MLX MinGPT

In [ ]:
```python
import mlx.core as mx
import mlx.nn as nn
import mlx.optimizers as optim
from mlx.utils import tree_flatten, tree_map, tree_unflatten
```

In [ ]:
```python
# hyperparameters
batch_size = 16 # how many independent sequences will we process in parallel?
block_size = 32 # what is the maximum context length for predictions?
max_iters = 1000
eval_interval = 100
learning_rate = 1e-3
eval_iters = 200
n_embd = 64
n_head = 4
n_layer = 4
dropout = 0.00
```

In [ ]:
```python
# device = 'cuda' if torch.cuda.is_available() else 'cpu'
device = mx.default_device()
```

In [ ]:
```python
# ------------

mx.random.seed(1337)

# wget https://raw.githubusercontent.com/karpathy/char-rnn/master/data/tinyshakespeare/input.txt
with open('input.txt', 'r', encoding='utf-8') as f:
    text = f.read()
```

In [ ]:
```python
# here are all the unique characters that occur in this text
chars = sorted(list(set(text)))
vocab_size = len(chars)
# 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) }
encode = lambda s: [stoi[c] for c in s] # encoder: take a string, output a list of integers
decode = lambda l: ''.join([itos[i] for i in l]) # decoder: take a list of integers, output a string
```

In [ ]:
```python
# Train and test splits
data = mx.array(encode(text), dtype=mx.int64)
n = int(0.9*len(data)) # first 90% will be train, rest val
train_data = data[:n]
val_data = data[n:]
```

In [ ]:
```python
# data loading
def get_batch(split):
    # generate a small batch of data of inputs x and targets y
    data = train_data if split == 'train' else val_data
    ix = mx.random.randint(0, len(data) - block_size, (batch_size,))
    ix = [i.item() for i in ix]
    x = mx.stack([data[i:i+block_size] for i in ix])
    y = mx.stack([data[i+1:i+block_size+1] for i in ix])
    # x, y = x.to(device), y.to(device)
    return x, y
```

In [ ]:
```python
class Head(nn.Module):
    """ one head of self-attention """

    def __init__(self, head_size):
        super().__init__()
        self.key = nn.Linear(n_embd, head_size, bias=False)
        self.query = nn.Linear(n_embd, head_size, bias=False)
        self.value = nn.Linear(n_embd, head_size, bias=False)
        self.tril = mx.tril(mx.ones([block_size, block_size]))
        self.dropout = nn.Dropout(dropout)

    def __call__(self, x):
        B,T,C = x.shape
        k = self.key(x)   # (B,T,C)
        q = self.query(x) # (B,T,C)
        # compute attention scores ("affinities")
        wei = q @ k.transpose((0,2,1)) * C**-0.5 # (B, T, C) @ (B, C, T) -> (B, T, T)
        mask = self.tril[:T, :T] == 0
        wei = mx.where(mask, float('-inf'), wei) # (B, T, T)
        wei = nn.softmax(wei, axis=-1) # (B, T, T)
        wei = self.dropout(wei)
        # perform the weighted aggregation of the values
        v = self.value(x) # (B,T,C)
        out = wei @ v # (B, T, T) @ (B, T, C) -> (B, T, C)
        return out
```

In [ ]:
```python
class MultiHeadAttention(nn.Module):
    """ multiple heads of self-attention in parallel """
```

```python
    def __init__(self, num_heads, head_size):
        super().__init__()
        self.heads = [Head(head_size) for _ in range(num_heads)]
        self.proj = nn.Linear(n_embd, n_embd)
        self.dropout = nn.Dropout(dropout)

    def __call__(self, x):
        out = mx.concatenate([h(x) for h in self.heads], axis=-1)
        out = self.dropout(self.proj(out))
        return out

class FeedFoward(nn.Module):
    """ a simple linear layer followed by a non-linearity """

    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),
        )

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

In [ ]:
```python
class Block(nn.Module):
    """ Transformer block: communication followed by computation """

    def __init__(self, n_embd, n_head):
        # n_embd: embedding dimension, n_head: the number of heads we'd like
        super().__init__()
        head_size = n_embd // n_head
        self.sa = MultiHeadAttention(n_head, head_size)
        self.ffwd = FeedFoward(n_embd)
        self.ln1 = nn.LayerNorm(n_embd)
        self.ln2 = nn.LayerNorm(n_embd)

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

In [ ]:
```python
# super simple bigram model
class LanguageModel(nn.Module):

    def __init__(self):
        super().__init__()
        # each token directly reads off the logits for the next token from a lookup table
        self.token_embedding_table = nn.Embedding(vocab_size, n_embd)
        self.position_embedding_table = nn.Embedding(block_size, n_embd)
        self.blocks = nn.Sequential(*[Block(n_embd, n_head=n_head) for _ in range(n_layer)])
        self.ln_f = nn.LayerNorm(n_embd) # final layer norm
        self.lm_head = nn.Linear(n_embd, vocab_size)

    def __call__(self, idx, targets=None):
        B, T = idx.shape

        # idx and targets are both (B,T) tensor of integers
        tok_emb = self.token_embedding_table(idx) # (B,T,C)
        pos_emb = self.position_embedding_table(mx.arange(T)) # (T,C)
        x = tok_emb + pos_emb # (B,T,C)
        x = self.blocks(x) # (B,T,C)
        x = self.ln_f(x) # (B,T,C)
        logits = self.lm_head(x) # (B,T,vocab_size)

        if targets is None:
            loss = None
        else:
            B, T, C = logits.shape
            logits = logits.reshape(B*T, C)
            targets = targets.reshape(B*T)
            loss = nn.losses.cross_entropy(logits, targets, reduction='mean')

        return logits, loss

    def generate(self, idx, max_new_tokens):
        # idx is (B, T) array of indices in the current context
        for _ in range(max_new_tokens):
            # crop idx to the last block_size tokens
            idx_cond = idx[:, -block_size:]
            # get the predictions
            logits, _ = self(idx_cond)
            # focus only on the last time step
            logits = logits[:, -1, :] # becomes (B, C)
            # apply softmax to get probabilities
            probs = nn.softmax(logits, axis=-1) # (B, C)
            # sample from the distribution
            idx_next = mx.random.categorical(probs, axis=-1, num_samples=1)
```

```
            # append sampled index to the running sequence
            idx = mx.concatenate((idx, idx_next), axis=1) # (B, T+1)
        return idx
```

In [ ]:
```
model = LanguageModel()
```

In [ ]:
```
params = model.parameters()
```

In [ ]:
```
# print the number of parameters in the model
p = sum(v.size for _, v in tree_flatten(model.parameters())) / 10**6
print(f"Total parameters {p:.3f}M")
```

```
Total parameters 0.226M
```

In [ ]:
```
# create an optimizer
optimizer = optim.AdamW(learning_rate)
```

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

def loss_fn(model, x, y):
    _, loss = model(x, y)
    return loss

def step(model, optimizer, inputs, targets):
    loss_and_grad_fn = nn.value_and_grad(model, loss_fn)
    loss, grads = loss_and_grad_fn(model, inputs, targets)
    optimizer.update(model, grads)
    return loss
```

In [ ]:
```
import time
start = time.time()
model.train()
for iter in range(max_iters):
    # every once in a while evaluate the loss on train and val sets
    if iter % eval_interval == 0 or iter == max_iters - 1:
        losses = estimate_loss(model)
        print(f"step {iter}: train loss {losses['train'].item():.4f}, val loss {losses['val'].item():.4f}")

    # sample a batch of data
    xb, yb = get_batch('train')

    # evaluate the loss
    step(model, optimizer, xb, yb)
end = time.time()
print(f"Training complete in {end-start}s")
```

```
step 0: train loss 4.3217, val loss 4.3203
step 100: train loss 2.5272, val loss 2.5337
step 200: train loss 2.3866, val loss 2.3862
step 300: train loss 2.3022, val loss 2.3156
step 400: train loss 2.2350, val loss 2.2326
step 500: train loss 2.1573, val loss 2.1803
step 600: train loss 2.1053, val loss 2.1409
step 700: train loss 2.0655, val loss 2.1170
step 800: train loss 2.0389, val loss 2.0914
step 900: train loss 1.9903, val loss 2.0560
step 999: train loss 1.9541, val loss 2.0427
Training complete in 70.77645587921143s
```

In [ ]:
```
print(model)
tree_flatten(model)
```

```
LanguageModel(
  (token_embedding_table): Embedding(65, 64)
  (position_embedding_table): Embedding(32, 64)
  (blocks): Sequential(
    (layers.0): Block(
      (sa): MultiHeadAttention(
        (heads.0): Head(
          (key): Linear(input_dims=64, output_dims=16, bias=False)
          (query): Linear(input_dims=64, output_dims=16, bias=False)
          (value): Linear(input_dims=64, output_dims=16, bias=False)
          (dropout): Dropout(p=0.0)
        )
        (heads.1): Head(
          (key): Linear(input_dims=64, output_dims=16, bias=False)
          (query): Linear(input_dims=64, output_dims=16, bias=False)
          (value): Linear(input_dims=64, output_dims=16, bias=False)
          (dropout): Dropout(p=0.0)
        )
```

```
    (heads.2): Head(
      (key): Linear(input_dims=64, output_dims=16, bias=False)
      (query): Linear(input_dims=64, output_dims=16, bias=False)
      (value): Linear(input_dims=64, output_dims=16, bias=False)
      (dropout): Dropout(p=0.0)
    )
    (heads.3): Head(
      (key): Linear(input_dims=64, output_dims=16, bias=False)
      (query): Linear(input_dims=64, output_dims=16, bias=False)
      (value): Linear(input_dims=64, output_dims=16, bias=False)
      (dropout): Dropout(p=0.0)
    )
    (proj): Linear(input_dims=64, output_dims=64, bias=True)
    (dropout): Dropout(p=0.0)
  )
  (ffwd): FeedFoward(
    (net): Sequential(
      (layers.0): Linear(input_dims=64, output_dims=256, bias=True)
      (layers.1): ReLU()
      (layers.2): Linear(input_dims=256, output_dims=64, bias=True)
      (layers.3): Dropout(p=0.0)
    )
  )
  (ln1): LayerNorm(64, eps=1e-05, affine=True)
  (ln2): LayerNorm(64, eps=1e-05, affine=True)
)
(layers.1): Block(
  (sa): MultiHeadAttention(
    (heads.0): Head(
      (key): Linear(input_dims=64, output_dims=16, bias=False)
      (query): Linear(input_dims=64, output_dims=16, bias=False)
      (value): Linear(input_dims=64, output_dims=16, bias=False)
      (dropout): Dropout(p=0.0)
    )
    (heads.1): Head(
      (key): Linear(input_dims=64, output_dims=16, bias=False)
      (query): Linear(input_dims=64, output_dims=16, bias=False)
      (value): Linear(input_dims=64, output_dims=16, bias=False)
      (dropout): Dropout(p=0.0)
    )
    (heads.2): Head(
      (key): Linear(input_dims=64, output_dims=16, bias=False)
      (query): Linear(input_dims=64, output_dims=16, bias=False)
      (value): Linear(input_dims=64, output_dims=16, bias=False)
      (dropout): Dropout(p=0.0)
    )
    (heads.3): Head(
      (key): Linear(input_dims=64, output_dims=16, bias=False)
      (query): Linear(input_dims=64, output_dims=16, bias=False)
      (value): Linear(input_dims=64, output_dims=16, bias=False)
      (dropout): Dropout(p=0.0)
    )
    (proj): Linear(input_dims=64, output_dims=64, bias=True)
    (dropout): Dropout(p=0.0)
  )
  (ffwd): FeedFoward(
    (net): Sequential(
      (layers.0): Linear(input_dims=64, output_dims=256, bias=True)
      (layers.1): ReLU()
      (layers.2): Linear(input_dims=256, output_dims=64, bias=True)
      (layers.3): Dropout(p=0.0)
    )
  )
  (ln1): LayerNorm(64, eps=1e-05, affine=True)
  (ln2): LayerNorm(64, eps=1e-05, affine=True)
)
(layers.2): Block(
  (sa): MultiHeadAttention(
    (heads.0): Head(
      (key): Linear(input_dims=64, output_dims=16, bias=False)
      (query): Linear(input_dims=64, output_dims=16, bias=False)
      (value): Linear(input_dims=64, output_dims=16, bias=False)
      (dropout): Dropout(p=0.0)
    )
    (heads.1): Head(
      (key): Linear(input_dims=64, output_dims=16, bias=False)
      (query): Linear(input_dims=64, output_dims=16, bias=False)
      (value): Linear(input_dims=64, output_dims=16, bias=False)
      (dropout): Dropout(p=0.0)
    )
    (heads.2): Head(
      (key): Linear(input_dims=64, output_dims=16, bias=False)
      (query): Linear(input_dims=64, output_dims=16, bias=False)
      (value): Linear(input_dims=64, output_dims=16, bias=False)
      (dropout): Dropout(p=0.0)
    )
    (heads.3): Head(
      (key): Linear(input_dims=64, output_dims=16, bias=False)
      (query): Linear(input_dims=64, output_dims=16, bias=False)
      (value): Linear(input_dims=64, output_dims=16, bias=False)
      (dropout): Dropout(p=0.0)
```

```
            )
            (proj): Linear(input_dims=64, output_dims=64, bias=True)
            (dropout): Dropout(p=0.0)
          )
          (ffwd): FeedFoward(
            (net): Sequential(
              (layers.0): Linear(input_dims=64, output_dims=256, bias=True)
              (layers.1): ReLU()
              (layers.2): Linear(input_dims=256, output_dims=64, bias=True)
              (layers.3): Dropout(p=0.0)
            )
          )
          (ln1): LayerNorm(64, eps=1e-05, affine=True)
          (ln2): LayerNorm(64, eps=1e-05, affine=True)
        )
        (layers.3): Block(
          (sa): MultiHeadAttention(
            (heads.0): Head(
              (key): Linear(input_dims=64, output_dims=16, bias=False)
              (query): Linear(input_dims=64, output_dims=16, bias=False)
              (value): Linear(input_dims=64, output_dims=16, bias=False)
              (dropout): Dropout(p=0.0)
            )
            (heads.1): Head(
              (key): Linear(input_dims=64, output_dims=16, bias=False)
              (query): Linear(input_dims=64, output_dims=16, bias=False)
              (value): Linear(input_dims=64, output_dims=16, bias=False)
              (dropout): Dropout(p=0.0)
            )
            (heads.2): Head(
              (key): Linear(input_dims=64, output_dims=16, bias=False)
              (query): Linear(input_dims=64, output_dims=16, bias=False)
              (value): Linear(input_dims=64, output_dims=16, bias=False)
              (dropout): Dropout(p=0.0)
            )
            (heads.3): Head(
              (key): Linear(input_dims=64, output_dims=16, bias=False)
              (query): Linear(input_dims=64, output_dims=16, bias=False)
              (value): Linear(input_dims=64, output_dims=16, bias=False)
              (dropout): Dropout(p=0.0)
            )
            (proj): Linear(input_dims=64, output_dims=64, bias=True)
            (dropout): Dropout(p=0.0)
          )
          (ffwd): FeedFoward(
            (net): Sequential(
              (layers.0): Linear(input_dims=64, output_dims=256, bias=True)
              (layers.1): ReLU()
              (layers.2): Linear(input_dims=256, output_dims=64, bias=True)
              (layers.3): Dropout(p=0.0)
            )
          )
          (ln1): LayerNorm(64, eps=1e-05, affine=True)
          (ln2): LayerNorm(64, eps=1e-05, affine=True)
        )
      )
      (ln_f): LayerNorm(64, eps=1e-05, affine=True)
      (lm_head): Linear(input_dims=64, output_dims=65, bias=True)
    )
```

Out[ ]:

```
[('token_embedding_table.weight',
  array([[0.0569056, -0.108505, -0.0591656, ..., -0.0295052, -0.167381, -0.330299],
         [-0.0247517, 0.1248, -0.0542025, ..., -0.040112, -0.155497, -0.079975],
         [0.0888196, -0.133462, 0.104857, ..., 0.0700017, 0.372513, -0.196217],
         ...,
         [-0.304017, -0.111619, -0.255181, ..., 0.0155221, 0.261499, 0.156096],
         [-0.31015, -0.0777493, 0.142672, ..., -0.0192186, 0.0906891, 0.326396],
         [0.0921749, 0.0883812, 0.152577, ..., 0.238317, -0.0521913, 0.146304]], dtype=float32)),
 ('position_embedding_table.weight',
  array([[0.240161, -0.0972059, -0.0630608, ..., -0.146608, 0.156927, 0.0355572],
         [0.197704, 0.330197, 0.192751, ..., -0.00278693, 0.0464535, -0.256711],
         [0.0276793, -0.13228, 0.200821, ..., 0.0338127, -0.177899, 0.145335],
         ...,
         [-0.0649977, -0.0499931, -0.00518212, ..., 0.240493, -0.039005, 0.0695617],
         [-0.159428, -0.119792, 0.166959, ..., 0.17033, -0.0632758, 0.0185298],
         [-0.163784, -0.2995, -0.112194, ..., 0.198379, -0.109867, 0.0965027]], dtype=float32)),
 ('blocks.layers.0.sa.heads.0.key.weight',
  array([[-0.212204, -0.0116126, -0.00476231, ..., -0.124191, 0.0886937, 0.139496],
         [-0.0290291, 0.100437, -0.293302, ..., -0.135936, -0.0459674, 0.0556262],
         [0.0554746, 0.150323, -0.0712375, ..., -0.128718, 0.148299, -0.0854722],
         ...,
         [0.170673, 0.218622, -0.158681, ..., 0.0336357, -0.017136, -0.0175923],
         [0.0214662, 0.148259, 0.26242, ..., -0.0483463, 0.129475, -0.161671],
         [0.088504, 0.139545, -0.186668, ..., -0.0710314, 0.117455, 0.0199894]], dtype=float32)),
 ('blocks.layers.0.sa.heads.0.query.weight',
  array([[-0.0777069, -0.15631, 0.0402544, ..., 0.0158825, -0.112124, 0.231638],
         [0.216487, 0.0463411, -0.105414, ..., 0.151841, 0.0889331, -0.0843533],
         [0.220334, 0.196224, -0.101364, ..., 0.0588995, 0.169689, 0.044681],
         ...,
         [0.0409375, -0.0613336, 0.268037, ..., -0.122839, 0.0362136, -0.0826811],
         [-0.0925311, -0.0826283, 0.0531075, ..., 0.214324, -0.00017696, 0.0751491],
```

```
       [-0.0705703, 0.150746, -0.134885, ..., -0.0147897, -0.0380374, 0.0546641]], dtype=float32)),
('blocks.layers.0.sa.heads.0.value.weight',
 array([[-0.0083666, 0.0457222, 0.0860089, ..., -0.0180602, -0.0482407, -0.0188551],
       [0.0570256, 0.00801994, 0.00894356, ..., -0.0240573, 0.014144, 0.0653234],
       [0.0322991, -0.0131179, 0.0356787, ..., 0.112579, -0.026229, 0.137099],
       ...,
       [0.0130809, -0.0669691, -0.00962943, ..., -0.0633317, 0.0448235, 0.0699695],
       [0.137957, -0.0355169, -0.00510415, ..., -0.0357121, 0.0270071, 0.0680251],
       [0.0612603, 0.0416675, -0.0316414, ..., 0.14787, -0.0228016, -0.0104794]], dtype=float32)),
('blocks.layers.0.sa.heads.0.tril',
 array([[0.990034, 0, 0, ..., 0, 0, 0],
       [0.990034, 0.990034, 0, ..., 0, 0, 0],
       [0.990034, 0.990034, 0.990034, ..., 0, 0, 0],
       ...,
       [0.990034, 0.990034, 0.990034, ..., 0.990034, 0, 0],
       [0.990034, 0.990034, 0.990034, ..., 0.990034, 0.990034, 0],
       [0.990034, 0.990034, 0.990034, ..., 0.990034, 0.990034, 0.990034]], dtype=float32)),
('blocks.layers.0.sa.heads.1.key.weight',
 array([[0.0741267, 0.336166, -0.0841677, ..., -0.256574, -0.0333224, -0.0274585],
       [0.292252, -0.172843, 0.023563, ..., -0.136797, -0.0350301, 0.052731],
       [-0.154734, 0.0500228, 0.200013, ..., 0.223372, -0.0112552, 0.067084],
       ...,
       [0.129644, 0.147772, 0.0840407, ..., -0.0952977, 0.34527, -0.0424926],
       [-0.0959521, 0.0962014, -0.114042, ..., 0.185166, 0.2131, -0.0807275],
       [0.0832532, -0.0108217, 0.00503111, ..., 0.0223744, 0.131232, -0.0434297]], dtype=float32)),
('blocks.layers.0.sa.heads.1.query.weight',
 array([[0.0617979, 0.162937, -0.0370589, ..., -0.0330176, 0.381425, 0.279078],
       [0.055526, -0.230617, -0.0475865, ..., 0.0515149, 0.06851, -0.125395],
       [-0.358368, -0.406894, 0.101085, ..., 0.104353, 0.0929611, 0.164325],
       ...,
       [0.171947, 0.138839, -0.100574, ..., -0.111097, 0.0113719, 0.0353256],
       [-0.0728198, 0.238142, -0.096829, ..., 0.314482, -0.345996, 0.187388],
       [0.0867947, 0.0497772, 0.102624, ..., -0.0292544, 0.0470783, -0.0712383]], dtype=float32)),
('blocks.layers.0.sa.heads.1.value.weight',
 array([[-0.0458363, -0.136903, 0.0985585, ..., -0.152861, 0.175256, 0.0456455],
       [-0.135467, -0.0501712, -0.0789093, ..., 0.0318343, 0.0922103, 0.13947],
       [-0.0503074, -0.11178, 0.0359108, ..., 0.195892, 0.193848, 0.14076],
       ...,
       [-0.0161059, 0.100328, -0.0501909, ..., -0.128425, 0.11128, -0.0468212],
       [-0.173121, -0.10779, 0.0262938, ..., -0.0761064, 0.033183, 0.0761139],
       [-0.0116229, -0.162631, -0.0462601, ..., -0.018762, 0.0290576, 0.0358211]], dtype=float32)),
('blocks.layers.0.sa.heads.1.tril',
 array([[0.990034, 0, 0, ..., 0, 0, 0],
       [0.990034, 0.990034, 0, ..., 0, 0, 0],
       [0.990034, 0.990034, 0.990034, ..., 0, 0, 0],
       ...,
       [0.990034, 0.990034, 0.990034, ..., 0.990034, 0, 0],
       [0.990034, 0.990034, 0.990034, ..., 0.990034, 0.990034, 0],
       [0.990034, 0.990034, 0.990034, ..., 0.990034, 0.990034, 0.990034]], dtype=float32)),
('blocks.layers.0.sa.heads.2.key.weight',
 array([[0.225742, 0.0395885, -0.167717, ..., 0.0664072, -0.0565393, 0.102331],
       [0.138366, 0.132621, -0.0674459, ..., -0.122743, 0.148749, 0.10407],
       [-0.134284, 0.125783, -0.0864195, ..., 0.160739, 0.0162833, -0.0183218],
       ...,
       [-0.135364, -0.105325, 0.0599173, ..., 0.0281735, 0.0156238, 0.154578],
       [-0.0819611, 0.016915, 0.0629366, ..., -0.0223734, 0.105923, -0.174631],
       [-0.0127577, -0.12043, 0.0822888, ..., -0.0689502, 0.165715, -0.0584451]], dtype=float32)),
('blocks.layers.0.sa.heads.2.query.weight',
 array([[-0.0646671, 0.0881438, 0.0546715, ..., 0.182205, -0.00130001, 0.180221],
       [-0.322763, 0.23477, 0.0747066, ..., -0.108083, 0.00185001, 0.0806403],
       [0.0257742, 0.110053, -0.132245, ..., 0.0564632, -0.25561, -0.0492529],
       ...,
       [-0.136825, -0.186873, 0.234908, ..., 0.0390865, 0.0312915, 0.105961],
       [0.0506164, -0.120311, 0.154555, ..., 0.153619, 0.0285971, 0.0988147],
       [0.012863, 0.00186447, 0.271138, ..., 0.0688859, 0.161953, -0.061845]], dtype=float32)),
('blocks.layers.0.sa.heads.2.value.weight',
 array([[-0.0659879, 0.00671469, -0.139393, ..., 0.0239584, 0.0787109, 0.106297],
       [0.00423507, 0.000852474, -0.0026943, ..., 0.0327331, -0.10192, 0.0297759],
       [-0.110979, -0.0798683, 0.0430389, ..., 0.069609, -0.121735, -0.193447],
       ...,
       [-0.0237439, 0.0849989, -0.102894, ..., -0.0114851, -0.142924, -0.0784186],
       [0.0166122, 0.0629365, -0.07018, ..., 0.0984517, 0.0746157, 0.108983],
       [-0.0216689, -0.0133309, -0.118804, ..., -0.146019, 0.131997, 0.0289374]], dtype=float32)),
('blocks.layers.0.sa.heads.2.tril',
 array([[0.990034, 0, 0, ..., 0, 0, 0],
       [0.990034, 0.990034, 0, ..., 0, 0, 0],
       [0.990034, 0.990034, 0.990034, ..., 0, 0, 0],
       ...,
       [0.990034, 0.990034, 0.990034, ..., 0.990034, 0, 0],
       [0.990034, 0.990034, 0.990034, ..., 0.990034, 0.990034, 0],
       [0.990034, 0.990034, 0.990034, ..., 0.990034, 0.990034, 0.990034]], dtype=float32)),
('blocks.layers.0.sa.heads.3.key.weight',
 array([[-0.0584969, -0.13265, 0.120294, ..., -0.125362, 0.0214862, -0.03225],
       [-0.0775881, -0.114855, -0.043828, ..., 0.162201, -0.0185332, -0.105651],
       [0.0470195, 0.0276839, 0.0404781, ..., -0.154421, -0.0780001, -0.0075379],
       ...,
       [-0.108582, 0.295963, -0.276396, ..., 0.02364, 0.189528, -0.0207657],
       [-0.0502759, 0.00788624, -0.0770217, ..., 0.0101936, 0.0513964, 0.118844],
       [0.187144, -0.024212, 0.0204314, ..., 0.0464797, -0.0112921, -0.185106]], dtype=float32)),
('blocks.layers.0.sa.heads.3.query.weight',
 array([[-0.207683, 0.0959908, -0.0375251, ..., -0.0199141, -0.0670158, 0.0231145],
```

```
         [-0.000841613, -0.150792, -0.133074, ..., 0.146266, -0.0585914, 0.0368219],
         [0.146316, -0.15485, -0.198485, ..., -0.103056, 0.0197344, -0.149121],
         ...,
         [-0.234953, 0.106815, -0.0620763, ..., 0.131673, 0.0390895, 0.00231142],
         [-0.228003, 0.0759926, 0.127682, ..., -0.0285783, 0.0556012, 0.0849731],
         [-0.0455871, -0.162565, -0.180212, ..., 0.0129636, -0.0111393, 0.0886634]], dtype=float32)),
('blocks.layers.0.sa.heads.3.value.weight',
 array([[-0.0429183, 0.075276, 0.0875643, ..., 0.0192732, 0.0631509, 0.088815],
         [-0.106771, -0.00194325, 0.0653757, ..., 0.0142777, 0.0396832, -0.0847913],
         [-0.0511605, -0.0251607, -0.081103, ..., -0.0173308, 0.10846, 0.00430768],
         ...,
         [-0.0961996, -0.0439604, -0.0941418, ..., 0.0530029, -0.0862674, 0.0541486],
         [0.00834448, 0.0122604, -0.120694, ..., 0.180159, 0.0737529, 0.0477497],
         [0.101644, 0.1114, -0.0634289, ..., -0.11401, 0.000782103, -0.00297229]], dtype=float32)),
('blocks.layers.0.sa.heads.3.tril',
 array([[0.990034, 0, 0, ..., 0, 0, 0],
         [0.990034, 0.990034, 0, ..., 0, 0, 0],
         [0.990034, 0.990034, 0.990034, ..., 0, 0, 0],
         ...,
         [0.990034, 0.990034, 0.990034, ..., 0.990034, 0, 0],
         [0.990034, 0.990034, 0.990034, ..., 0.990034, 0.990034, 0],
         [0.990034, 0.990034, 0.990034, ..., 0.990034, 0.990034, 0.990034]], dtype=float32)),
('blocks.layers.0.sa.proj.weight',
 array([[0.0189016, -0.0224225, -0.0791916, ..., -0.103196, -0.0617523, 0.0166789],
         [-0.0858147, -0.0566555, 0.0518055, ..., 0.0127358, -0.0546344, 0.0799209],
         [0.057348, -0.0184728, -0.096285, ..., 0.066703, 0.00382683, -0.0828508],
         ...,
         [0.010785, -0.153925, -0.0841011, ..., 0.10743, -0.0976022, 0.107687],
         [0.0346688, 0.0589551, -0.0496183, ..., -0.114235, -0.101909, 0.0703862],
         [-0.0228695, 0.0652986, 0.00853055, ..., -0.124409, -0.136369, 0.0588746]], dtype=float32)),
('blocks.layers.0.sa.proj.bias',
 array([0.0453669, -0.0573561, -0.122711, ..., 0.051397, 0.104167, -0.0138329], dtype=float32)),
('blocks.layers.0.ffwd.net.layers.0.weight',
 array([[0.146932, 0.00781153, -0.0285263, ..., -0.0285435, -0.100344, 0.0864525],
         [0.0726349, -0.152488, -0.119121, ..., -0.0602606, -0.0503663, -0.173942],
         [0.0207345, 0.0385403, -0.0290862, ..., -0.190674, -0.0539099, 0.0953822],
         ...,
         [0.220585, -0.233233, 0.0209499, ..., 0.0279794, 0.181033, 0.0474235],
         [-0.260233, 0.290065, 0.0960865, ..., -0.0618864, -0.0119735, -0.0513444],
         [0.0403047, 0.0247038, 0.103679, ..., -0.053348, -0.121677, -0.210664]], dtype=float32)),
('blocks.layers.0.ffwd.net.layers.0.bias',
 array([-0.0988094, -0.0960946, -0.122396, ..., -0.0441685, -0.0895277, -0.0579924], dtype=float32)),
('blocks.layers.0.ffwd.net.layers.2.weight',
 array([[0.0320946, -0.0285338, -0.000931176, ..., 0.0201184, 0.0733887, -0.189327],
         [0.0186216, 0.0559758, -0.141304, ..., -0.063271, 0.169299, 0.0391877],
         [-0.156745, 0.0116302, -0.138103, ..., -0.016956, -0.0859499, 0.0405698],
         ...,
         [0.108988, 0.0458492, -0.0351158, ..., -0.0224554, 0.0879382, -0.0550328],
         [-0.0180455, 0.0268296, -0.147194, ..., 0.0217109, 0.0722182, -0.0501522],
         [0.0149388, -0.0825689, -0.0650666, ..., -0.0689762, -0.154858, -0.0364142]], dtype=float32)),
('blocks.layers.0.ffwd.net.layers.2.bias',
 array([0.0868792, 0.0620685, -0.00348119, ..., 0.0583432, 0.0494743, 0.0512454], dtype=float32)),
('blocks.layers.0.ln1.bias',
 array([-0.0552004, -0.047925, 0.00733104, ..., -0.037557, 0.00185288, 0.0145351], dtype=float32)),
('blocks.layers.0.ln1.weight',
 array([0.994992, 1.06808, 0.967012, ..., 0.889132, 0.972033, 0.911277], dtype=float32)),
('blocks.layers.0.ln2.bias',
 array([-0.0728743, -0.0474489, 0.00973444, ..., 0.0167013, -0.0660945, 0.0262342], dtype=float32)),
('blocks.layers.0.ln2.weight',
 array([1.12285, 1.09891, 1.02228, ..., 1.12008, 1.02993, 1.07981], dtype=float32)),
('blocks.layers.1.sa.heads.0.key.weight',
 array([[-0.0400535, -0.11265, 0.0474975, ..., -0.0340356, -0.0866461, 0.0859639],
         [-0.162278, 0.0799884, 0.0986748, ..., -0.178171, -0.0304753, -0.0526115],
         [0.0646977, 0.0591206, -0.0980352, ..., -0.0455171, -0.0354039, 0.219043],
         ...,
         [-0.0118364, -0.125774, 0.163565, ..., 0.0112293, -0.0912547, 0.0540388],
         [0.0671749, -0.0958561, 0.0237581, ..., -0.0421129, -0.0635759, -0.0624491],
         [0.0168242, 0.115556, 0.151519, ..., -0.00878156, -0.0126368, -0.0968675]], dtype=float32)),
('blocks.layers.1.sa.heads.0.query.weight',
 array([[0.0490923, 0.212021, -0.139577, ..., 0.169279, 0.00968103, 0.0756638],
         [-0.162711, 0.094784, 0.0158674, ..., -0.0773354, -0.0558774, 0.114708],
         [0.215295, 0.105867, -0.00698034, ..., 0.100298, 0.109384, 0.0287008],
         ...,
         [0.0111188, 0.161368, 0.119772, ..., -0.158011, -0.0339805, 0.143629],
         [-0.0240684, -0.0583943, 0.0291848, ..., 0.00519326, -0.085642, -0.13891],
         [0.287629, 0.021142, -0.151947, ..., -0.0719784, 0.079971, 0.150241]], dtype=float32)),
('blocks.layers.1.sa.heads.0.value.weight',
 array([[0.0604561, -0.0310873, -0.0705645, ..., 0.00451247, 0.00540587, -0.0608642],
         [0.0805266, 0.156747, 0.132474, ..., 0.00514947, -0.131971, -0.152045],
         [-0.0985657, 0.0275796, 0.0470056, ..., 0.100793, 0.0755097, 0.0373311],
         ...,
         [0.0385057, 0.0274103, -0.109391, ..., 0.0742823, 0.114556, -0.020403],
         [0.0216323, 0.148229, -0.00474409, ..., -0.11581, -0.00920013, -0.0195847],
         [-0.0110111, -0.155175, -0.0739493, ..., -0.118214, 0.137258, 0.0870445]], dtype=float32)),
('blocks.layers.1.sa.heads.0.tril',
 array([[0.990034, 0, 0, ..., 0, 0, 0],
         [0.990034, 0.990034, 0, ..., 0, 0, 0],
         [0.990034, 0.990034, 0.990034, ..., 0, 0, 0],
         ...,
         [0.990034, 0.990034, 0.990034, ..., 0.990034, 0, 0],
         [0.990034, 0.990034, 0.990034, ..., 0.990034, 0.990034, 0],
```

```
        [0.990034, 0.990034, 0.990034, ..., 0.990034, 0.990034, 0.990034]], dtype=float32)),
('blocks.layers.1.sa.heads.1.key.weight',
 array([[-0.0137523, 0.0554814, -0.027609, ..., 0.0932836, 0.000499967, -0.188066],
        [-0.067945, 0.149034, 0.0345904, ..., 0.106345, -0.287478, -0.0964092],
        [-0.05186, 0.0962455, 0.0263724, ..., 0.11767, -0.100935, -0.100331],
        ...,
        [0.158729, -0.0704473, -0.176647, ..., 0.0101056, 0.161039, -0.0345473],
        [0.0865949, 0.113211, 0.0662116, ..., 0.0817748, -0.0311247, 0.0337568],
        [0.181555, -0.0595193, -0.0189953, ..., 0.0330592, 0.144205, 0.00355098]], dtype=float32)),
('blocks.layers.1.sa.heads.1.query.weight',
 array([[0.0543543, 0.0520112, -0.128679, ..., 0.0331313, 0.0132594, -0.0341528],
        [0.0430379, -0.105778, 0.0714036, ..., 0.166159, 0.0568783, 0.122565],
        [-0.0697753, -0.0353642, 0.194245, ..., -2.40444e-05, 0.0649022, 0.118377],
        ...,
        [0.116212, 0.042656, 0.0137871, ..., 0.0402008, -0.00857886, -0.110084],
        [-0.212346, 0.0664015, 0.0664784, ..., -0.0910784, -0.0174408, -0.114355],
        [-0.0787514, -0.192375, 0.15852, ..., -0.0584974, 0.070565, 0.115919]], dtype=float32)),
('blocks.layers.1.sa.heads.1.value.weight',
 array([[0.122317, 0.0523012, 0.0189365, ..., 0.029964, -0.103333, 0.0252909],
        [-0.0838236, -0.147123, -0.0284879, ..., -0.0247119, -0.0948232, -0.0539475],
        [-0.134917, -0.0978696, -0.0478646, ..., 0.0607934, -0.0212352, -0.0838142],
        ...,
        [-0.0525428, -0.0466305, 0.118392, ..., -0.0904291, -0.0661673, 0.00506546],
        [0.0303712, -0.0392764, 0.0494058, ..., 0.0645968, -0.110808, -0.0369371],
        [0.205401, -0.0578569, -0.0942512, ..., 0.0506146, -0.0815157, 0.0153096]], dtype=float32)),
('blocks.layers.1.sa.heads.1.tril',
 array([[0.990034, 0, 0, ..., 0, 0, 0],
        [0.990034, 0.990034, 0, ..., 0, 0, 0],
        [0.990034, 0.990034, 0.990034, ..., 0, 0, 0],
        ...,
        [0.990034, 0.990034, 0.990034, ..., 0.990034, 0, 0],
        [0.990034, 0.990034, 0.990034, ..., 0.990034, 0.990034, 0],
        [0.990034, 0.990034, 0.990034, ..., 0.990034, 0.990034, 0.990034]], dtype=float32)),
('blocks.layers.1.sa.heads.2.key.weight',
 array([[0.202434, 0.00387022, 0.0339389, ..., 0.0282088, -0.0209039, -0.00948913],
        [-0.148163, 0.0334721, 0.155887, ..., -0.0787436, 0.0298006, 0.0469819],
        [0.0358729, 0.107184, 0.128457, ..., 0.0198307, -0.041543, -0.123559],
        ...,
        [0.0533823, -0.158353, -0.105448, ..., 0.118329, -0.122224, 0.0617975],
        [-0.071202, 0.162084, 0.0995896, ..., 0.0415497, 0.101227, -0.0965683],
        [0.0707414, 0.194437, 0.0444762, ..., 0.0870921, -0.123649, 0.0953524]], dtype=float32)),
('blocks.layers.1.sa.heads.2.query.weight',
 array([[0.111152, 0.0247546, -0.120694, ..., -0.0252794, -0.0993965, 0.110962],
        [-0.0146107, -0.0732751, 0.174097, ..., 0.0361764, 0.0299604, -0.107343],
        [-0.00868426, 0.0703274, 0.120223, ..., 0.0147634, -0.0741594, 0.0557424],
        ...,
        [-0.0938769, -0.0908229, -0.0523405, ..., 0.164684, 0.149125, 0.11362],
        [-0.0875885, -0.0916771, -0.050898, ..., 0.128296, -0.0105255, 0.12558],
        [-0.133319, 0.126561, -0.055477, ..., -0.082277, 0.0166835, -0.160708]], dtype=float32)),
('blocks.layers.1.sa.heads.2.value.weight',
 array([[-0.0880173, -0.00907329, 0.0715499, ..., -0.0643163, -0.120099, 0.0718239],
        [0.0661108, -0.100488, 0.0366348, ..., 0.0367812, 0.0798968, 0.118994],
        [-0.124216, 0.0636774, 0.00208044, ..., -0.0274996, 0.055151, 0.12131],
        ...,
        [-0.0709319, -0.0124059, -0.0052829, ..., -0.0918196, -0.0533389, -0.0451327],
        [0.169826, 0.037426, -0.133561, ..., -0.0395437, 0.102663, -0.0286632],
        [0.00623682, 0.0692017, 0.00499522, ..., -0.103557, -0.0870507, -0.057297]], dtype=float32)),
('blocks.layers.1.sa.heads.2.tril',
 array([[0.990034, 0, 0, ..., 0, 0, 0],
        [0.990034, 0.990034, 0, ..., 0, 0, 0],
        [0.990034, 0.990034, 0.990034, ..., 0, 0, 0],
        ...,
        [0.990034, 0.990034, 0.990034, ..., 0.990034, 0, 0],
        [0.990034, 0.990034, 0.990034, ..., 0.990034, 0.990034, 0],
        [0.990034, 0.990034, 0.990034, ..., 0.990034, 0.990034, 0.990034]], dtype=float32)),
('blocks.layers.1.sa.heads.3.key.weight',
 array([[0.098463, 0.223371, 0.0703019, ..., -0.0746497, 0.238476, -0.241393],
        [-0.0232655, 0.0366935, -0.204814, ..., 0.0285997, 0.190207, -0.14111],
        [0.270968, 0.0571484, -0.1226, ..., -0.0820532, 0.0392901, 0.115424],
        ...,
        [0.12329, 0.125657, 0.212812, ..., -0.0974634, 0.00380679, -0.0254332],
        [-0.144095, -0.177051, 0.242309, ..., -0.0247473, -0.375839, 0.0943837],
        [-0.131005, -0.288595, -0.0905633, ..., 0.00847323, -0.218605, 0.243874]], dtype=float32)),
('blocks.layers.1.sa.heads.3.query.weight',
 array([[0.0506066, 0.0518229, 0.133543, ..., -0.0120137, 0.00746959, 0.0589005],
        [0.180368, -0.046641, -0.0582548, ..., 0.0895432, -0.178726, -0.0416296],
        [0.00637355, 0.0183431, 0.0877594, ..., -0.0497577, 0.0190254, -0.162235],
        ...,
        [-0.108821, -0.00189329, -0.140921, ..., -0.166193, -0.0880122, -0.216735],
        [-0.108933, 0.0217968, -0.10322, ..., 0.100829, -0.055628, 0.0585639],
        [-0.221925, -0.185992, 0.103029, ..., -0.0465214, 0.108189, 0.16278]], dtype=float32)),
('blocks.layers.1.sa.heads.3.value.weight',
 array([[0.013671, 0.0635034, 0.0963904, ..., 0.0900481, 0.0340098, -0.1882],
        [-0.0715059, 0.0748685, -0.056611, ..., -0.00250101, 0.0214303, -0.207396],
        [0.0182891, -0.00179538, 0.0336794, ..., -0.0544777, 0.133857, 0.0673014],
        ...,
        [-0.115938, 0.0211694, 0.0202091, ..., 0.0582344, 0.10555, -0.0748243],
        [-0.0975486, -0.112571, 0.0180214, ..., -0.070878, -0.0266714, 0.0615814],
        [-0.111656, 0.112896, -0.0608083, ..., -0.0778921, 0.0476918, 0.142316]], dtype=float32)),
('blocks.layers.1.sa.heads.3.tril',
 array([[0.990034, 0, 0, ..., 0, 0, 0],
```

