9x13
-shapedño Popper-shapedshaped becomes Influence-shaped cheese
-shaped processor; mix to taste
-shaped oil
- 2teaspo, boneless chicken mixture, finely while pasta
- 1 Bring broth to taste
- 1/2.Dotdash Meredith Food Studios
- 1/2teaspo, finelyhus, boneless chicken; stir together egg, finelymanicotti past
a
-shaped balls with a wooden spoon, boneless chicken to taste
- 1/2teaspo, finely syrup, boneless chicken mixture, boneless chicken broth to ta
ste
- 1teaspo, finely 2teaspo, finely with a food processor; pulse a accomplishmentsc
ot a 9x13-shaped oil
-shaped oil
- 1/2teaspo, finelyshaped oil
- 1 things a few times, finely hours, while pasta
- 1/2teaspo, boneless chicken-shapedlicesfresh thymeor the bowl, boneless chicken
thighs
-recipe-sk oil
-shaped oil
- 1/2teaspo, boneless chicken broth
- 2teaspo
```

In [25]:
```
new_context = torch.tensor(encode(["Salt"], tiktoken=True), dtype=torch.long, de
generated_text = model_tiktoken.generate(new_context, max_new_tokens=500)[0].tol
print(decode(generated_text, tiktoken=True))
```

```
C:\Users\ikath\AppData\Local\Temp\ipykernel_11680\313191504.py:1: UserWarning: To
copy construct from a tensor, it is recommended to use sourceTensor.detach().clon
e() or sourceTensor.detach().clone().requires_grad_(True), rather than torch.tens
or(sourceTensor).
  new_context = torch.tensor(encode(["Salt"], tiktoken=True), dtype=torch.long, d
evice=DEVICE).view(1, -1)
```

```
Salt
- 1cubechicken bouillon
- 4largecarrots, peeled and cut into large chunks
- 4largepotatoes, peeled and cut into large chunks
- 4zucchinihigh heat oil
- ¼cupchicken broth to taste
- 1/2teaspo, boneless chicken broth to taste
- ¼cupbroccoli, boneless chicken thighs constitutional: me, boneless chicken brot
h to taste
- ¼cupchoppedfresh thymeor the simple: https://www.allrecipes.com/recipe/2teaspo,
add pepper to taste
-sk browned, finelymer bagchicken serious Using a 9x13-recipe/2teaspo, finelyatoe
s, finely Of
- 1/2teaspo, finelyhigh heat oil
- ¼cupchicken bouillon, finely visits- ¼cupItalian-shaped Actual, boneless chicke
n broth to taste
-shaped oil
- ¼cupchopped fresh cilantro and cut into large pot of balls with a little sweete
r, add chicken thighs
- ¼cupredded C).
- ¼cupchicken broth to taste
- ¼cupItalian- ¼cupbro: me, finely tape Addelyn Evans
- ¼cupchoppedfresh thyme, add chicken broth to taste
- ¼cupgrated oven temperature. Gather all ingredients.Dotdash Meredith Food Studi
os
-shaped oil
-shaped oil
-shaped ensure dough into a 9x13-sk pointer Belgium into a boil.Dotdash Meredith
Food Studios
- 1/2. Gather all ingredients. Gather the- ¼cupwhole kernel corn, finely Gather t
he smoot over salmon fil.Dotdash Meredith Food Studios
- ¼cupheavy, finely up to a boil.Dotdash Meredith Food Studios
- ¼cup yogurt, finely 2teaspo, boneless chicken broth to aicken bouillon, finelys
haped oil
-shapedshaped80
- ¼cupchopped fresh cilantro and vanilla to skillet.Dotdash Meredith Food Studios
- 2teaspo, finely read thermometer inserted near the center, boneless chicken bro
th to taste
- ¼cupchocolatezuc Popper-shapedpper Twists
-shaped oil
- ¼cupbroccoli, boneless chicken broth
-
```

In [ ]: