MLX MinGPT

In [ ]:
```
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
from bitlinear import BitLinear, test_bitlinear_forward_pass, test_bitlinear_initialization, test_bitlinear_no_bias,
```

In [ ]:
```
bitlinear = BitLinear(10, 6, bits=2)
input_tensor = mx.random.normal([6, 10])  # Example input tensor
output = bitlinear(input_tensor)
print(mx.round(output))  # Example output tensor
test_bitlinear_initialization()
test_bitlinear_forward_pass()
test_bitlinear_no_bias()
test_bitlinear_quantization()
```

```
array([[1, 1, -0, 0, 0, -0],
       [1, -0, -0, 0, -0, 0],
       [-1, -0, -0, -0, -0, -0],
       [1, 1, -0, -0, 1, -0],
       [-1, -1, 0, 1, -1, -0],
       [-1, -0, -0, -0, -0, 1]], dtype=float32)
```

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

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

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

    def __init__(self, head_size):
        super().__init__()
        self.key = BitLinear(n_embd, head_size, bias=False)
        self.query = BitLinear(n_embd, head_size, bias=False)
        self.value = BitLinear(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):
```

```python
        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 """

    def __init__(self, num_heads, head_size):
        super().__init__()
        self.heads = [Head(head_size) for _ in range(num_heads)]
        self.proj = BitLinear(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))
        mx.eval(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(
            BitLinear(n_embd, 4 * n_embd),
            nn.ReLU(),
            BitLinear(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 = BitLinear(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)
        mx.eval(logits)

        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)
            mx.eval(idx)
        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 0.240M
```

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)
    mx.eval(loss)
    mx.eval(grads)
    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 4.2018, val loss 4.2021
step 100: train loss 3.3425, val loss 3.3421
step 200: train loss 3.0192, val loss 3.0385
step 300: train loss 3.0550, val loss 3.0989
step 400: train loss 2.8336, val loss 2.8349
step 500: train loss 2.7463, val loss 2.7771
step 600: train loss 2.7106, val loss 2.7468
step 700: train loss 2.6810, val loss 2.7300
step 800: train loss 2.6558, val loss 2.6863
step 900: train loss 2.6376, val loss 2.6702
step 999: train loss 2.6080, val loss 2.6517
Training complete in 327.3664879798889s
```

```
In [ ]: # save weights
        model.save_weights("./bitlinearshakespearebigramweights.npz")
```

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

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

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

```
          )
          (dropout): Dropout(p=0.0)
        )
        (ffwd): FeedFoward(
          (net): Sequential(
            (layers.0): BitLinear(input_dims=64, output_dims=256, bias=True
              (norm): LayerNorm(64, eps=1e-05, affine=True)
            )
            (layers.1): ReLU()
            (layers.2): BitLinear(input_dims=256, output_dims=64, bias=True
              (norm): LayerNorm(256, eps=1e-05, affine=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): BitLinear(input_dims=64, output_dims=65, bias=True
      (norm): LayerNorm(64, eps=1e-05, affine=True)
    )
  )
```

Out[ ]:

```
[('token_embedding_table.weight',
  array([[-0.174924, -0.256206, -0.00602648, ..., -0.486972, 0.0349107, -0.433979],
         [0.671945, 0.513543, -0.155299, ..., -0.232984, -0.178144, -0.304057],
         [-0.256707, 0.0360707, -0.0291384, ..., 0.3691, 0.00463329, 0.25679],
         ...,
         [-0.215785, -0.0437228, -0.119936, ..., -0.172752, 0.353932, -0.0747143],
         [-0.435995, -0.301742, -0.0949814, ..., 0.124442, -0.163173, 0.18864],
         [0.108434, 0.0180779, -0.146745, ..., -0.0957022, 0.221605, 0.0811022]], dtype=float32)),
 ('position_embedding_table.weight',
  array([[-0.0720482, 0.0529957, -0.237977, ..., -0.0643842, 0.0388052, -0.00518194],
         [-0.0405347, 0.140975, -0.05469, ..., -0.0265713, -0.0371366, -0.105592],
         [0.0305713, 0.10734, 0.0803439, ..., -0.0341943, -0.105753, 0.0609333],
         ...,
         [0.0312081, -0.116281, -0.0712877, ..., 0.0789341, -0.145155, 0.000583636],
         [-0.0653304, 0.101005, 0.0639263, ..., 0.0999906, -0.0454837, -0.0433681],
         [-0.0663214, -0.169607, -0.149684, ..., 0.12134, -0.0992697, 0.08638]], dtype=float32)),
 ('blocks.layers.0.sa.heads.0.key.weight',
  array([[-0.153522, 0.0757482, 0.0857889, ..., 0.0713425, 0.128015, 0.129219],
         [-0.128857, 0.091083, -0.123425, ..., -0.156665, -0.0813458, 0.0743055],
         [0.0926207, -0.145864, -0.0735796, ..., 0.0973777, 0.154379, -0.125184],
         ...,
         [0.120148, 0.157459, -0.0723091, ..., 0.0928035, -0.0722707, 0.0709694],
         [0.112244, 0.147376, 0.0938806, ..., -0.0741007, 0.151567, -0.124321],
         [0.125769, 0.0800813, -0.0854335, ..., -0.0951241, 0.0902223, -0.102927]], dtype=float32)),
 ('blocks.layers.0.sa.heads.0.key.norm.bias',
  array([1.74634e-05, 1.61493e-05, 2.3265e-05, ..., 0.000100621, 4.85102e-06, -0.00161957], dtype=float32)),
 ('blocks.layers.0.sa.heads.0.key.norm.weight',
  array([0.832402, 0.732598, 0.943507, ..., 0.881786, 0.9686, 1.35929], dtype=float32)),
 ('blocks.layers.0.sa.heads.0.key.beta',
  array([0.108308, 0.108308, 0.108308, ..., 0.108308, 0.108308, 0.108308], dtype=float32)),
 ('blocks.layers.0.sa.heads.0.key.gamma',
  array([4.56335, 4.56335, 4.56335, ..., 4.56335, 4.56335, 4.56335], dtype=float32)),
 ('blocks.layers.0.sa.heads.0.query.weight',
  array([[-0.0901162, -0.108968, 0.0703857, ..., 0.116831, -0.138207, 0.119403],
         [0.145283, -0.0960958, 0.0815335, ..., 0.155326, -0.0938321, -0.139121],
         [0.135682, 0.106573, -0.132369, ..., 0.0808429, 0.157092, 0.0701572],
         ...,
         [0.0996058, 0.107535, 0.112792, ..., -0.0794489, -0.131377, 0.158615],
         [-0.0741193, 0.0756613, -0.0977425, ..., 0.15926, -0.0805117, 0.130704],
         [-0.11307, 0.0808884, -0.14921, ..., 0.106216, -0.140882, 0.0844541]], dtype=float32)),
 ('blocks.layers.0.sa.heads.0.query.norm.bias',
  array([0.179131, -0.0190841, -0.0850281, ..., -0.114517, 0.176961, -0.139511], dtype=float32)),
 ('blocks.layers.0.sa.heads.0.query.norm.weight',
  array([0.786623, 1.09622, 0.978566, ..., 0.979761, 1.08497, 0.984532], dtype=float32)),
 ('blocks.layers.0.sa.heads.0.query.beta',
  array([0.106562, 0.106562, 0.106562, ..., 0.106562, 0.106562, 0.106562], dtype=float32)),
 ('blocks.layers.0.sa.heads.0.query.gamma',
  array([4.06476, 4.06476, 4.06476, ..., 4.06476, 4.06476, 4.06476], dtype=float32)),
 ('blocks.layers.0.sa.heads.0.value.weight',
  array([[-0.112584, -0.127714, 0.131474, ..., 0.112741, -0.159905, 0.111986],
         [0.115937, -0.14303, 0.132946, ..., -0.148019, -0.109042, 0.151443],
         [0.131875, -0.101756, 0.159896, ..., 0.176842, -0.0900486, 0.162208],
         ...,
         [-0.108604, 0.116721, -0.180202, ..., -0.0974358, 0.119354, 0.181595],
         [0.177614, -0.0982067, -0.0915617, ..., -0.09653, 0.0968969, 0.15424],
         [0.18453, 0.166677, -0.192786, ..., 0.188099, 0.183974, 0.129304]], dtype=float32)),
 ('blocks.layers.0.sa.heads.0.value.norm.bias',
  array([0.0892703, 0.0702319, -0.0191316, ..., -0.0164472, -0.0112349, 0.0423471], dtype=float32)),
 ('blocks.layers.0.sa.heads.0.value.norm.weight',
  array([0.998843, 0.767486, 0.962169, ..., 1.04486, 0.991517, 0.943586], dtype=float32)),
 ('blocks.layers.0.sa.heads.0.value.beta',
  array([0.137352, 0.137352, 0.137352, ..., 0.137352, 0.137352, 0.137352], dtype=float32)),
 ('blocks.layers.0.sa.heads.0.value.gamma',
  array([4.71837, 4.71837, 4.71837, ..., 4.71837, 4.71837, 4.71837], 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([[-0.0909631, 0.133276, 0.0729454, ..., -0.114065, -0.140057, 0.0806142],
        [0.114533, 0.0719919, -0.0733555, ..., -0.149524, 0.121339, 0.0759629],
        [-0.0971918, 0.0722646, -0.0843887, ..., 0.13222, 0.106085, 0.0887604],
        ...,
        [-0.124537, -0.0856785, 0.0724449, ..., -0.114133, 0.110167, 0.149365],
        [0.0787512, 0.0811882, -0.137731, ..., 0.102251, 0.107399, 0.109382],
        [0.0843649, 0.10509, 0.0782485, ..., -0.0719666, -0.0788974, -0.116757]], dtype=float32)),
('blocks.layers.0.sa.heads.1.key.norm.bias',
 array([3.98981e-06, -2.05076e-06, -1.20097e-06, ..., -4.61311e-06, 1.03377e-06, -4.78519e-07], dtype=float32)),
('blocks.layers.0.sa.heads.1.key.norm.weight',
 array([1.07954, 0.855246, 1.09192, ..., 0.938021, 1.07664, 0.870571], dtype=float32)),
('blocks.layers.0.sa.heads.1.key.beta',
 array([0.0972558, 0.0972558, 0.0972558, ..., 0.0972558, 0.0972558, 0.0972558], dtype=float32)),
('blocks.layers.0.sa.heads.1.key.gamma',
 array([3.9404, 3.9404, 3.9404, ..., 3.9404, 3.9404, 3.9404], dtype=float32)),
('blocks.layers.0.sa.heads.1.query.weight',
 array([[-0.0724011, 0.0828326, -0.0726203, ..., 0.0725085, 0.109233, 0.141083],
        [0.0731296, -0.0738593, -0.0770035, ..., -0.0722988, 0.072503, -0.139202],
        [-0.132768, -0.0832781, -0.092318, ..., -0.0861787, 0.146674, -0.0853117],
        ...,
        [0.117449, 0.0740795, 0.0736635, ..., -0.079985, 0.0725408, 0.0725165],
        [0.0874987, 0.094957, 0.0721274, ..., 0.146256, -0.117205, 0.0861534],
        [0.119029, -0.118325, 0.137608, ..., -0.0726231, -0.0746976, -0.072134]], dtype=float32)),
('blocks.layers.0.sa.heads.1.query.norm.bias',
 array([0.0667772, 0.0482006, 0.130476, ..., 0.0234851, -0.0448154, 0.1889], dtype=float32)),
('blocks.layers.0.sa.heads.1.query.norm.weight',
 array([0.896434, 0.843298, 0.994291, ..., 1.09853, 0.921297, 0.749084], dtype=float32)),
('blocks.layers.0.sa.heads.1.query.beta',
 array([0.0960055, 0.0960055, 0.0960055, ..., 0.0960055, 0.0960055, 0.0960055], dtype=float32)),
('blocks.layers.0.sa.heads.1.query.gamma',
 array([4.41098, 4.41098, 4.41098, ..., 4.41098, 4.41098, 4.41098], dtype=float32)),
('blocks.layers.0.sa.heads.1.value.weight',
 array([[0.0601793, -0.0536188, 0.0521267, ..., -0.0589223, 0.0536413, -0.0621165],
        [0.0530147, -0.0542622, -0.0527771, ..., -0.0543432, 0.0532387, 0.0610811],
        [-0.0597266, 0.0610272, -0.0479463, ..., -0.060037, -0.0479129, 0.0537427],
        ...,
        [0.0612915, 0.0609289, -0.0607963, ..., 0.0479012, 0.050191, -0.0528118],
        [-0.0611825, 0.0544111, 0.0545781, ..., 0.0493376, 0.06087, -0.0544618],
        [0.0531906, 0.0609755, -0.0543146, ..., 0.0598544, 0.0525186, -0.0597794]], dtype=float32)),
('blocks.layers.0.sa.heads.1.value.norm.bias',
 array([0.0555079, 0.046389, 0.019977, ..., -0.0682895, 0.00718119, 0.0714401], dtype=float32)),
('blocks.layers.0.sa.heads.1.value.norm.weight',
 array([1.02791, 0.908382, 1.0082, ..., 0.938807, 0.877945, 0.95896], dtype=float32)),
('blocks.layers.0.sa.heads.1.value.beta',
 array([0.0555109, 0.0555109, 0.0555109, ..., 0.0555109, 0.0555109, 0.0555109], dtype=float32)),
('blocks.layers.0.sa.heads.1.value.gamma',
 array([4.57979, 4.57979, 4.57979, ..., 4.57979, 4.57979, 4.57979], 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([[0.115912, -0.0543329, -0.0835812, ..., -0.0718086, -0.0612293, -0.0663604],
        [0.12711, -0.121982, -0.0586069, ..., -0.130547, 0.0836773, 0.137506],
        [0.066144, -0.0566987, -0.0741122, ..., 0.0644699, 0.0650462, -0.0530856],
        ...,
        [-0.0683177, 0.0691225, 0.105412, ..., 0.0721855, 0.056718, 0.0921153],
        [-0.0910782, 0.133612, 0.0540616, ..., -0.056897, -0.0648498, -0.129637],
        [-0.126027, -0.0553463, -0.0567535, ..., 0.0710145, 0.122903, -0.0588624]], dtype=float32)),
('blocks.layers.0.sa.heads.2.key.norm.bias',
 array([2.3269e-05, 3.25225e-06, -4.17475e-06, ..., -2.77154e-05, -5.02932e-05, -1.34593e-05], dtype=float32)),
('blocks.layers.0.sa.heads.2.key.norm.weight',
 array([0.849475, 0.952182, 1.13536, ..., 1.23909, 1.00011, 0.85731], dtype=float32)),
('blocks.layers.0.sa.heads.2.key.beta',
 array([0.0880593, 0.0880593, 0.0880593, ..., 0.0880593, 0.0880593, 0.0880593], dtype=float32)),
('blocks.layers.0.sa.heads.2.key.gamma',
 array([4.29651, 4.29651, 4.29651, ..., 4.29651, 4.29651, 4.29651], dtype=float32)),
('blocks.layers.0.sa.heads.2.query.weight',
 array([[-0.0991103, 0.107724, -0.0595677, ..., 0.108311, -0.0825462, 0.0910974],
        [-0.105903, -0.0549184, 0.127974, ..., -0.0879812, 0.117865, 0.0845754],
        [0.0659454, -0.0553523, -0.0595668, ..., 0.0685694, -0.137535, -0.057566],
        ...,
        [0.0573655, 0.0565245, 0.0865003, ..., 0.0752613, 0.0616006, 0.127145],
        [0.0670247, 0.0647243, 0.120853, ..., 0.0882436, 0.0883197, 0.11144],
        [-0.0955227, 0.0735593, 0.117252, ..., 0.087374, 0.086954, -0.134913]], dtype=float32)),
('blocks.layers.0.sa.heads.2.query.norm.bias',
 array([-0.0160269, -0.0766072, 0.0404695, ..., -0.0235268, 0.202709, 0.0315957], dtype=float32)),
('blocks.layers.0.sa.heads.2.query.norm.weight',
 array([0.874273, 1.10892, 0.871018, ..., 0.919315, 1.10798, 1.04567], dtype=float32)),
('blocks.layers.0.sa.heads.2.query.beta',
```

```
 array([0.0886992, 0.0886992, 0.0886992, ..., 0.0886992, 0.0886992, 0.0886992], dtype=float32)),
('blocks.layers.0.sa.heads.2.query.gamma',
 array([4.75005, 4.75005, 4.75005, ..., 4.75005, 4.75005, 4.75005], dtype=float32)),
('blocks.layers.0.sa.heads.2.value.weight',
 array([[0.0483443, -0.0779714, -0.119712, ..., -0.0856682, 0.0921114, 0.0846855],
        [-0.0310682, 0.02982, -0.0274762, ..., 0.0520496, -0.128166, -0.0814812],
        [-0.0940021, -0.133278, -0.0534537, ..., 0.108692, -0.0493807, -0.053763],
        ...,
        [-0.120494, 0.137008, -0.0784338, ..., -0.0956946, -0.136493, -0.0393666],
        [-0.0408646, 0.119264, 0.0894604, ..., 0.0468993, 0.131591, 0.0397495],
        [-0.0650687, -0.0693476, -0.0836277, ..., -0.137904, 0.110588, 0.0520043]], dtype=float32)),
('blocks.layers.0.sa.heads.2.value.norm.bias',
 array([-0.0606496, 0.004877, 0.0457753, ..., -0.0581952, 0.0407701, -0.0935629], dtype=float32)),
('blocks.layers.0.sa.heads.2.value.norm.weight',
 array([0.905586, 1.0404, 0.894304, ..., 1.05544, 0.963619, 1.10099], dtype=float32)),
('blocks.layers.0.sa.heads.2.value.beta',
 array([0.0805477, 0.0805477, 0.0805477, ..., 0.0805477, 0.0805477, 0.0805477], dtype=float32)),
('blocks.layers.0.sa.heads.2.value.gamma',
 array([5.33461, 5.33461, 5.33461, ..., 5.33461, 5.33461, 5.33461], 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([[0.0532566, -0.0532208, -0.0740527, ..., -0.0341932, 0.0343797, -0.0483017],
        [-0.0792792, 0.0508154, -0.0532803, ..., 0.048548, -0.0339461, -0.0598186],
        [-0.0316465, 0.0365873, -0.0314455, ..., -0.0325844, -0.0960708, 0.0324214],
        ...,
        [0.0373856, 0.0856809, -0.0516244, ..., -0.0347755, 0.0321553, -0.034004],
        [0.0305403, -0.037145, -0.0339884, ..., -0.0363214, 0.0727085, -0.0339109],
        [0.0622668, -0.0355457, 0.0306383, ..., -0.0923623, -0.0534491, 0.0365668]], dtype=float32)),
('blocks.layers.0.sa.heads.3.key.norm.bias',
 array([-8.87596e-06, 1.66304e-05, -6.65174e-06, ..., 1.31122e-05, 9.82613e-07, 9.56673e-06], dtype=float32)),
('blocks.layers.0.sa.heads.3.key.norm.weight',
 array([1.08839, 1.018, 0.914303, ..., 1.07124, 1.01399, 1.03438], dtype=float32)),
('blocks.layers.0.sa.heads.3.key.beta',
 array([0.0540517, 0.0540517, 0.0540517, ..., 0.0540517, 0.0540517, 0.0540517], dtype=float32)),
('blocks.layers.0.sa.heads.3.key.gamma',
 array([3.95796, 3.95796, 3.95796, ..., 3.95796, 3.95796, 3.95796], dtype=float32)),
('blocks.layers.0.sa.heads.3.query.weight',
 array([[-0.0616408, 0.0575557, 0.0329936, ..., -0.0784667, -0.0368955, 0.0617556],
        [0.0930231, 0.033336, -0.0327286, ..., 0.0350484, -0.0365151, 0.0738824],
        [0.0310887, -0.10134, -0.100145, ..., 0.0339818, 0.095126, -0.0321659],
        ...,
        [-0.031277, 0.0466583, -0.0504751, ..., 0.0915575, 0.0344, 0.0290111],
        [-0.0990599, 0.0292123, 0.0550487, ..., -0.0324426, -0.0288512, 0.0423674],
        [-0.0920958, -0.0331466, -0.0943124, ..., 0.036505, 0.0348562, 0.0932523]], dtype=float32)),
('blocks.layers.0.sa.heads.3.query.norm.bias',
 array([0.0442051, 0.0089615, -0.0152196, ..., 0.0578274, 0.0346266, 0.103802], dtype=float32)),
('blocks.layers.0.sa.heads.3.query.norm.weight',
 array([1.0368, 0.960882, 1.12268, ..., 1.02004, 0.962241, 1.00709], dtype=float32)),
('blocks.layers.0.sa.heads.3.query.beta',
 array([0.0544941, 0.0544941, 0.0544941, ..., 0.0544941, 0.0544941, 0.0544941], dtype=float32)),
('blocks.layers.0.sa.heads.3.query.gamma',
 array([4.45835, 4.45835, 4.45835, ..., 4.45835, 4.45835, 4.45835], dtype=float32)),
('blocks.layers.0.sa.heads.3.value.weight',
 array([[-0.0519144, 0.0805436, 0.0881071, ..., -0.0913675, -0.0545108, -0.06984],
        [-0.0518971, -0.0642615, 0.0990016, ..., -0.0698669, 0.0655668, -0.0753211],
        [-0.0655924, 0.057547, -0.0657664, ..., 0.0521908, 0.0599997, -0.0588297],
        ...,
        [-0.0566299, 0.0516795, -0.0730555, ..., 0.0548371, 0.0612824, 0.0611912],
        [0.0534685, -0.0546696, 0.0646658, ..., 0.0947927, -0.0541219, 0.051611],
        [0.0843464, 0.0519288, 0.0674733, ..., 0.0607819, -0.0575678, 0.0591534]], dtype=float32)),
('blocks.layers.0.sa.heads.3.value.norm.bias',
 array([-0.0283065, -0.0256394, -0.0266231, ..., -0.0518848, 0.0105842, 0.0390735], dtype=float32)),
('blocks.layers.0.sa.heads.3.value.norm.weight',
 array([1.02195, 0.911673, 1.0259, ..., 1.01506, 1.0991, 1.03553], dtype=float32)),
('blocks.layers.0.sa.heads.3.value.beta',
 array([0.0660241, 0.0660241, 0.0660241, ..., 0.0660241, 0.0660241, 0.0660241], dtype=float32)),
('blocks.layers.0.sa.heads.3.value.gamma',
 array([3.92353, 3.92353, 3.92353, ..., 3.92353, 3.92353, 3.92353], 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([[0.00525264, -0.00251156, -0.00386324, ..., 0.00333162, 0.00194031, 0.0041148],
        [0.00488607, -0.00514601, 0.00360405, ..., 0.00484113, 0.00506546, -0.00442917],
        [0.00218628, -0.00507659, 0.00199852, ..., -0.00197272, -0.00566356, -0.00494454],
        ...,
        [0.00251372, -0.00198819, -0.00410708, ..., 0.00500537, -0.00411358, -0.00333124],
        [-0.00485857, -0.00432339, -0.00517581, ..., -0.00567143, 0.00401163, -0.00270999],
        [-0.00196036, -0.00515007, 0.00199902, ..., -0.00437191, 0.00486412, 0.0021838]], dtype=float32)),
```

```
('blocks.layers.0.sa.proj.bias',
 array([-0.0342164, 0.0221167, -0.145738, ..., 0.203503, -0.044272, -0.118985], dtype=float32)),
('blocks.layers.0.sa.proj.norm.bias',
 array([-0.105665, 0.0109794, 0.0473558, ..., -0.0723642, -0.0245013, -0.118124], dtype=float32)),
('blocks.layers.0.sa.proj.norm.weight',
 array([1.00619, 0.971266, 0.978792, ..., 1.11586, 0.937236, 1.03841], dtype=float32)),
('blocks.layers.0.sa.proj.beta',
 array([0.00452865, 0.00452865, 0.00452865, ..., 0.00452865, 0.00452865, 0.00452865], dtype=float32)),
('blocks.layers.0.sa.proj.gamma',
 array([4.14242, 4.14242, 4.14242, ..., -0.0211619, -0.057201, 0.0354126], dtype=float32)),
('blocks.layers.0.ffwd.net.layers.0.weight',
 array([[-0.028706, -0.00531509, 0.00728135, ..., -0.0048176, 0.029554, 0.0373312],
        [0.028359, 0.0121091, 0.0208292, ..., -0.0188137, 2.74536e-05, -0.0177039],
        [-0.00477664, -2.78148e-05, 0.00428192, ..., -0.0269861, 0.00238161, 0.0325592],
        ...,
        [-0.011965, -0.0157859, -0.0283405, ..., -6.08389e-05, -0.0222883, 0.0135259],
        [0.00472764, -0.0262064, 3.03808e-05, ..., -0.0195036, 0.00967828, 0.0253789],
        [-0.0134958, 0.00744427, 0.0292034, ..., 0.000230127, 0.0193958, 0.007865]], dtype=float32)),
('blocks.layers.0.ffwd.net.layers.0.bias',
 array([-0.20809, 0.173662, -0.165353, ..., 0.0644989, -0.0482321, 0.0425789], dtype=float32)),
('blocks.layers.0.ffwd.net.layers.0.norm.bias',
 array([-0.00899193, -0.187948, -0.0861969, ..., -0.0477558, -0.209153, 0.0858074], dtype=float32)),
('blocks.layers.0.ffwd.net.layers.0.norm.weight',
 array([0.941731, 1.02355, 1.09186, ..., 0.861353, 0.801856, 1.17174], dtype=float32)),
('blocks.layers.0.ffwd.net.layers.0.beta',
 array([0.0151266, 0.0151266, 0.0151266, ..., 0.0151266, 0.0151266, 0.0151266], dtype=float32)),
('blocks.layers.0.ffwd.net.layers.0.gamma',
 array([4.4731, 4.4731, 4.4731, ..., 0, 0, 0], dtype=float32)),
('blocks.layers.0.ffwd.net.layers.2.weight',
 array([[-8.26842e-05, -0.00221423, 0.000428998, ..., -0.000717072, -0.000254082, -0.00518875],
        [-0.00017781, -0.000815567, 0.000376629, ..., 0.000160015, -0.00016209, 0.00019461],
        [-0.00203767, 0.00221293, -0.00120721, ..., -0.000165048, 0.0017181, 0.000255348],
        ...,
        [0.00203917, 0.00183703, -0.00129844, ..., -0.000498242, -0.00203873, 0.000211376],
        [0.00016802, 4.35653e-05, -0.000713677, ..., 0.00216151, -0.000627215, 0.000716834],
        [0.00239324, 0.000178911, 0.000249887, ..., -0.0017594, 0.000264509, -0.000309832]], dtype=float32)),
('blocks.layers.0.ffwd.net.layers.2.bias',
 array([-0.0875532, 0.258657, 0.0342219, ..., 0.374254, -0.00859845, -0.0194522], dtype=float32)),
('blocks.layers.0.ffwd.net.layers.2.norm.bias',
 array([-0.134357, 0.0616206, -0.135285, ..., -0.0573165, 0.108993, 0.0831157], dtype=float32)),
('blocks.layers.0.ffwd.net.layers.2.norm.weight',
 array([0.771382, 1.10109, 0.743615, ..., 1.01247, 0.852797, 0.9634], dtype=float32)),
('blocks.layers.0.ffwd.net.layers.2.beta',
 array([0.00121778, 0.00121778, 0.00121778, ..., 0.00121778, 0.00121778, 0.00121778], dtype=float32)),
('blocks.layers.0.ffwd.net.layers.2.gamma',
 array([13.2335, 13.2335, 13.2335, ..., 0.297613, -0.194444, 0.575788], dtype=float32)),
('blocks.layers.0.ln1.bias',
 array([0.0613944, 0.0358848, 0.00992538, ..., -0.113643, 0.0300603, -0.0529632], dtype=float32)),
('blocks.layers.0.ln1.weight',
 array([0.880502, 0.866058, 0.991547, ..., 1.01717, 0.981604, 1.07562], dtype=float32)),
('blocks.layers.0.ln2.bias',
 array([0.00871623, -0.186571, -0.0795889, ..., -0.0307241, -0.195097, 0.0976039], dtype=float32)),
('blocks.layers.0.ln2.weight',
 array([0.933083, 1.00789, 1.08571, ..., 0.87058, 0.778188, 1.18695], dtype=float32)),
('blocks.layers.1.sa.heads.0.key.weight',
 array([[-0.00561396, -0.00511645, -0.00539, ..., -0.00506118, 0.00509877, -0.0082099],
        [-0.00681938, -0.00566296, -0.00497653, ..., 0.00599303, -0.00558971, -0.00551599],
        [0.00441666, -0.00620946, 0.0044053, ..., 0.00392598, -0.00825722, -0.00572008],
        ...,
        [0.00497798, 0.00495522, 0.00561536, ..., 0.00570759, -0.00320228, 0.00391894],
        [-0.0203905, -0.00574238, -0.00509961, ..., 0.00593701, 0.00505369, -0.00795373],
        [0.00436587, 0.00481381, 0.00572437, ..., -0.00776662, -0.00738234, -0.0040913]], dtype=float32)),
('blocks.layers.1.sa.heads.0.key.norm.bias',
 array([1.96287e-06, -3.37438e-05, -1.4691e-06, ..., 1.43374e-06, 2.68392e-06, -4.46244e-06], dtype=float32)),
('blocks.layers.1.sa.heads.0.key.norm.weight',
 array([0.87695, 0.939196, 0.947585, ..., 1.10441, 1.00918, 0.854907], dtype=float32)),
('blocks.layers.1.sa.heads.0.key.beta',
 array([0.00580456, 0.00580456, 0.00580456, ..., 0.00580456, 0.00580456, 0.00580456], dtype=float32)),
('blocks.layers.1.sa.heads.0.key.gamma',
 array([3.54513, 3.54513, 3.54513, ..., 3.54513, 3.54513, 3.54513], dtype=float32)),
('blocks.layers.1.sa.heads.0.query.weight',
 array([[0.00658693, -0.00592033, 0.0156851, ..., 0.00700563, -0.00382035, 0.00580315],
        [0.00249171, -0.00656996, -0.0061168, ..., -0.00438258, -0.00438813, 0.00710978],
        [-0.00378565, 0.00248249, -0.00996248, ..., -0.00662148, -0.00410263, 0.00374011],
        ...,
        [0.00673337, 0.00250585, -0.0213697, ..., 0.00767163, -0.00320725, 0.00481798],
        [0.00324941, -0.00412298, 0.00825675, ..., 0.00564933, -0.00376114, -0.00685516],
        [0.00614202, -0.00250258, 0.00822485, ..., 0.00320967, 0.00516287, -0.00319394]], dtype=float32)),
('blocks.layers.1.sa.heads.0.query.norm.bias',
 array([0.0440023, -0.0479969, -0.0570684, ..., 0.0858662, 0.0734066, 0.0207458], dtype=float32)),
('blocks.layers.1.sa.heads.0.query.norm.weight',
 array([1.0671, 0.884978, 0.953502, ..., 0.906224, 1.03587, 1.0555], dtype=float32)),
('blocks.layers.1.sa.heads.0.query.beta',
 array([0.00581242, 0.00581242, 0.00581242, ..., 0.00581242, 0.00581242, 0.00581242], dtype=float32)),
('blocks.layers.1.sa.heads.0.query.gamma',
 array([3.68215, 3.68215, 3.68215, ..., 3.68215, 3.68215, 3.68215], dtype=float32)),
('blocks.layers.1.sa.heads.0.value.weight',
 array([[0.162913, -0.216298, -0.188817, ..., -0.144758, 0.158691, -0.177463],
        [0.21192, 0.244181, 0.253791, ..., 0.144607, -0.246512, -0.223925],
        [0.169524, 0.168746, 0.166129, ..., 0.246966, 0.200233, 0.196958],
        ...,
```

```
        [0.225199, 0.211847, -0.233566, ..., 0.196746, 0.228922, -0.208823],
        [0.195953, 0.254288, -0.15704, ..., -0.202771, 0.136849, 0.140595],
        [-0.203606, -0.247108, -0.25945, ..., -0.237137, 0.25452, 0.199029]], dtype=float32)),
('blocks.layers.1.sa.heads.0.value.norm.bias',
 array([0.0899908, 0.249424, -0.109722, ..., -0.086134, 0.0413329, -0.178831], dtype=float32)),
('blocks.layers.1.sa.heads.0.value.norm.weight',
 array([0.826227, 1.18317, 1.00389, ..., 1.04225, 0.830358, 1.07175], dtype=float32)),
('blocks.layers.1.sa.heads.0.value.beta',
 array([0.197953, 0.197953, 0.197953, ..., 0.197953, 0.197953, 0.197953], dtype=float32)),
('blocks.layers.1.sa.heads.0.value.gamma',
 array([4.21223, 4.21223, 4.21223, ..., 4.21223, 4.21223, 4.21223], 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([[0.18171, 0.116484, 0.145418, ..., 0.181569, 0.13276, -0.177693],
        [0.164629, 0.179073, 0.109399, ..., 0.156664, -0.173575, 0.133277],
        [-0.110488, 0.184613, 0.153601, ..., 0.165084, -0.197785, -0.184148],
        ...,
        [0.168503, -0.0965505, -0.197578, ..., -0.154397, 0.116767, -0.146449],
        [0.113348, 0.167648, 0.129419, ..., 0.101683, -0.187636, 0.104569],
        [0.128527, -0.113118, 0.144667, ..., -0.100875, 0.113013, -0.189466]], dtype=float32)),
('blocks.layers.1.sa.heads.1.key.norm.bias',
 array([1.40235e-05, -1.26776e-05, 1.12071e-05, ..., -1.559e-05, 1.12157e-05, -1.59607e-05], dtype=float32)),
('blocks.layers.1.sa.heads.1.key.norm.weight',
 array([0.89913, 0.801847, 1.05894, ..., 0.859688, 1.09962, 1.06454], dtype=float32)),
('blocks.layers.1.sa.heads.1.key.beta',
 array([0.144664, 0.144664, 0.144664, ..., 0.144664, 0.144664, 0.144664], dtype=float32)),
('blocks.layers.1.sa.heads.1.key.gamma',
 array([4.25886, 4.25886, 4.25886, ..., 4.25886, 4.25886, 4.25886], dtype=float32)),
('blocks.layers.1.sa.heads.1.query.weight',
 array([[0.132657, 0.194046, -0.174441, ..., -0.130469, -0.147619, -0.17343],
        [0.112867, -0.125209, 0.180319, ..., -0.130919, 0.188058, -0.115656],
        [-0.139173, 0.109989, 0.190529, ..., -0.179112, 0.183355, 0.163515],
        ...,
        [0.170164, 0.18483, -0.103676, ..., 0.16991, 0.171597, -0.18375],
        [-0.129417, 0.136742, -0.171277, ..., -0.106194, 0.129755, -0.156968],
        [-0.141851, -0.176604, 0.153394, ..., -0.117472, 0.171599, 0.179068]], dtype=float32)),
('blocks.layers.1.sa.heads.1.query.norm.bias',
 array([-0.0308227, 0.236169, -0.0268955, ..., -0.0747902, 0.0547579, -0.0496506], dtype=float32)),
('blocks.layers.1.sa.heads.1.query.norm.weight',
 array([0.820035, 1.0512, 1.12657, ..., 0.993627, 1.14055, 1.19959], dtype=float32)),
('blocks.layers.1.sa.heads.1.query.beta',
 array([0.144621, 0.144621, 0.144621, ..., 0.144621, 0.144621, 0.144621], dtype=float32)),
('blocks.layers.1.sa.heads.1.query.gamma',
 array([3.97467, 3.97467, 3.97467, ..., 3.97467, 3.97467, 3.97467], dtype=float32)),
('blocks.layers.1.sa.heads.1.value.weight',
 array([[-0.00696314, 0.0180977, 0.00671812, ..., -0.0194653, -0.0271942, 0.0183689],
        [-0.0157033, 0.0069782, 0.0105964, ..., 0.0147887, 0.0105413, 0.00382718],
        [0.0186677, 0.00668693, 0.0277484, ..., -0.00447226, -0.018908, -0.0129601],
        ...,
        [0.0192432, 0.0102686, 0.0130204, ..., 0.00674987, 0.00680662, 0.0175687],
        [-0.00800178, -0.0124064, 0.0149456, ..., -0.00543347, -0.0099451, -0.0275938],
        [-0.00550275, 0.00631572, 0.0146506, ..., -0.0147055, -0.0118005, 0.00511285]], dtype=float32)),
('blocks.layers.1.sa.heads.1.value.norm.bias',
 array([-0.119304, 0.053314, -0.0312358, ..., -0.0353769, -0.0287679, -0.0236793], dtype=float32)),
('blocks.layers.1.sa.heads.1.value.norm.weight',
 array([1.07807, 0.982916, 1.01958, ..., 0.986216, 0.910838, 1.00158], dtype=float32)),
('blocks.layers.1.sa.heads.1.value.beta',
 array([0.0134228, 0.0134228, 0.0134228, ..., 0.0134228, 0.0134228, 0.0134228], dtype=float32)),
('blocks.layers.1.sa.heads.1.value.gamma',
 array([3.95363, 3.95363, 3.95363, ..., 3.95363, 3.95363, 3.95363], 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([[0.242394, -0.170227, 0.179464, ..., -0.148085, -0.162737, -0.182487],
        [-0.193168, 0.22855, 0.235524, ..., -0.224269, -0.166716, 0.173465],
        [0.240336, 0.253123, 0.243207, ..., 0.185499, -0.198558, -0.22378],
        ...,
        [0.158969, -0.248891, -0.221319, ..., 0.222981, -0.224578, 0.202694],
        [-0.203974, 0.207388, 0.252962, ..., 0.218846, 0.244881, -0.245623],
        [-0.181228, 0.24985, 0.145417, ..., 0.185962, -0.19796, 0.17443]], dtype=float32)),
('blocks.layers.1.sa.heads.2.key.norm.bias',
 array([0.00055427, 0.000251486, 0.000280288, ..., 0.000550489, -0.000279098, -1.58039e-05], dtype=float32)),
('blocks.layers.1.sa.heads.2.key.norm.weight',
 array([1.02564, 0.987566, 1.13262, ..., 0.674369, 1.05391, 1.1437], dtype=float32)),
('blocks.layers.1.sa.heads.2.key.beta',
 array([0.194271, 0.194271, 0.194271, ..., 0.194271, 0.194271, 0.194271], dtype=float32)),
('blocks.layers.1.sa.heads.2.key.gamma',
 array([3.74211, 3.74211, 3.74211, ..., 3.74211, 3.74211, 3.74211], dtype=float32)),
```

```
('blocks.layers.1.sa.heads.2.query.weight',
 array([[0.207069, -0.132982, -0.232713, ..., 0.143238, -0.181896, 0.203008],
        [0.15249, -0.183404, 0.248978, ..., 0.207447, -0.179428, -0.199425],
        [-0.22018, 0.219774, 0.226531, ..., 0.143471, -0.215641, 0.159745],
        ...,
        [-0.164277, -0.196353, 0.175653, ..., 0.231982, 0.197045, 0.248591],
        [-0.211984, -0.175845, -0.168291, ..., 0.235039, -0.224141, 0.249203],
        [-0.220642, 0.21326, -0.166822, ..., -0.222613, 0.16674, -0.25431]], dtype=float32)),
('blocks.layers.1.sa.heads.2.query.norm.bias',
 array([0.0209746, -0.14008, -0.0932231, ..., -0.0308156, 0.0528947, -0.114926], dtype=float32)),
('blocks.layers.1.sa.heads.2.query.norm.weight',
 array([0.964882, 0.972171, 1.25532, ..., 1.02752, 0.803501, 0.941028], dtype=float32)),
('blocks.layers.1.sa.heads.2.query.beta',
 array([0.194028, 0.194028, 0.194028, ..., 0.194028, 0.194028, 0.194028], dtype=float32)),
('blocks.layers.1.sa.heads.2.query.gamma',
 array([4.7004, 4.7004, 4.7004, ..., 4.7004, 4.7004, 4.7004], dtype=float32)),
('blocks.layers.1.sa.heads.2.value.weight',
 array([[-0.0860178, -0.0212617, 0.0619372, ..., -0.0646112, -0.0932709, 0.0888742],
        [0.0451033, -0.0504794, 0.0450114, ..., -0.0955302, 0.0229228, 0.0272601],
        [-0.0684763, -0.0447053, -0.0365621, ..., -0.0374052, 0.0511692, 0.049373],
        ...,
        [-0.0819971, -0.0948336, 0.0541039, ..., -0.0944114, -0.0631267, -0.0781442],
        [0.0811393, 0.0646254, -0.0953044, ..., -0.0637476, 0.0845208, -0.0949726],
        [0.0731486, 0.0460823, 0.0434461, ..., -0.0985354, -0.0718457, 0.0539093]], dtype=float32)),
('blocks.layers.1.sa.heads.2.value.norm.bias',
 array([-0.242258, -0.00290805, 0.0726632, ..., 0.0494219, -0.166681, -0.177902], dtype=float32)),
('blocks.layers.1.sa.heads.2.value.norm.weight',
 array([1.09628, 1.02415, 0.945715, ..., 1.00716, 0.925761, 1.11017], dtype=float32)),
('blocks.layers.1.sa.heads.2.value.beta',
 array([0.0655327, 0.0655327, 0.0655327, ..., 0.0655327, 0.0655327, 0.0655327], dtype=float32)),
('blocks.layers.1.sa.heads.2.value.gamma',
 array([4.29603, 4.29603, 4.29603, ..., 4.29603, 4.29603, 4.29603], 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([[0.152858, -0.112018, 0.11323, ..., -0.205295, 0.187489, -0.151258],
        [-0.218265, -0.187373, -0.175724, ..., 0.121991, -0.136173, -0.134112],
        [0.209513, -0.162892, -0.224133, ..., -0.153186, -0.186122, 0.189103],
        ...,
        [0.153635, 0.177484, 0.212104, ..., 0.113404, -0.175413, 0.177631],
        [0.103233, -0.120001, 0.171318, ..., 0.118824, -0.186859, 0.181697],
        [-0.167817, -0.120423, -0.171313, ..., 0.174034, 0.128288, 0.191318]], dtype=float32)),
('blocks.layers.1.sa.heads.3.key.norm.bias',
 array([-1.1514e-05, 0.000278597, -7.42928e-06, ..., -4.06212e-05, 0.000158174, -0.000112174], dtype=float32)),
('blocks.layers.1.sa.heads.3.key.norm.weight',
 array([1.13387, 1.15798, 1.05241, ..., 1.04498, 0.869018, 1.02297], dtype=float32)),
('blocks.layers.1.sa.heads.3.key.beta',
 array([0.163266, 0.163266, 0.163266, ..., 0.163266, 0.163266, 0.163266], dtype=float32)),
('blocks.layers.1.sa.heads.3.key.gamma',
 array([3.95206, 3.95206, 3.95206, ..., 3.95206, 3.95206, 3.95206], dtype=float32)),
('blocks.layers.1.sa.heads.3.query.weight',
 array([[-0.208676, 0.105755, 0.221737, ..., -0.13563, 0.192644, 0.225446],
        [0.223271, 0.111191, -0.111407, ..., 0.13922, -0.210959, -0.19222],
        [-0.151124, 0.111777, 0.20307, ..., -0.211206, 0.1639, -0.174477],
        ...,
        [-0.161253, 0.114958, -0.208736, ..., -0.168744, -0.185498, -0.214879],
        [-0.152816, -0.12424, -0.211166, ..., 0.168215, -0.208528, 0.221825],
        [-0.223149, -0.208395, 0.176234, ..., 0.141449, 0.109774, -0.154102]], dtype=float32)),
('blocks.layers.1.sa.heads.3.query.norm.bias',
 array([-6.84817e-05, 0.15418, -0.0404984, ..., -0.0234929, -0.0583097, 0.0238919], dtype=float32)),
('blocks.layers.1.sa.heads.3.query.norm.weight',
 array([1.05025, 1.05171, 0.788123, ..., 1.07561, 0.896115, 0.755596], dtype=float32)),
('blocks.layers.1.sa.heads.3.query.beta',
 array([0.163937, 0.163937, 0.163937, ..., 0.163937, 0.163937, 0.163937], dtype=float32)),
('blocks.layers.1.sa.heads.3.query.gamma',
 array([4.07755, 4.07755, 4.07755, ..., 4.07755, 4.07755, 4.07755], dtype=float32)),
('blocks.layers.1.sa.heads.3.value.weight',
 array([[-0.0416752, 0.0153699, 0.0431519, ..., 0.0888875, 0.0384437, -0.0925827],
        [-0.0437805, 0.02767, -0.0634696, ..., -0.0277473, -0.0176756, -0.104534],
        [-0.0219018, 0.0380765, 0.0744587, ..., -0.100502, 0.105468, -0.0180272],
        ...,
        [-0.0232259, -0.0378908, -0.0277935, ..., 0.0452508, 0.0760994, -0.0197684],
        [-0.014643, -0.0954126, 0.0438752, ..., -0.0831824, -0.0278842, -0.0273377],
        [-0.0538047, 0.092746, -0.0181927, ..., -0.0589692, 0.0749334, 0.024171]], dtype=float32)),
('blocks.layers.1.sa.heads.3.value.norm.bias',
 array([0.0359199, 0.00154843, 0.0198282, ..., 0.037433, 0.0755274, 0.0793006], dtype=float32)),
('blocks.layers.1.sa.heads.3.value.norm.weight',
 array([1.19696, 0.992095, 0.912675, ..., 0.931426, 0.942254, 0.878822], dtype=float32)),
('blocks.layers.1.sa.heads.3.value.beta',
 array([0.0518275, 0.0518275, 0.0518275, ..., 0.0518275, 0.0518275, 0.0518275], dtype=float32)),
('blocks.layers.1.sa.heads.3.value.gamma',
 array([3.83102, 3.83102, 3.83102, ..., 3.83102, 3.83102, 3.83102], 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([[0.000635489, 0.000485308, 0.00141273, ..., -0.00160126, -0.00159752, -0.00203078],
        [-0.00137041, -0.000589031, 0.000348758, ..., -0.00138147, 0.000612366, 0.000624706],
        [0.00139969, 0.00133246, -0.00114376, ..., -0.00137767, -0.00137368, -0.00140041],
        ...,
        [-0.000770359, -0.00114987, -0.00203184, ..., -0.000759528, -0.000640328, -0.000738978],
        [-0.000711441, 0.00218341, -0.000709618, ..., -0.00151753, -0.000814952, 0.000911398],
        [0.00218641, -0.000235746, 0.00138001, ..., 0.00202701, 0.000779812, 0.00142475]], dtype=float32)),
('blocks.layers.1.sa.proj.bias',
 array([-0.196634, 0.256967, -0.00278145, ..., -0.309086, -0.0511221, 0.193555], dtype=float32)),
('blocks.layers.1.sa.proj.norm.bias',
 array([0.00529242, 0.0539704, -0.284012, ..., 0.0107283, -0.0247369, -0.0956836], dtype=float32)),
('blocks.layers.1.sa.proj.norm.weight',
 array([0.72502, 0.963715, 0.87137, ..., 0.995283, 1.02108, 0.982475], dtype=float32)),
('blocks.layers.1.sa.proj.beta',
 array([0.000745652, 0.000745652, 0.000745652, ..., 0.000745652, 0.000745652, 0.000745652], dtype=float32)),
('blocks.layers.1.sa.proj.gamma',
 array([5.99749, 5.99749, 5.99749, ..., -0.0272826, -0.104036, -0.210128], dtype=float32)),
('blocks.layers.1.ffwd.net.layers.0.weight',
 array([[0.00578902, 0.0306367, -0.028798, ..., -0.00845208, -0.030308, 0.0375361],
        [-0.0356648, -0.0394911, 0.00426653, ..., -0.0400245, -0.00362503, -0.0274544],
        [-0.0269654, -0.0136022, 0.00893842, ..., 0.0305749, -0.0289137, -0.039514],
        ...,
        [0.0342972, -0.0328986, -0.0310103, ..., 0.0360698, 0.0570765, 0.00641959],
        [0.0358668, -0.00860157, 0.0306944, ..., 0.0352434, -0.037972, -0.0346558],
        [0.03481, -0.0354283, 0.0374558, ..., 0.0384967, -0.0375858, 0.0376325]], dtype=float32)),
('blocks.layers.1.ffwd.net.layers.0.bias',
 array([-0.110341, 0.0247525, 0.0584188, ..., 0.0134211, 0.118706, 0.0160132], dtype=float32)),
('blocks.layers.1.ffwd.net.layers.0.norm.bias',
 array([-0.313861, 0.26158, 0.156387, ..., -0.146737, 0.142741, -0.261547], dtype=float32)),
('blocks.layers.1.ffwd.net.layers.0.norm.weight',
 array([1.08969, 1.10778, 0.724832, ..., 0.987178, 0.975334, 1.31351], dtype=float32)),
('blocks.layers.1.ffwd.net.layers.0.beta',
 array([0.0296339, 0.0296339, 0.0296339, ..., 0.0296339, 0.0296339, 0.0296339], dtype=float32)),
('blocks.layers.1.ffwd.net.layers.0.gamma',
 array([5.33643, 5.33643, 5.33643, ..., 0, 0, 0], dtype=float32)),
('blocks.layers.1.ffwd.net.layers.2.weight',
 array([[-0.000376157, 0.00226138, 6.60557e-05, ..., -0.00318431, 0.000942084, 0.000893937],
        [-0.000908904, -0.00069726, 0.0014591, ..., 0.00098832, -0.0032751, 0.00281486],
        [0.000574096, -6.53911e-05, 0.00285955, ..., 0.00328922, 0.000340501, 0.00316225],
        ...,
        [-0.00328714, 0.000462941, 0.00315016, ..., 0.0004647, 0.000915649, 0.000760221],
        [-0.000235118, -0.00080126, -0.0024289, ..., 0.000400375, -0.00285627, -0.00334555],
        [0.000892674, -0.00344189, -0.000576264, ..., -0.000777892, -0.00327708, -0.00166663]], dtype=float32)),
('blocks.layers.1.ffwd.net.layers.2.bias',
 array([-0.0349325, 0.0679623, -0.0255273, ..., 0.235359, 0.0578397, 0.0911798], dtype=float32)),
('blocks.layers.1.ffwd.net.layers.2.norm.bias',
 array([-0.16736, -0.164527, 0.0187389, ..., 0.0134206, -0.03485, 0.136171], dtype=float32)),
('blocks.layers.1.ffwd.net.layers.2.norm.weight',
 array([0.840673, 0.967878, 0.88618, ..., 0.966752, 1.04931, 0.861615], dtype=float32)),
('blocks.layers.1.ffwd.net.layers.2.beta',
 array([0.00174075, 0.00174075, 0.00174075, ..., 0.00174075, 0.00174075, 0.00174075], dtype=float32)),
('blocks.layers.1.ffwd.net.layers.2.gamma',
 array([13.9453, 13.9453, 13.9453, ..., 0.158777, 0.22106, -0.197524], dtype=float32)),
('blocks.layers.1.ln1.bias',
 array([-0.178226, 0.22172, -0.0891448, ..., -0.0363571, 0.0299928, -0.137004], dtype=float32)),
('blocks.layers.1.ln1.weight',
 array([1.11361, 1.10517, 0.9197, ..., 1.0542, 0.741108, 0.967163], dtype=float32)),
('blocks.layers.1.ln2.bias',
 array([-0.299627, 0.274795, 0.172074, ..., -0.141955, 0.16294, -0.267235], dtype=float32)),
('blocks.layers.1.ln2.weight',
 array([1.08831, 1.11247, 0.726584, ..., 1.00892, 0.971484, 1.21704], dtype=float32)),
('blocks.layers.2.sa.heads.0.key.weight',
 array([[-0.0738496, -0.0485664, 0.0614584, ..., 0.0810552, 0.0547505, -0.0932569],
        [0.0560509, -0.0550904, 0.0524176, ..., 0.0727731, -0.109541, -0.0449605],
        [-0.0493726, 0.0565005, -0.0720222, ..., 0.0864682, 0.0719729, -0.0805422],
        ...,
        [-0.0496125, 0.0684562, 0.0656936, ..., 0.107561, 0.0558877, 0.0703757],
        [0.0756866, 0.0485687, -0.0831832, ..., 0.0944967, 0.0611912, 0.0927661],
        [-0.104884, -0.0481324, -0.0463012, ..., 0.0541711, 0.0567723, 0.106103]], dtype=float32)),
('blocks.layers.2.sa.heads.0.key.norm.bias',
 array([-0.000173494, -2.71747e-05, 2.02246e-06, ..., -5.59716e-06, 2.80638e-05, -2.285e-05], dtype=float32)),
('blocks.layers.2.sa.heads.0.key.norm.weight',
 array([0.985126, 0.825755, 0.9377, ..., 1.10185, 1.14245, 0.905358], dtype=float32)),
('blocks.layers.2.sa.heads.0.key.beta',
 array([0.0723096, 0.0723096, 0.0723096, ..., 0.0723096, 0.0723096, 0.0723096], dtype=float32)),
('blocks.layers.2.sa.heads.0.key.gamma',
 array([3.90628, 3.90628, 3.90628, ..., 3.90628, 3.90628, 3.90628], dtype=float32)),
('blocks.layers.2.sa.heads.0.query.weight',
 array([[0.0805463, -0.111832, 0.0649759, ..., 0.0771951, 0.045785, -0.0674731],
        [-0.0986747, 0.0955238, 0.0601452, ..., -0.0878145, -0.0826593, 0.0743067],
        [0.0479479, -0.0707163, 0.0467791, ..., 0.0475436, 0.0956044, -0.0622677],
        ...,
        [-0.0656437, -0.0657023, 0.0496633, ..., -0.0626795, -0.0534992, -0.0806814],
        [-0.0497788, 0.0716657, -0.0754104, ..., 0.0581298, 0.0678984, -0.0890034],
        [0.088085, -0.0709129, 0.0613097, ..., -0.0752494, -0.0468831, -0.0846885]], dtype=float32)),
```

```
('blocks.layers.2.sa.heads.0.query.norm.bias',
 array([0.115599, 0.0791384, -0.0594401, ..., 0.0398441, 0.0157856, -0.162089], dtype=float32)),
('blocks.layers.2.sa.heads.0.query.norm.weight',
 array([0.833509, 1.0175, 0.947438, ..., 1.0151, 0.65122, 1.166], dtype=float32)),
('blocks.layers.2.sa.heads.0.query.beta',
 array([0.0707642, 0.0707642, 0.0707642, ..., 0.0707642, 0.0707642, 0.0707642], dtype=float32)),
('blocks.layers.2.sa.heads.0.query.gamma',
 array([4.4549, 4.4549, 4.4549, ..., 4.4549, 4.4549, 4.4549], dtype=float32)),
('blocks.layers.2.sa.heads.0.value.weight',
 array([[-0.0768991, -0.0835745, 0.0847666, ..., -0.0596397, -0.0545639, 0.104771],
        [0.0564625, -0.100315, 0.0935783, ..., -0.0943618, 0.0884175, 0.0620358],
        [-0.0688659, -0.102763, -0.0776379, ..., -0.0664881, 0.146983, 0.114125],
        ...,
        [-0.142654, -0.0807899, 0.0555015, ..., 0.0917064, -0.143577, 0.0693294],
        [0.120511, -0.0813731, 0.0551328, ..., -0.118657, 0.126185, -0.0779104],
        [0.116527, -0.114196, -0.0947616, ..., -0.137416, 0.0748042, 0.135942]], dtype=float32)),
('blocks.layers.2.sa.heads.0.value.norm.bias',
 array([-0.152813, -0.0130034, 0.0626074, ..., 0.121534, -0.0632042, 0.064478], dtype=float32)),
('blocks.layers.2.sa.heads.0.value.norm.weight',
 array([0.993616, 1.10883, 0.958624, ..., 0.885992, 0.808511, 0.906487], dtype=float32)),
('blocks.layers.2.sa.heads.0.value.beta',
 array([0.0964626, 0.0964626, 0.0964626, ..., 0.0964626, 0.0964626, 0.0964626], dtype=float32)),
('blocks.layers.2.sa.heads.0.value.gamma',
 array([4.40481, 4.40481, 4.40481, ..., 4.40481, 4.40481, 4.40481], 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([[0.0692456, -0.152191, 0.0473618, ..., -0.0918176, -0.0458712, -0.109515],
        [0.0421885, 0.0900879, 0.146135, ..., -0.142192, 0.0980739, -0.0478441],
        [0.117787, -0.0809765, 0.0409004, ..., -0.143661, -0.0568442, -0.102467],
        ...,
        [0.0358875, -0.0703686, 0.0566876, ..., -0.0736499, -0.0379242, 0.0992749],
        [-0.141088, 0.109765, 0.101046, ..., -0.0866638, -0.043984, -0.0416453],
        [-0.118056, 0.0740156, 0.136173, ..., 0.151482, -0.126186, 0.0466522]], dtype=float32)),
('blocks.layers.2.sa.heads.1.key.norm.bias',
 array([6.97978e-06, 1.33576e-05, -2.81794e-07, ..., 5.44192e-06, 7.65162e-06, -4.50031e-06], dtype=float32)),
('blocks.layers.2.sa.heads.1.key.norm.weight',
 array([1.06399, 1.01841, 1.14897, ..., 0.943814, 1.10382, 0.972624], dtype=float32)),
('blocks.layers.2.sa.heads.1.key.beta',
 array([0.0904466, 0.0904466, 0.0904466, ..., 0.0904466, 0.0904466, 0.0904466], dtype=float32)),
('blocks.layers.2.sa.heads.1.key.gamma',
 array([3.59072, 3.59072, 3.59072, ..., 3.59072, 3.59072, 3.59072], dtype=float32)),
('blocks.layers.2.sa.heads.1.query.weight',
 array([[0.121897, -0.144629, -0.149298, ..., -0.149023, 0.101947, 0.0544696],
        [0.0650016, -0.062889, -0.107791, ..., 0.0483493, 0.111136, 0.0626169],
        [0.107276, 0.0926742, -0.109082, ..., 0.116003, -0.0784151, 0.0440781],
        ...,
        [-0.0982632, 0.117239, -0.112012, ..., -0.0704885, -0.0581643, -0.0593274],
        [-0.0644367, -0.094558, -0.0724628, ..., -0.08628, 0.0600008, -0.103721],
        [-0.0969137, 0.0706074, -0.131439, ..., 0.0473227, -0.0637746, 0.147366]], dtype=float32)),
('blocks.layers.2.sa.heads.1.query.norm.bias',
 array([0.111516, 0.0190729, -0.143294, ..., -0.168759, 0.0576426, 0.0384983], dtype=float32)),
('blocks.layers.2.sa.heads.1.query.norm.weight',
 array([0.958041, 1.16743, 1.09706, ..., 1.01335, 1.06424, 1.11527], dtype=float32)),
('blocks.layers.2.sa.heads.1.query.beta',
 array([0.0895312, 0.0895312, 0.0895312, ..., 0.0895312, 0.0895312, 0.0895312], dtype=float32)),
('blocks.layers.2.sa.heads.1.query.gamma',
 array([4.51008, 4.51008, 4.51008, ..., 4.51008, 4.51008, 4.51008], dtype=float32)),
('blocks.layers.2.sa.heads.1.value.weight',
 array([[-0.0227989, -0.0208801, -0.0275896, ..., -0.0272948, 0.0241305, 0.0375896],
        [-0.027223, -0.0300616, -0.0273855, ..., -0.0310028, 0.0223202, -0.0358201],
        [-0.0366966, 0.0214995, 0.0275354, ..., 0.0236722, 0.0250659, -0.0196484],
        ...,
        [-0.0288439, 0.0284384, -0.027203, ..., -0.0251149, 0.0287761, 0.0237235],
        [-0.0340191, 0.0264391, 0.0227656, ..., -0.0286073, 0.0262409, -0.0228045],
        [-0.0375736, -0.021514, 0.0303186, ..., -0.0358176, -0.0289923, 0.0292666]], dtype=float32)),
('blocks.layers.2.sa.heads.1.value.norm.bias',
 array([0.206938, -0.0588094, 0.0719428, ..., 0.0905598, 0.108144, -0.166008], dtype=float32)),
('blocks.layers.2.sa.heads.1.value.norm.weight',
 array([0.827019, 0.969317, 0.921273, ..., 0.989789, 0.928329, 1.21796], dtype=float32)),
('blocks.layers.2.sa.heads.1.value.beta',
 array([0.0276416, 0.0276416, 0.0276416, ..., 0.0276416, 0.0276416, 0.0276416], dtype=float32)),
('blocks.layers.2.sa.heads.1.value.gamma',
 array([4.56006, 4.56006, 4.56006, ..., 4.56006, 4.56006, 4.56006], 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([[-0.0211862, 0.0148137, 0.0212719, ..., 0.0296384, 0.0260528, 0.0201666],
        [0.0197955, 0.025061, -0.0223159, ..., -0.0152515, 0.00995898, -0.0148582],
```

```
        [0.014668, -0.0244991, -0.0201105, ..., 0.015187, -0.0155624, -0.0190911],
        ...,
        [0.0229931, -0.0192614, 0.0264302, ..., 0.0248741, -0.0159296, -0.0209951],
        [0.023689, -0.0191933, 0.0221, ..., -0.012014, -0.0202688, 0.0221724],
        [0.0253952, 0.0182392, 0.0258797, ..., -0.0177745, -0.0120604, 0.0148882]], dtype=float32)),
('blocks.layers.2.sa.heads.2.key.norm.bias',
 array([-4.40767e-06, 7.12443e-06, -2.94991e-06, ..., 3.79545e-06, -1.53153e-06, -2.7046e-06], dtype=float32)),
('blocks.layers.2.sa.heads.2.key.norm.weight',
 array([0.969466, 0.979173, 0.919216, ..., 0.949178, 0.963322, 0.944556], dtype=float32)),
('blocks.layers.2.sa.heads.2.key.beta',
 array([0.0192081, 0.0192081, 0.0192081, ..., 0.0192081, 0.0192081, 0.0192081], dtype=float32)),
('blocks.layers.2.sa.heads.2.key.gamma',
 array([3.48542, 3.48542, 3.48542, ..., 3.48542, 3.48542, 3.48542], dtype=float32)),
('blocks.layers.2.sa.heads.2.query.weight',
 array([[0.015105, -0.0227698, 0.0189, ..., -0.0131805, -0.0263941, 0.0202842],
        [0.015713, 0.0252021, -0.0199871, ..., -0.0174195, -0.0204306, 0.024131],
        [0.0241698, -0.0230967, 0.018449, ..., -0.0229188, -0.0167529, -0.0152243],
        ...,
        [0.0203958, 0.0191604, 0.0226803, ..., -0.0255608, 0.0200908, -0.0228235],
        [-0.0175774, 0.0145, 0.0188576, ..., -0.0156479, -0.0178541, -0.0157711],
        [-0.0184612, -0.0203535, -0.0153801, ..., 0.0196377, 0.0239799, 0.024805]], dtype=float32)),
('blocks.layers.2.sa.heads.2.query.norm.bias',
 array([0.103507, 0.026799, 0.0799047, ..., -0.0818618, -0.00785794, 0.0715289], dtype=float32)),
('blocks.layers.2.sa.heads.2.query.norm.weight',
 array([0.913824, 1.00277, 0.903392, ..., 0.974943, 0.856806, 1.09833], dtype=float32)),
('blocks.layers.2.sa.heads.2.query.beta',
 array([0.0191371, 0.0191371, 0.0191371, ..., 0.0191371, 0.0191371, 0.0191371], dtype=float32)),
('blocks.layers.2.sa.heads.2.query.gamma',
 array([3.66141, 3.66141, 3.66141, ..., 3.66141, 3.66141, 3.66141], dtype=float32)),
('blocks.layers.2.sa.heads.2.value.weight',
 array([[-0.120883, -0.222144, 0.178228, ..., 0.152371, 0.117088, -0.141359],
        [0.132596, -0.11531, 0.203109, ..., -0.155163, 0.21439, -0.124738],
        [-0.226489, 0.180117, 0.226656, ..., 0.114585, -0.158749, -0.124087],
        ...,
        [0.130647, -0.157652, 0.187429, ..., 0.137145, 0.225717, 0.232008],
        [0.159771, 0.122235, 0.161238, ..., 0.205877, -0.214564, 0.126017],
        [0.144179, -0.138402, 0.134344, ..., 0.12159, -0.144884, 0.193326]], dtype=float32)),
('blocks.layers.2.sa.heads.2.value.norm.bias',
 array([0.0386308, -0.100007, 0.0526956, ..., -0.123868, 0.105253, -0.0946603], dtype=float32)),
('blocks.layers.2.sa.heads.2.value.norm.weight',
 array([1.01337, 1.06945, 0.931094, ..., 1.0247, 0.986773, 1.02099], dtype=float32)),
('blocks.layers.2.sa.heads.2.value.beta',
 array([0.171507, 0.171507, 0.171507, ..., 0.171507, 0.171507, 0.171507], dtype=float32)),
('blocks.layers.2.sa.heads.2.value.gamma',
 array([4.29842, 4.29842, 4.29842, ..., 4.29842, 4.29842, 4.29842], 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([[-0.145057, 0.143447, 0.153049, ..., 0.198429, 0.144465, -0.108554],
        [-0.169583, 0.128322, 0.204127, ..., -0.119876, -0.204763, 0.204705],
        [0.212979, -0.160316, -0.130891, ..., 0.117739, -0.166108, 0.177243],
        ...,
        [-0.211455, -0.20643, 0.133171, ..., -0.200817, 0.132387, -0.156868],
        [0.128272, -0.156903, -0.12848, ..., -0.131268, -0.129967, -0.141209],
        [0.0960882, -0.0953572, 0.102624, ..., -0.115067, -0.139232, -0.130098]], dtype=float32)),
('blocks.layers.2.sa.heads.3.key.norm.bias',
 array([-2.66755e-05, -4.73927e-06, -1.76387e-06, ..., -3.98835e-05, -4.32979e-05, -2.10643e-07], dtype=float32)),
('blocks.layers.2.sa.heads.3.key.norm.weight',
 array([1.08597, 0.922931, 1.08858, ..., 1.04004, 1.08017, 1.04394], dtype=float32)),
('blocks.layers.2.sa.heads.3.key.beta',