```
        [0.990034, 0.990034, 0, ..., 0, 0, 0],
        [0.990034, 0.990034, 0.990034, ..., 0, 0, 0],
        ...,
        [0.990034, 0.990034, 0.990034, ..., 0.990034, 0, 0],
        [0.990034, 0.990034, 0.990034, ..., 0.990034, 0.990034, 0],
        [0.990034, 0.990034, 0.990034, ..., 0.990034, 0.990034, 0.990034]], dtype=float32)),
('blocks.layers.1.sa.proj.weight',
 array([[-0.0553812, 0.00582115, 0.226637, ..., 0.00273342, 0.0427934, -0.0821208],
        [-0.0357759, 0.0733274, -0.171754, ..., 0.0651126, 0.0790596, -0.0307711],
        [0.174329, 0.0560628, 0.100226, ..., -0.115152, -0.157284, 0.00735914],
        ...,
        [0.111938, -0.10057, -0.120091, ..., -0.0935056, 0.0542092, 0.0637828],
        [0.0562655, 0.0438043, 0.0440522, ..., -0.0753328, 0.0574559, -0.035853],
        [0.0464259, 0.0748157, -0.105638, ..., -0.0464397, -0.0784602, 0.0655274]], dtype=float32)),
('blocks.layers.1.sa.proj.bias',
 array([-0.107806, 0.0583784, -0.0415804, ..., 0.0489442, -0.0070709, 0.0951004], dtype=float32)),
('blocks.layers.1.ffwd.net.layers.0.weight',
 array([[0.0391631, -0.0665326, -0.114141, ..., -0.00590902, -0.136226, 0.155681],
        [0.0300198, -0.185446, 0.0316573, ..., 0.0174752, -0.13945, -0.0748946],
        [0.0320904, -0.104872, 0.0588656, ..., -0.0284272, -0.141577, -0.207504],
        ...,
        [-0.105557, -0.0383901, -0.0334372, ..., -0.0335263, 0.0403357, -0.0885975],
        [-0.0539022, -0.031476, 0.0710668, ..., -0.0399973, 0.120447, -0.138385],
        [-0.165615, 0.0874978, 0.165415, ..., -0.132703, -0.0246205, 0.127328]], dtype=float32)),
('blocks.layers.1.ffwd.net.layers.0.bias',
 array([0.0603768, 0.00547501, 0.0533781, ..., 0.010105, 0.0367536, -0.0349384], dtype=float32)),
('blocks.layers.1.ffwd.net.layers.2.weight',
 array([[-0.0691776, -0.0152331, 0.231139, ..., 0.0683929, -0.0984744, 0.0513661],
        [0.0340309, -0.0151894, -0.156243, ..., 0.0266403, -0.0814764, 0.0626006],
        [-0.0619185, -0.108422, 0.244511, ..., -0.120492, 0.0562261, -0.0396828],
        ...,
        [0.0488872, 0.0377593, -0.303864, ..., 0.0268625, -0.0242797, 0.0451677],
        [0.0111531, -0.0203534, -0.166248, ..., -0.00708144, -0.0280786, -0.0156141],
        [0.076095, 0.112199, -0.125662, ..., -0.0357408, -0.0151737, 0.0276942]], dtype=float32)),
('blocks.layers.1.ffwd.net.layers.2.bias',
 array([-0.000527298, -0.0343769, -0.04777, ..., -0.0194451, -0.0134919, 0.0189715], dtype=float32)),
('blocks.layers.1.ln1.bias',
 array([-0.0110889, -0.0622378, 0.0974116, ..., -0.0956701, -0.0206666, 0.0278902], dtype=float32)),
('blocks.layers.1.ln1.weight',
 array([1.04359, 0.991135, 0.828751, ..., 0.834305, 0.988476, 0.981301], dtype=float32)),
('blocks.layers.1.ln2.bias',
 array([0.056183, 0.0579904, 0.0324644, ..., 0.0219587, 0.00872758, 0.0167562], dtype=float32)),
('blocks.layers.1.ln2.weight',
 array([1.05358, 0.95364, 0.893239, ..., 0.923622, 1.06274, 0.964229], dtype=float32)),
('blocks.layers.2.sa.heads.0.key.weight',
 array([[-0.187597, -0.0986563, 0.0403632, ..., 0.101032, -0.0449269, -0.14489],
        [-0.0467735, 0.148238, 0.0860622, ..., -0.0608474, -0.159591, -0.158423],
        [0.199134, -0.0469497, -0.0283443, ..., 0.142015, 0.0136401, 0.00903703],
        ...,
        [-0.262374, 0.0397383, 0.0176563, ..., 0.0728063, 0.0787197, -0.0999131],
        [0.278207, -0.0169608, -0.169198, ..., 0.162988, -0.000368108, 0.206477],
        [-0.180665, 0.0350572, 0.0107607, ..., 0.0922623, 0.151175, 0.205289]], dtype=float32)),
('blocks.layers.2.sa.heads.0.query.weight',
 array([[0.114198, -0.137205, -0.040768, ..., 0.197988, 0.0970206, 0.0900465],
        [0.0166445, 0.215591, 0.0722866, ..., -0.161084, -0.166928, -0.0707737],
        [-0.0269633, -0.0964022, -0.0114589, ..., 0.068481, 0.202701, 0.0709616],
        ...,
        [-0.0216387, -0.118522, 0.104724, ..., 0.00600445, -0.0560659, -0.122777],
        [-0.0383753, -0.0246657, -0.15453, ..., 0.0507093, 0.197482, -0.0285432],
        [-0.0122249, -0.00303289, 0.128109, ..., 0.0300376, -0.0583705, -0.0238033]], dtype=float32)),
('blocks.layers.2.sa.heads.0.value.weight',
 array([[-0.00832111, -0.0292508, 0.080084, ..., -0.079498, -0.0154344, 0.00792454],
        [-0.0333173, -0.0315659, 0.0330832, ..., -0.0319158, -0.0387865, 0.000849065],
        [0.0828489, 0.0499791, 0.0536267, ..., -0.0385391, 0.0583446, 0.0906799],
        ...,
        [-0.120196, -0.102519, -0.0212586, ..., 0.0811519, -0.0855079, -0.0118441],
        [0.0475961, -0.0369533, 0.0644993, ..., -0.113041, 0.100361, -0.0936346],
        [0.070415, -0.0230663, -0.0420532, ..., -0.14764, 0.0432007, 0.117954]], dtype=float32)),
('blocks.layers.2.sa.heads.0.tril',
 array([[0.990034, 0, 0, ..., 0, 0, 0],
        [0.990034, 0.990034, 0, ..., 0, 0, 0],
        [0.990034, 0.990034, 0.990034, ..., 0, 0, 0],
        ...,
        [0.990034, 0.990034, 0.990034, ..., 0.990034, 0, 0],
        [0.990034, 0.990034, 0.990034, ..., 0.990034, 0.990034, 0],
        [0.990034, 0.990034, 0.990034, ..., 0.990034, 0.990034, 0.990034]], dtype=float32)),
('blocks.layers.2.sa.heads.1.key.weight',
 array([[0.153512, -0.120719, 0.0071866, ..., 0.00489977, 0.183641, 0.078357],
        [-0.0222693, 0.152045, 0.184106, ..., -0.140078, -0.0964828, -0.0879873],
        [0.235911, -0.00366993, 0.0355738, ..., -0.0328392, -0.0747224, -0.09845],
        ...,
        [0.0919853, -0.0671492, -0.128853, ..., 0.024661, 0.260692, 0.15124],
        [-0.0872199, 0.188349, 0.175553, ..., -0.0972703, -0.134144, -0.112999],
        [-0.00951975, 0.0945436, 0.125063, ..., 0.0807255, 0.0109544, 0.148854]], dtype=float32)),
('blocks.layers.2.sa.heads.1.query.weight',
 array([[0.15023, -0.0110593, -0.23319, ..., -0.0347369, 0.047355, 0.0456066],
        [-0.0292373, -0.0175737, 0.00750622, ..., 0.0153322, 0.179966, -0.0896988],
        [0.0645168, 0.208392, -0.138228, ..., 0.127202, -0.0705027, 0.0180287],
        ...,
        [-0.0877973, 0.137082, -0.187244, ..., 0.0238796, 0.0674456, 0.0577891],
        [-0.0267793, -0.0334146, 0.0579558, ..., -0.0850635, 0.0274475, -0.12801],
```

```
        [-0.0614221, 0.100886, -0.182827, ..., 0.0666869, 0.0465052, 0.13984]], dtype=float32)),
('blocks.layers.2.sa.heads.1.value.weight',
 array([[0.0159674, -0.0299807, -0.065026, ..., -0.0848531, -0.0764302, -0.0606403],
        [-0.0704923, 0.0270815, -0.0747891, ..., 0.127992, -0.0499965, 0.0442049],
        [0.122013, 0.00234251, -0.0767631, ..., 0.0242639, -0.0319893, -0.0258769],
        ...,
        [0.122436, -0.0253894, 0.00840425, ..., 0.100116, 0.0686485, 0.0501273],
        [0.179053, -0.136492, -0.0209334, ..., -0.00437786, 0.0817625, 0.00408246],
        [0.089249, -0.0586061, 0.0377334, ..., 0.164487, 0.126189, 0.0606503]], dtype=float32)),
('blocks.layers.2.sa.heads.1.tril',
 array([[0.990034, 0, 0, ..., 0, 0, 0],
        [0.990034, 0.990034, 0, ..., 0, 0, 0],
        [0.990034, 0.990034, 0.990034, ..., 0, 0, 0],
        ...,
        [0.990034, 0.990034, 0.990034, ..., 0.990034, 0, 0],
        [0.990034, 0.990034, 0.990034, ..., 0.990034, 0.990034, 0],
        [0.990034, 0.990034, 0.990034, ..., 0.990034, 0.990034, 0.990034]], dtype=float32)),
('blocks.layers.2.sa.heads.2.key.weight',
 array([[-0.0522512, -0.11445, 0.00756314, ..., 0.167918, 0.145717, 0.185841],
        [0.134313, 0.169497, -0.081236, ..., -0.0404235, 0.278349, 0.0575469],
        [0.0854794, 0.0464511, -0.00768706, ..., -0.0468738, 0.220962, -0.101814],
        ...,
        [-0.0743536, 0.00899745, -0.163147, ..., 0.0160496, 0.0281667, -0.15461],
        [-0.0178432, 0.0104654, -0.0453272, ..., -0.00690575, 0.113823, 0.162981],
        [0.0268709, -0.237201, 0.0203512, ..., 0.0721035, -0.0397057, 0.219867]], dtype=float32)),
('blocks.layers.2.sa.heads.2.query.weight',
 array([[-0.0165754, 0.0627241, 0.0918853, ..., -0.254629, -0.150786, -0.13998],
        [0.0293133, 0.195299, -0.0958444, ..., -0.104332, 0.0386914, 0.127928],
        [0.116388, 0.0133537, -0.0591793, ..., -0.0560022, -0.019381, -0.0693941],
        ...,
        [-0.00919587, 0.134186, 0.0150581, ..., -0.0487431, 0.131776, -0.0446411],
        [-0.0728134, 0.108586, 0.0503679, ..., 0.0174462, -0.170092, -0.0290942],
        [-0.122376, 0.0216968, 0.24499, ..., -0.146752, -0.178811, -0.0609676]], dtype=float32)),
('blocks.layers.2.sa.heads.2.value.weight',
 array([[0.000267624, -0.113233, 0.0773238, ..., 0.0233368, -0.0237623, -0.0607592],
        [0.149006, 0.0117226, 0.101793, ..., -0.0413513, 0.155505, -0.0244487],
        [-0.130527, 0.100845, 0.132996, ..., -0.0188726, -0.0845216, 0.0165726],
        ...,
        [0.0334835, -0.0114566, 0.0858786, ..., 0.0357424, 0.088296, 0.0838804],
        [-0.00973454, 0.0252932, 0.0745305, ..., 0.0850739, -0.159339, -0.0270227],
        [0.0375007, 0.0141815, 0.0691071, ..., -0.0332664, -0.0472939, 0.0537389]], dtype=float32)),
('blocks.layers.2.sa.heads.2.tril',
 array([[0.990034, 0, 0, ..., 0, 0, 0],
        [0.990034, 0.990034, 0, ..., 0, 0, 0],
        [0.990034, 0.990034, 0.990034, ..., 0, 0, 0],
        ...,
        [0.990034, 0.990034, 0.990034, ..., 0.990034, 0, 0],
        [0.990034, 0.990034, 0.990034, ..., 0.990034, 0.990034, 0],
        [0.990034, 0.990034, 0.990034, ..., 0.990034, 0.990034, 0.990034]], dtype=float32)),
('blocks.layers.2.sa.heads.3.key.weight',
 array([[-0.208302, -0.051894, -0.0251771, ..., 0.0762505, 0.111724, -0.124818],
        [-0.192069, -0.149641, -0.00531491, ..., -0.0265566, -0.0451469, 0.0331848],
        [0.234647, 0.0217227, -0.0990483, ..., 0.0253714, -0.00975956, 0.0198607],
        ...,
        [-0.168347, -0.0605731, 0.133507, ..., -0.217694, 6.28118e-05, -0.145785],
        [-0.0281367, -0.118505, -0.0946721, ..., 0.0123714, 0.0144717, -0.0504547],
        [0.0997702, 0.167144, 0.187869, ..., -0.0146896, -0.0696699, -0.130507]], dtype=float32)),
('blocks.layers.2.sa.heads.3.query.weight',
 array([[-0.00455695, -0.160598, 0.105529, ..., -0.0754108, 0.00519012, 0.141972],
        [0.0433742, -0.00462315, -0.0481037, ..., -0.136014, 0.0592697, 0.180117],
        [-0.0462135, -0.138555, 0.129989, ..., -0.0689285, -0.0655558, 0.0574138],
        ...,
        [-0.0796801, 0.0368843, 0.0925858, ..., 0.0787214, 0.0388648, 0.11195],
        [0.0837566, 0.23488, 0.127388, ..., -0.184899, -0.123292, -0.0310886],
        [-0.097453, 0.118593, -0.0780089, ..., 0.114343, -0.0136758, -0.0838988]], dtype=float32)),
('blocks.layers.2.sa.heads.3.value.weight',
 array([[-0.0477094, 0.0509297, -0.0600367, ..., -0.0998949, 0.0660393, 0.0919744],
        [-0.102256, 0.116286, -0.00266547, ..., 0.0476366, 0.0225585, -0.167419],
        [-0.0824199, -0.110024, -0.0170612, ..., 0.0270502, 0.12063, 0.0778325],
        ...,
        [0.0715327, 0.00996926, 0.0464942, ..., -0.0100232, 0.00397722, 0.074436],
        [-0.133333, 0.0755601, -0.06233, ..., -0.0172725, -0.102178, 0.0924894],
        [-0.113591, -0.0631053, 0.0878908, ..., -0.0502102, -0.0109144, -0.0795243]], dtype=float32)),
('blocks.layers.2.sa.heads.3.tril',
 array([[0.990034, 0, 0, ..., 0, 0, 0],
        [0.990034, 0.990034, 0, ..., 0, 0, 0],
        [0.990034, 0.990034, 0.990034, ..., 0, 0, 0],
        ...,
        [0.990034, 0.990034, 0.990034, ..., 0.990034, 0, 0],
        [0.990034, 0.990034, 0.990034, ..., 0.990034, 0.990034, 0],
        [0.990034, 0.990034, 0.990034, ..., 0.990034, 0.990034, 0.990034]], dtype=float32)),
('blocks.layers.2.sa.proj.weight',
 array([[-0.0796633, 0.0950704, -0.130945, ..., -0.0530155, -0.00961084, -0.00805925],
        [0.0461319, 0.0964148, 0.103794, ..., 0.0721841, 0.047567, -0.00545223],
        [0.0854583, -0.0815579, 0.00534617, ..., 0.0613969, 0.0580245, 0.0658779],
        ...,
        [-0.0217461, 0.0525998, -0.104968, ..., 0.0313424, -0.0740164, 0.050091],
        [0.0620569, 0.0795206, 0.132323, ..., 0.110241, 0.00449282, -0.0593857],
        [0.0688009, 0.0570101, -0.112392, ..., -0.169973, -0.0855229, 0.0242827]], dtype=float32)),
('blocks.layers.2.sa.proj.bias',
 array([-0.0985098, -0.0340715, 0.0180208, ..., 0.118773, 0.0841364, 0.116754], dtype=float32)),
```