 array([0.151164, 0.151164, 0.151164, ..., 0.151164, 0.151164, 0.151164], dtype=float32)),
('blocks.layers.2.sa.heads.3.key.gamma',
 array([5.04749, 5.04749, 5.04749, ..., 5.04749, 5.04749, 5.04749], dtype=float32)),
('blocks.layers.2.sa.heads.3.query.weight',
 array([[-0.142751, -0.139451, 0.18904, ..., 0.0994231, 0.128607, 0.169043],
        [-0.107293, 0.188527, -0.111921, ..., 0.0989829, 0.145818, 0.143964],
        [-0.212143, -0.206453, 0.171789, ..., -0.1861, -0.0917208, 0.1592],
        ...,
        [0.168832, -0.113321, 0.187626, ..., 0.195598, 0.111976, 0.16027],
        [0.129725, 0.212199, 0.120627, ..., -0.16246, -0.207983, -0.0963606],
        [0.100253, 0.126697, -0.201686, ..., 0.140976, 0.0903835, 0.125185]], dtype=float32)),
('blocks.layers.2.sa.heads.3.query.norm.bias',
 array([-0.0551932, -0.0436891, 0.0798272, ..., 0.00803579, 0.0552877, -0.0322302], dtype=float32)),
('blocks.layers.2.sa.heads.3.query.norm.weight',
 array([0.848662, 0.991568, 1.08746, ..., 1.09935, 1.01382, 1.19657], dtype=float32)),
('blocks.layers.2.sa.heads.3.query.beta',
 array([0.152243, 0.152243, 0.152243, ..., 0.152243, 0.152243, 0.152243], dtype=float32)),
('blocks.layers.2.sa.heads.3.query.gamma',
 array([4.68981, 4.68981, 4.68981, ..., 4.68981, 4.68981, 4.68981], dtype=float32)),
('blocks.layers.2.sa.heads.3.value.weight',
 array([[-0.0310527, 0.0729361, -0.0547734, ..., -0.0812509, 0.0611145, 0.0476268],
        [-0.0650665, 0.0257898, -0.019614, ..., 0.0404917, 0.029288, -0.0727162],
        [-0.0733281, -0.042169, -0.0239916, ..., 0.0333355, 0.095263, -0.0129504],
        ...,
        [-0.0309312, 0.0197065, 0.0631463, ..., -0.0287697, 0.0111879, -0.0128541],
```

```
        [-0.0806123, 0.0289893, -0.0415727, ..., -0.0168582, -0.0893888, 0.0191024],
        [-0.0186998, -0.0626435, -0.0309932, ..., -0.0159075, 0.063007, -0.0873265]], dtype=float32)),
('blocks.layers.2.sa.heads.3.value.norm.bias',
 array([-0.051713, 0.100066, 0.053713, ..., 0.072152, -0.124783, -0.0341713], dtype=float32)),
('blocks.layers.2.sa.heads.3.value.norm.weight',
 array([0.986752, 0.923019, 0.861705, ..., 0.767729, 1.14721, 0.920813], dtype=float32)),
('blocks.layers.2.sa.heads.3.value.beta',
 array([0.043986, 0.043986, 0.043986, ..., 0.043986, 0.043986, 0.043986], dtype=float32)),
('blocks.layers.2.sa.heads.3.value.gamma',
 array([4.01467, 4.01467, 4.01467, ..., 4.01467, 4.01467, 4.01467], 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([[-0.000953435, -0.000780821, -7.7243e-05, ..., -0.000768685, -0.000757532, 8.07868e-05],
        [-0.000160472, -0.000157995, 0.000995562, ..., 0.00107023, -0.000971746, 0.000608585],
        [-0.000410266, 0.000747528, -0.000762716, ..., -0.000903033, -0.000281393, -0.000300584],
        ...,
        [0.00103036, 0.000168996, -0.000272987, ..., -0.000996202, -0.000756342, -0.000250929],
        [-0.000621157, 0.000181385, 0.000404993, ..., -0.000566001, 0.000108917, -0.000623773],
        [-0.000762515, -0.00050741, -0.000976115, ..., 0.00093275, 0.000131841, 0.000758608]], dtype=float32)),
('blocks.layers.2.sa.proj.bias',
 array([-0.177545, 0.131829, -0.0496801, ..., 0.505729, 0.112964, 0.197203], dtype=float32)),
('blocks.layers.2.sa.proj.norm.bias',
 array([-0.0279902, -0.0377726, 0.141205, ..., 0.0743717, 0.148093, -0.249083], dtype=float32)),
('blocks.layers.2.sa.proj.norm.weight',
 array([1.01811, 0.989176, 0.812918, ..., 0.913393, 1.07549, 1.06717], dtype=float32)),
('blocks.layers.2.sa.proj.beta',
 array([0.00084887, 0.00084887, 0.00084887, ..., 0.00084887, 0.00084887, 0.00084887], dtype=float32)),
('blocks.layers.2.sa.proj.gamma',
 array([4.76249, 4.76249, 4.76249, ..., 0.0835745, 0.142052, 0.0203527], dtype=float32)),
('blocks.layers.2.ffwd.net.layers.0.weight',
 array([[-0.0336482, -0.0117413, 0.0278904, ..., 0.0242187, 0.0356472, 0.0368042],
        [0.00796137, -2.44399e-05, 0.0241869, ..., 0.0243887, 0.0225557, 0.022262],
        [-0.0196678, 0.0350487, -0.0248911, ..., -0.018741, 0.0188241, -0.00581687],
        ...,
        [0.0360474, 0.0281902, -0.02507, ..., 0.0252712, -0.0247471, -0.0243748],
        [0.0358069, -0.0250657, -0.0231554, ..., 0.0233596, 0.0357525, 0.0368825],
        [-0.0335181, 0.0223524, -0.0248295, ..., -0.0312874, 0.0346531, -0.0143622]], dtype=float32)),
('blocks.layers.2.ffwd.net.layers.0.bias',
 array([0.143316, 0.0982513, -0.0177486, ..., 0.0854443, 0.0291045, 0.0241355], dtype=float32)),
('blocks.layers.2.ffwd.net.layers.0.norm.bias',
 array([0.0808973, -0.0429652, -0.0765387, ..., 0.0851981, 0.0433614, 0.0594383], dtype=float32)),
('blocks.layers.2.ffwd.net.layers.0.norm.weight',
 array([0.882728, 0.846142, 1.04245, ..., 1.09825, 0.861775, 1.19135], dtype=float32)),
('blocks.layers.2.ffwd.net.layers.0.beta',
 array([0.022321, 0.022321, 0.022321, ..., 0.022321, 0.022321, 0.022321], dtype=float32)),
('blocks.layers.2.ffwd.net.layers.0.gamma',
 array([4.82322, 4.82322, 4.82322, ..., 0, 0, 0], dtype=float32)),
('blocks.layers.2.ffwd.net.layers.2.weight',
 array([[-0.00322417, 0.00327778, 0.00321422, ..., -0.00206323, 0.00132941, -0.00145515],
        [-0.00178515, -0.00123403, 0.00178665, ..., -0.00183452, 0.00132142, -0.00127342],
        [-0.0012711, 0.00188362, 0.00321905, ..., -0.00217671, 0.0018848, 0.00144677],
        ...,
        [0.00205895, 0.00131706, 0.000778739, ..., -0.00134653, -0.00186182, -0.00232169],
        [-0.00289738, 0.00193766, -0.00131925, ..., -0.00322478, -0.0017508, 0.0013399],
        [-0.00208433, -0.00145969, -0.00206093, ..., -0.00174829, 0.00194854, 0.00290636]], dtype=float32)),
('blocks.layers.2.ffwd.net.layers.2.bias',
 array([-0.0308847, 0.124839, -0.115651, ..., -0.333841, -0.0815619, 0.145164], dtype=float32)),
('blocks.layers.2.ffwd.net.layers.2.norm.bias',
 array([0.0650113, 0.0559113, -0.135714, ..., 0.156596, -0.00719056, -0.143033], dtype=float32)),
('blocks.layers.2.ffwd.net.layers.2.norm.weight',
 array([0.965741, 0.92258, 0.719706, ..., 0.838636, 0.999393, 1.12834], dtype=float32)),
('blocks.layers.2.ffwd.net.layers.2.beta',
 array([0.00219148, 0.00219148, 0.00219148, ..., 0.00219148, 0.00219148, 0.00219148], dtype=float32)),
('blocks.layers.2.ffwd.net.layers.2.gamma',
 array([13.9072, 13.9072, 13.9072, ..., -0.201851, 0.227636, 0.0272021], dtype=float32)),
('blocks.layers.2.ln1.bias',
 array([-0.0671132, -0.0476957, 0.0787136, ..., 0.0303333, 0.0237535, -0.129358], dtype=float32)),
('blocks.layers.2.ln1.weight',
 array([0.939338, 1.04774, 0.921071, ..., 0.916474, 1.02762, 0.968636], dtype=float32)),
('blocks.layers.2.ln2.bias',
 array([0.0616367, -0.0587703, -0.0861588, ..., 0.069126, 0.0385237, 0.0291468], dtype=float32)),
('blocks.layers.2.ln2.weight',
 array([0.879131, 0.850311, 1.04511, ..., 1.07225, 0.91555, 1.22108], dtype=float32)),
('blocks.layers.3.sa.heads.0.key.weight',
 array([[-0.223916, 0.271021, -0.276548, ..., 0.270859, -0.186558, 0.231048],
        [-0.28981, 0.217989, 0.300584, ..., -0.206439, 0.201568, 0.253087],
        [-0.291728, -0.189179, -0.225075, ..., 0.197717, -0.28621, 0.291631],
        ...,
        [-0.212016, 0.258124, 0.211297, ..., 0.232774, -0.217013, 0.303323],
        [-0.245613, 0.235566, 0.200906, ..., -0.226734, -0.201213, 0.207168],
        [0.297456, -0.300693, -0.24571, ..., -0.212871, 0.232851, -0.238103]], dtype=float32)),
('blocks.layers.3.sa.heads.0.key.norm.bias',
 array([-4.82733e-05, 5.33913e-05, 4.69446e-05, ..., 6.90406e-05, 0.000155294, -5.4705e-05], dtype=float32)),
('blocks.layers.3.sa.heads.0.key.norm.weight',
```

```
 array([1.01509, 1.12094, 1.00954, ..., 1.06244, 0.636905, 1.04209], dtype=float32)),
('blocks.layers.3.sa.heads.0.key.beta',
 array([0.243609, 0.243609, 0.243609, ..., 0.243609, 0.243609, 0.243609], dtype=float32)),
('blocks.layers.3.sa.heads.0.key.gamma',
 array([5.20419, 5.20419, 5.20419, ..., 5.20419, 5.20419, 5.20419], dtype=float32)),
('blocks.layers.3.sa.heads.0.query.weight',
 array([[0.297055, 0.253898, -0.303725, ..., -0.222551, 0.252155, -0.281773],
        [-0.188719, -0.198047, 0.288424, ..., 0.229367, 0.205542, -0.284594],
        [-0.288892, 0.271186, 0.297186, ..., 0.244481, 0.242236, 0.234106],
        ...,
        [0.274951, -0.248051, -0.25041, ..., 0.21093, -0.237585, 0.238201],
        [0.184215, 0.252704, -0.22162, ..., 0.285987, 0.276302, 0.225407],
        [-0.184356, 0.296762, -0.279915, ..., -0.230144, 0.251148, 0.230297]], dtype=float32)),
('blocks.layers.3.sa.heads.0.query.norm.bias',
 array([0.162657, -0.474193, 0.297284, ..., 0.195995, -0.328407, -0.248835], dtype=float32)),