```
('blocks.layers.2.ffwd.net.layers.0.weight',
 array([[0.0275276, -0.177702, -0.0102963, ..., -0.00529089, -0.0445838, 0.0405454],
        [0.155641, -0.046852, 0.171314, ..., -0.040416, 0.126528, 0.00760775],
        [0.120315, 0.018707, 0.0025548, ..., -0.0256544, -0.053679, -0.0704607],
        ...,
        [-0.110392, -0.099334, 0.0109057, ..., 0.0313674, 0.00189814, -0.0180012],
        [0.243012, 0.0477812, -0.0196961, ..., -0.0829057, 0.0197937, 0.068832],
        [-0.0784591, -0.1875, -0.00223208, ..., -0.166161, -0.0885785, 0.0464497]], dtype=float32)),
('blocks.layers.2.ffwd.net.layers.0.bias',
 array([-0.0484412, -0.0960981, -0.0549224, ..., -0.0370725, -0.0341577, -0.0177792], dtype=float32)),
('blocks.layers.2.ffwd.net.layers.2.weight',
 array([[-0.0723016, -0.0857352, 0.165463, ..., 0.14125, -0.0461706, -0.0119633],
        [0.0720734, 0.0471059, -0.151961, ..., 0.0796267, 0.0566066, -0.0223233],
        [-0.0242366, -0.0501359, -0.093549, ..., -0.0275947, -0.016835, 0.0860876],
        ...,
        [-0.0103615, -0.00150952, -0.0184681, ..., 0.0238518, -0.072515, -0.103023],
        [-0.0293546, 0.0215113, 0.0526912, ..., -0.0600884, -0.142138, 0.0280916],
        [0.0159591, 0.11823, 0.137633, ..., -0.0141856, 0.0290131, 0.173439]], dtype=float32)),
('blocks.layers.2.ffwd.net.layers.2.bias',
 array([0.0611714, -0.0426206, -0.0169752, ..., 0.0270404, -0.0464103, 0.0337226], dtype=float32)),
('blocks.layers.2.ln1.bias',
 array([-0.0145753, -0.00992254, 0.101071, ..., -0.0445146, -0.0855191, -0.0353889], dtype=float32)),
('blocks.layers.2.ln1.weight',
 array([1.09515, 0.957278, 0.914665, ..., 0.900869, 0.940505, 0.959067], dtype=float32)),
('blocks.layers.2.ln2.bias',
 array([0.0261532, 0.0353615, -0.0740954, ..., 0.0259853, 0.047084, 0.0360262], dtype=float32)),
('blocks.layers.2.ln2.weight',
 array([0.981391, 1.0432, 1.01904, ..., 0.932926, 1.00655, 0.990262], dtype=float32)),
('blocks.layers.3.sa.heads.0.key.weight',
 array([[-0.0176635, 0.228202, -0.193654, ..., 0.111313, 0.0555834, 0.171299],
        [-0.167075, 0.0131204, 0.289895, ..., -0.0659993, -0.0344124, 0.169704],
        [-0.192158, -0.0454054, -0.122172, ..., 0.0369429, -0.00619299, -0.0412868],
        ...,
        [0.0815338, 0.130057, -0.176359, ..., 0.101748, -0.0251637, 0.0336932],
        [-0.0118426, 0.0187528, 0.0712743, ..., -0.00897303, 0.141014, -0.0303928],
        [0.0953686, -0.111199, -0.0703709, ..., 0.0489961, 0.0901838, -0.30264]], dtype=float32)),
('blocks.layers.3.sa.heads.0.query.weight',
 array([[0.207605, 0.02653, -0.0882732, ..., -0.141643, -0.0990606, -0.025319],
        [0.0421186, -0.022183, 0.102823, ..., 0.106313, 0.064266, -0.153607],
        [-0.283804, 0.102545, 0.119894, ..., 0.131185, 0.266548, 0.0938521],
        ...,
        [0.086717, -0.111233, -0.086993, ..., -0.062974, -0.202026, 0.103741],
        [-0.0608554, 0.0988972, -0.0264032, ..., 0.117248, 0.136821, 0.144836],
        [-0.0965798, 0.222478, -0.0959651, ..., -0.00353334, 0.108194, 0.0839873]], dtype=float32)),
('blocks.layers.3.sa.heads.0.value.weight',
 array([[0.193452, 0.00515808, 0.0938609, ..., -0.0229018, 0.0147691, 0.0198202],
        [0.0855011, -0.149968, 0.09149, ..., -0.063795, 0.0275119, -0.100528],
        [-0.148784, 0.0878136, -0.0262331, ..., -0.131316, 0.0247462, -0.0369592],
        ...,
        [0.0200214, -0.0569506, 0.148757, ..., -0.00280587, 0.165911, 0.0015306],
        [0.118129, 0.00452062, -0.103022, ..., -0.00156841, 0.0105379, 0.0204095],
        [0.199042, -0.169988, 0.0391915, ..., 0.131052, -0.0794245, -0.121337]], dtype=float32)),
('blocks.layers.3.sa.heads.0.tril',
 array([[0.990034, 0, 0, ..., 0, 0, 0],
        [0.990034, 0.990034, 0, ..., 0, 0, 0],
        [0.990034, 0.990034, 0.990034, ..., 0, 0, 0],
        ...,
        [0.990034, 0.990034, 0.990034, ..., 0.990034, 0, 0],
        [0.990034, 0.990034, 0.990034, ..., 0.990034, 0.990034, 0],
        [0.990034, 0.990034, 0.990034, ..., 0.990034, 0.990034, 0.990034]], dtype=float32)),
('blocks.layers.3.sa.heads.1.key.weight',
 array([[0.12117, 0.0149442, -0.0787572, ..., -0.0427371, 0.0413996, -0.111196],
        [-0.103431, 0.135002, -0.234384, ..., -0.0204299, 0.130643, 0.112583],
        [-0.0745859, 0.0266555, 0.0848426, ..., 0.131141, -0.0745413, -0.0841877],
        ...,
        [0.028785, -0.0742018, 0.0239781, ..., -0.0234777, 0.0930419, 0.012102],
        [0.0711586, -0.207466, 0.0289238, ..., -0.00900671, -0.0474684, -0.162263],
        [0.0832313, 0.187553, -0.262938, ..., -0.0285838, -0.162486, 0.129751]], dtype=float32)),
('blocks.layers.3.sa.heads.1.query.weight',
 array([[-0.0350894, -0.077688, -0.126188, ..., 0.0345096, -0.144583, 0.102251],
        [-0.0257399, 0.0388406, -0.185367, ..., 0.0040016, 0.0753518, -0.0301342],
        [0.0455586, 0.0768657, 0.0507159, ..., 0.114637, 0.0605263, -0.0431879],
        ...,
        [-0.120495, -0.140277, 0.0858393, ..., 0.00390999, 0.146412, -0.0783367],
        [0.0237606, -0.125073, 0.00227502, ..., 0.00147299, -0.00288017, -0.0757856],
        [-0.209331, 0.197968, -0.0362723, ..., -0.037409, 0.109251, 0.0303097]], dtype=float32)),
('blocks.layers.3.sa.heads.1.value.weight',
 array([[-0.0343663, 0.00263952, -0.0273027, ..., 0.0768747, -0.0243774, -0.139655],
        [0.0258405, 0.00760198, 0.0488757, ..., -0.0676936, -0.201655, 0.0525828],
        [0.0214535, 0.053808, -0.118163, ..., 0.0890145, -0.039961, 0.095494],
        ...,
        [-0.0560705, -0.103867, -0.0496419, ..., 0.137277, -0.0480695, 0.109898],
        [0.0786566, 0.0364906, 0.0805726, ..., 0.0604027, -0.0951187, -0.0187166],
        [0.020259, -0.0536216, -0.0801038, ..., -0.0303378, -0.0783241, 0.0306764]], dtype=float32)),
('blocks.layers.3.sa.heads.1.tril',
 array([[0.990034, 0, 0, ..., 0, 0, 0],
        [0.990034, 0.990034, 0, ..., 0, 0, 0],
        [0.990034, 0.990034, 0.990034, ..., 0, 0, 0],
        ...,
        [0.990034, 0.990034, 0.990034, ..., 0.990034, 0, 0],
        [0.990034, 0.990034, 0.990034, ..., 0.990034, 0.990034, 0],
```

```
        [0.990034, 0.990034, 0.990034, ..., 0.990034, 0.990034, 0.990034]], dtype=float32)),
('blocks.layers.3.sa.heads.2.key.weight',
 array([[-0.0115221, 0.144307, -0.214265, ..., -0.0124423, 0.120685, -0.160133],
        [0.0247436, 0.139601, -0.026324, ..., 0.0200322, 0.258198, 0.0709675],
        [-0.0748027, -0.0922402, 0.0112884, ..., 0.0331468, -0.152365, 0.278636],
        ...,
        [0.140582, 0.0101029, -0.135973, ..., 0.0593902, -0.0574604, 0.0569314],
        [0.0707739, 0.0337702, 0.105921, ..., -0.0343455, 0.0542912, 0.0173908],
        [0.126128, 0.0414577, 0.0345816, ..., 0.0689424, -0.11557, -0.045795]], dtype=float32)),
('blocks.layers.3.sa.heads.2.query.weight',
 array([[-0.0731734, -0.118741, 0.0239597, ..., -0.107721, 0.00610271, 0.0704958],
        [-0.230862, -0.145681, -0.136921, ..., -0.0173255, -0.0347745, -0.0381323],
        [-0.0731044, -0.118401, -0.0597044, ..., -0.0601359, -0.0480029, -0.0065954],
        ...,
        [0.118389, -0.0695624, 0.0754661, ..., -0.137716, 0.0958593, 0.068771],
        [-0.0755171, 0.0217767, -0.00689107, ..., 0.0677599, -0.0589825, -0.25491],
        [0.0982634, 0.0170704, 0.0772643, ..., -0.0424041, 0.0765525, -0.000922494]], dtype=float32)),
('blocks.layers.3.sa.heads.2.value.weight',
 array([[-0.069164, 0.0750477, -0.0534379, ..., 0.0572123, 0.0241343, -0.0783943],
        [-0.0347451, 0.0690448, -0.124451, ..., 0.0308588, -0.0501579, -0.0720494],
        [0.166908, -0.0911132, 0.116853, ..., -0.0779368, 0.0680362, -0.0180819],
        ...,
        [0.114187, -0.0106381, 0.0846519, ..., 0.00794327, -0.157871, 0.0209198],
        [0.0471285, -0.0631039, 0.0503799, ..., 0.0076953, -0.0587598, 0.0280318],
        [-0.16792, 0.0488084, 0.00904168, ..., -0.0932744, 0.0571771, 0.0974373]], dtype=float32)),
('blocks.layers.3.sa.heads.2.tril',
 array([[0.990034, 0, 0, ..., 0, 0, 0],
        [0.990034, 0.990034, 0, ..., 0, 0, 0],
        [0.990034, 0.990034, 0.990034, ..., 0, 0, 0],
        ...,
        [0.990034, 0.990034, 0.990034, ..., 0.990034, 0, 0],
        [0.990034, 0.990034, 0.990034, ..., 0.990034, 0.990034, 0],
        [0.990034, 0.990034, 0.990034, ..., 0.990034, 0.990034, 0.990034]], dtype=float32)),
('blocks.layers.3.sa.heads.3.key.weight',
 array([[-0.0215196, -0.187364, 0.0219175, ..., -0.00983379, 0.170564, -0.28362],
        [0.0378501, 0.0807256, -0.119765, ..., 0.119321, -0.161299, -0.0578262],
        [-0.0492165, -0.0156067, 0.0435092, ..., -0.0453536, -0.214215, -0.0602221],
        ...,
        [-0.0591777, -0.0865342, -0.148822, ..., 0.100837, 0.249717, 0.109423],
        [0.12606, -0.134655, 0.018069, ..., 0.0537716, 0.0863532, -0.0206452],
        [-0.0950247, 0.110011, 0.0551485, ..., -0.0583815, -0.173406, 0.0492085]], dtype=float32)),
('blocks.layers.3.sa.heads.3.query.weight',
 array([[0.0524355, 0.066362, -0.00344221, ..., -0.17558, -0.0766167, 0.0082174],
        [-0.064071, -0.108571, 0.0267517, ..., 0.0653989, 0.140506, 0.0702281],
        [0.0389898, 0.019327, -0.06997, ..., -0.0276925, -0.175018, -0.0999239],
        ...,
        [-0.0480277, -0.120686, 0.029525, ..., 0.0738712, -0.100609, 0.0962115],
        [0.139348, 0.0340478, -0.0196409, ..., 0.0227176, -0.274583, -0.0452162],
        [0.131753, 0.0948592, 0.146556, ..., 0.0458581, 0.00758573, -0.0170579]], dtype=float32)),
('blocks.layers.3.sa.heads.3.value.weight',
 array([[-0.132125, 0.167599, -0.0928647, ..., -0.0409796, 0.00145021, 0.199332],
        [-0.148623, -0.109367, -0.0803037, ..., 0.0198569, -0.0742917, 0.0348374],
        [-0.0884751, 0.115752, 0.0648701, ..., 0.0485228, 0.115357, -0.0281059],
        ...,
        [0.0079577, 0.0635123, -0.0933061, ..., -0.110173, 0.103271, -0.00283601],
        [-0.0550832, -0.0181185, 0.0897263, ..., 0.0434859, -0.0207388, -0.0788009],
        [0.0493263, -0.054047, 0.0608284, ..., -0.0931911, 0.0878982, -0.127275]], dtype=float32)),
('blocks.layers.3.sa.heads.3.tril',
 array([[0.990034, 0, 0, ..., 0, 0, 0],
        [0.990034, 0.990034, 0, ..., 0, 0, 0],
        [0.990034, 0.990034, 0.990034, ..., 0, 0, 0],
        ...,
        [0.990034, 0.990034, 0.990034, ..., 0.990034, 0, 0],
        [0.990034, 0.990034, 0.990034, ..., 0.990034, 0.990034, 0],
        [0.990034, 0.990034, 0.990034, ..., 0.990034, 0.990034, 0.990034]], dtype=float32)),
('blocks.layers.3.sa.proj.weight',
 array([[0.0629363, -0.00679711, 0.083756, ..., 0.0451305, 0.0468571, -0.00835683],
        [0.089052, 0.100368, -0.0346261, ..., -0.17372, -0.0916165, 0.0693777],
        [0.0239351, -0.169322, -0.13728, ..., -0.0203086, 0.00116427, 0.0518053],
        ...,
        [0.0758082, -0.0365724, -0.128635, ..., 0.0781651, 0.102281, -0.0882395],
        [0.00761165, -0.0812714, 0.104707, ..., -0.183323, -0.151983, -0.117825],
        [-0.172656, -0.0207099, -0.0347238, ..., -0.0903112, -0.0853877, 0.0806339]], dtype=float32)),
('blocks.layers.3.sa.proj.bias',
 array([-0.0874269, 0.108266, 0.100784, ..., 0.0620784, 0.0891846, -0.0400162], dtype=float32)),
('blocks.layers.3.ffwd.net.layers.0.weight',
 array([[-0.0069606, 0.046753, -0.0242468, ..., -0.156297, -0.0474405, 0.0202649],
        [0.0751495, -0.0878872, 0.0430179, ..., 0.0773437, 0.0432913, -0.226253],
        [0.304533, 0.152276, 0.142898, ..., -0.170776, -0.0538775, 0.095037],
        ...,
        [-0.0270641, -0.124677, 0.0298718, ..., -0.190682, 0.0982283, -0.173949],
        [0.224475, -0.0181628, 0.125442, ..., -0.0114666, 0.0320751, -0.169612],
        [0.0818121, -0.0999957, 0.0210102, ..., 0.00397924, 0.0691424, 0.0848297]], dtype=float32)),
('blocks.layers.3.ffwd.net.layers.0.bias',
 array([-0.131082, -0.142714, -0.106559, ..., -0.0378557, -0.101047, 0.0616757], dtype=float32)),
('blocks.layers.3.ffwd.net.layers.2.weight',
 array([[-0.0184209, -0.0413515, -0.0355623, ..., 0.0132736, 0.0290336, -0.060557],
        [-0.104655, 0.0328077, 0.0486469, ..., 0.0170743, -0.00394356, 0.0182039],
        [0.0242023, -0.109486, 0.0856898, ..., -0.00901788, 0.0830233, -0.0519543],
        ...,
        [0.0250935, 0.120142, 0.0530416, ..., 0.0115614, -0.00537657, 0.0722595],
```

```
        [0.0949215, 0.04287, 0.0308901, ..., -0.0282363, -0.0614647, -0.0226501],
        [0.00194801, 0.0523767, -0.0635542, ..., 0.0300639, -0.0805319, -0.00419356]], dtype=float32)),
 ('blocks.layers.3.ffwd.net.layers.2.bias',
  array([0.0181848, -0.025388, -0.0605397, ..., 0.0627084, 0.0329226, -0.0390073], dtype=float32)),
 ('blocks.layers.3.ln1.bias',
  array([-0.000276397, -0.0036356, 0.061893, ..., -0.0739511, -0.0647893, -0.200105], dtype=float32)),
 ('blocks.layers.3.ln1.weight',
  array([1.02127, 0.97629, 0.931964, ..., 0.898508, 0.916476, 0.802185], dtype=float32)),
 ('blocks.layers.3.ln2.bias',
  array([0.163224, 0.123759, 0.00455278, ..., 0.0191849, -0.149665, 0.203633], dtype=float32)),
 ('blocks.layers.3.ln2.weight',
  array([0.929846, 1.15319, 0.883077, ..., 1.01516, 0.829225, 1.10779], dtype=float32)),
 ('ln_f.bias',
  array([-0.0506792, 0.00220676, -0.103878, ..., 0.125491, 0.00617851, 0.116968], dtype=float32)),
 ('ln_f.weight',
  array([1.27071, 1.09882, 1.24079, ..., 1.17694, 1.23005, 1.17487], dtype=float32)),
 ('lm_head.weight',
  array([[-0.290424, 0.0571078, -0.110498, ..., 0.115559, 0.123492, 0.0969794],
        [-0.216339, 0.155754, -0.214695, ..., 0.196383, -0.02611, 0.101849],
        [-0.0794341, -0.14962, 0.00913196, ..., 0.132328, -0.0142611, -0.15894],
        ...,
        [0.000720234, -0.138623, -0.00983193, ..., -0.180588, -0.0698929, -0.0369843],
        [0.122889, -0.00964618, 0.102958, ..., 0.0830981, -0.0112409, 0.0560844],
        [0.15354, -0.041468, 0.217757, ..., 0.0522735, 0.0500577, -0.296807]], dtype=float32)),
 ('lm_head.bias',
  array([-0.0383118, 0.136764, -0.10708, ..., -0.146706, -0.0376732, -0.0786435], dtype=float32))]
```

In [ ]:

```python
# save weights
model.save_weights("./shakespearebigramweights.npz")
```

In [ ]:

```python
def generate(model, idx, max_new_tokens):
    # idx is (B, T) array of indices in the current context
    for _ in range(max_new_tokens):
        # crop idx to the last block_size tokens
        idx_cond = idx[:, -block_size:]
        # get the predictions
        logits, _ = model(idx_cond)
        # focus only on the last time step
        logits = logits[:, -1, :] # becomes (B, C)
        # apply softmax to get probabilities
        # probs = nn.softmax(logits, axis=-1) # (B, C)
        probs = logits
        # sample from the distribution
        idx_next = mx.random.categorical(probs, axis=-1, num_samples=1)
        # append sampled index to the running sequence
        idx = mx.concatenate((idx, idx_next), axis=1) # (B, T+1)
        mx.eval(idx)
    return idx
```

In [ ]:

```python
# generate from the model
model.eval()
print("Generating..")
context = mx.zeros((1, 1), dtype=mx.int64)
#generated = model.generate(context, max_new_tokens=1000)[0].tolist()
generated = generate(model, context, max_new_tokens=500)[0].tolist()
print(decode(generated))
```

```
Generating..

fer mablle somedrokes as but bas?

ILsCAUTUS:
Thight my is and, bent shatth, is the is I blorder has goldiust thh wringsenclimes tabeget, my of your thread:
I onlday whersm, dear dockerd to
Loth looke chorrdak whroud in thich mudard, dewer loaftel
And mignsive is ond epuchurtion as rofe.
Of Lance.
To hid, a'e whos; seak, who hew my Godd's midest kignk,
Fagr in as kinkrfeit, ip Ist Ethore
I rake is onate usfh.
HESTIO I:
I whe ofare this prestoondser,
Lettient asike: in bothin so goveet sprorttles
```

In [ ]:

```python
load_weights = True
model = LanguageModel()
if load_weights:
    model.load_weights("./shakespearebigramweights.npz")
context = mx.zeros((1, 1), dtype=mx.int64)
generated = model.generate(context, max_new_tokens=2000)[0].tolist()
print(decode(generated))
```