('blocks.layers.3.sa.heads.0.query.norm.weight',
 array([0.841151, 0.645461, 0.838516, ..., 0.97273, 0.891377, 1.20936], dtype=float32)),
('blocks.layers.3.sa.heads.0.query.beta',
 array([0.243587, 0.243587, 0.243587, ..., 0.243587, 0.243587, 0.243587], dtype=float32)),
('blocks.layers.3.sa.heads.0.query.gamma',
 array([5.52915, 5.52915, 5.52915, ..., 5.52915, 5.52915, 5.52915], dtype=float32)),
('blocks.layers.3.sa.heads.0.value.weight',
 array([[0.016137, 0.0057691, 0.0485364, ..., -0.0099503, 0.0071376, 0.00422907],
        [0.0157737, -0.0209382, 0.0297648, ..., -0.0165767, -0.0120659, -0.0174176],
        [-0.0253962, 0.001944, -0.00878727, ..., -0.020284, 0.021964, -0.00636472],
        ...,
        [-0.0564451, -0.0267467, 0.0278821, ..., -0.00450891, 0.0474142, -0.00580153],
        [0.0535713, 0.0177457, -0.046506, ..., 0.00518505, -0.00763173, 0.00511059],
        [0.0535394, -0.0484043, 0.00820591, ..., 0.0423765, -0.00914041, -0.0421512]], dtype=float32)),
('blocks.layers.3.sa.heads.0.value.norm.bias',
 array([-0.0284509, -0.0576503, 0.0423483, ..., -0.052778, -0.0662249, -0.000768885], dtype=float32)),
('blocks.layers.3.sa.heads.0.value.norm.weight',
 array([0.966412, 0.973414, 0.958124, ..., 1.15808, 0.896532, 0.930732], dtype=float32)),
('blocks.layers.3.sa.heads.0.value.beta',
 array([0.0170099, 0.0170099, 0.0170099, ..., 0.0170099, 0.0170099, 0.0170099], dtype=float32)),
('blocks.layers.3.sa.heads.0.value.gamma',
 array([4.31957, 4.31957, 4.31957, ..., 4.31957, 4.31957, 4.31957], 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([[0.0976342, 0.0984636, -0.0945717, ..., -0.0764702, -0.1837, -0.168793],
        [-0.164633, 0.143182, -0.116405, ..., 0.0921462, 0.1323, 0.103812],
        [-0.0677842, 0.0957363, 0.181573, ..., 0.102747, -0.140016, -0.142939],
        ...,
        [0.0732156, 0.090951, 0.181985, ..., 0.0695885, 0.150422, 0.124472],
        [0.0819728, -0.160694, -0.0890542, ..., 0.137736, 0.101072, -0.0675593],
        [0.139077, -0.108968, -0.107344, ..., -0.168877, -0.0764583, 0.163657]], dtype=float32)),
('blocks.layers.3.sa.heads.1.key.norm.bias',
 array([-0.000443734, -0.0001369, -0.000395154, ..., -3.07043e-05, 6.90796e-05, -0.000139972], dtype=float32)),
('blocks.layers.3.sa.heads.1.key.norm.weight',
 array([1.07322, 1.01087, 1.11229, ..., 0.718845, 0.699349, 0.546005], dtype=float32)),
('blocks.layers.3.sa.heads.1.key.beta',
 array([0.121958, 0.121958, 0.121958, ..., 0.121958, 0.121958, 0.121958], dtype=float32)),
('blocks.layers.3.sa.heads.1.key.gamma',
 array([5.42237, 5.42237, 5.42237, ..., 5.42237, 5.42237, 5.42237], dtype=float32)),
('blocks.layers.3.sa.heads.1.query.weight',
 array([[-0.133536, -0.136503, -0.182682, ..., 0.138422, -0.165043, 0.138714],
        [-0.0659944, 0.130523, -0.172463, ..., -0.0836118, 0.145939, -0.172996],
        [0.133818, 0.0791008, 0.108216, ..., 0.16242, 0.110616, 0.0753005],
        ...,
        [-0.124784, -0.154047, 0.16539, ..., -0.0668161, 0.134346, -0.0941774],
        [-0.0943022, -0.109139, 0.138885, ..., -0.124726, 0.102544, -0.178799],
        [-0.14205, 0.170761, -0.0670118, ..., -0.0975436, 0.102377, 0.0995452]], dtype=float32)),
('blocks.layers.3.sa.heads.1.query.norm.bias',
 array([-0.0171755, -0.103379, 0.0249411, ..., -0.0415328, 0.125627, -0.0495897], dtype=float32)),
('blocks.layers.3.sa.heads.1.query.norm.weight',
 array([1.16458, 0.778895, 1.09611, ..., 1.36371, 0.699646, 1.13361], dtype=float32)),
('blocks.layers.3.sa.heads.1.query.beta',
 array([0.12252, 0.12252, 0.12252, ..., 0.12252, 0.12252, 0.12252], dtype=float32)),
('blocks.layers.3.sa.heads.1.query.gamma',
 array([4.73137, 4.73137, 4.73137, ..., 4.73137, 4.73137, 4.73137], dtype=float32)),
('blocks.layers.3.sa.heads.1.value.weight',
 array([[-0.100342, 0.0837224, -0.089895, ..., 0.143475, -0.0910355, -0.138974],
        [0.150014, -0.101149, 0.0926754, ..., -0.104711, -0.177622, 0.0915914],
        [0.0916937, 0.142952, -0.150849, ..., 0.170148, -0.114537, 0.130596],
        ...,
        [0.0761373, -0.160954, -0.0702426, ..., 0.162793, -0.143804, 0.147741],
        [0.165123, -0.0689369, 0.150098, ..., 0.147036, -0.11178, 0.0765894],
        [0.0727818, -0.153327, -0.178394, ..., -0.112168, -0.151872, 0.172193]], dtype=float32)),
('blocks.layers.3.sa.heads.1.value.norm.bias',
 array([0.111972, -0.0214319, 0.0342332, ..., 0.00653537, 0.0835422, 0.0106867], dtype=float32)),
('blocks.layers.3.sa.heads.1.value.norm.weight',
 array([0.888576, 0.985698, 0.900294, ..., 1.07182, 0.993863, 0.89091], dtype=float32)),
('blocks.layers.3.sa.heads.1.value.beta',
 array([0.123857, 0.123857, 0.123857, ..., 0.123857, 0.123857, 0.123857], dtype=float32)),
```

```
('blocks.layers.3.sa.heads.1.value.gamma',
 array([5.58755, 5.58755, 5.58755, ..., 5.58755, 5.58755, 5.58755], 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([[-0.227561, 0.280339, -0.279797, ..., -0.227114, 0.231259, -0.231351],
        [0.227059, 0.227196, 0.225954, ..., 0.2398, 0.227184, 0.227647],
        [-0.226829, 0.225894, 0.252442, ..., -0.230525, -0.237425, 0.263281],
        ...,
        [0.275672, 0.261609, -0.250367, ..., -0.226853, 0.226566, 0.236266],
        [0.227421, 0.240224, 0.228962, ..., 0.227617, 0.256192, 0.278379],
        [-0.227207, 0.252852, 0.230627, ..., 0.269849, 0.227424, -0.275062]], dtype=float32)),
('blocks.layers.3.sa.heads.2.key.norm.bias',
 array([1.89873e-05, 6.80934e-05, -0.000210049, ..., 0.000215683, 1.59924e-05, -1.41152e-05], dtype=float32)),
('blocks.layers.3.sa.heads.2.key.norm.weight',
 array([1.11007, 0.958227, 1.24678, ..., 1.18885, 0.704242, 0.86073], dtype=float32)),
('blocks.layers.3.sa.heads.2.key.beta',
 array([0.241315, 0.241315, 0.241315, ..., 0.241315, 0.241315, 0.241315], dtype=float32)),
('blocks.layers.3.sa.heads.2.key.gamma',
 array([5.69822, 5.69822, 5.69822, ..., 5.69822, 5.69822, 5.69822], dtype=float32)),
('blocks.layers.3.sa.heads.2.query.weight',
 array([[-0.226892, -0.261431, -0.22786, ..., -0.22802, 0.227538, 0.257181],
        [-0.266244, 0.231537, -0.2513, ..., 0.232169, -0.255399, -0.227952],
        [0.228566, -0.227866, 0.227531, ..., 0.229165, -0.247633, -0.254937],
        ...,
        [-0.229039, -0.267216, -0.22959, ..., -0.239399, 0.26717, 0.227811],
        [0.228765, -0.22773, 0.228503, ..., 0.2316, -0.228241, -0.262611],
        [-0.226743, 0.243484, 0.276935, ..., -0.227744, 0.264622, 0.236377]], dtype=float32)),
('blocks.layers.3.sa.heads.2.query.norm.bias',
 array([-0.546623, -0.474802, -0.244374, ..., 0.17389, 0.0338363, -0.10568], dtype=float32)),
('blocks.layers.3.sa.heads.2.query.norm.weight',
 array([1.64832, 0.702848, 1.42325, ..., 1.58178, 1.36048, 0.870391], dtype=float32)),
('blocks.layers.3.sa.heads.2.query.beta',
 array([0.242712, 0.242712, 0.242712, ..., 0.242712, 0.242712, 0.242712], dtype=float32)),
('blocks.layers.3.sa.heads.2.query.gamma',
 array([5.32776, 5.32776, 5.32776, ..., 5.32776, 5.32776, 5.32776], dtype=float32)),
('blocks.layers.3.sa.heads.2.value.weight',
 array([[-0.0227923, 0.0871727, -0.0170504, ..., -0.0349972, -0.0557801, -0.0458547],
        [-0.0500761, 0.0842572, -0.0915066, ..., -0.0179543, -0.0409188, -0.0646627],
        [0.0509451, -0.101607, 0.0542162, ..., -0.0574656, 0.0369963, 0.0211588],
        ...,
        [0.0567561, -0.0410958, 0.0473571, ..., 0.0406913, -0.0532907, -0.0634956],
        [0.0295716, -0.0608072, 0.02173, ..., -0.0247378, -0.0244555, 0.031526],
        [-0.0912525, -0.0542476, 0.0637456, ..., -0.0495875, -0.0232802, 0.0595243]], dtype=float32)),
('blocks.layers.3.sa.heads.2.value.norm.bias',
 array([0.0622427, -0.0617772, 0.0555218, ..., -0.0313082, 0.0405988, -0.0491843], dtype=float32)),
('blocks.layers.3.sa.heads.2.value.norm.weight',
 array([0.927665, 0.99219, 0.88386, ..., 0.853098, 0.580521, 1.09796], dtype=float32)),
('blocks.layers.3.sa.heads.2.value.beta',
 array([0.0575814, 0.0575814, 0.0575814, ..., 0.0575814, 0.0575814, 0.0575814], dtype=float32)),
('blocks.layers.3.sa.heads.2.value.gamma',
 array([6.23278, 6.23278, 6.23278, ..., 6.23278, 6.23278, 6.23278], 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([[-0.0926396, -0.113204, 0.0976017, ..., 0.0943035, 0.106103, -0.112987],
        [0.105439, 0.0952304, -0.0926863, ..., 0.106052, -0.0918448, -0.109429],
        [0.105489, -0.0924768, -0.114053, ..., 0.104526, -0.10542, -0.109309],
        ...,
        [0.0943324, -0.0900576, 0.105728, ..., 0.113142, -0.0935697, 0.0913857],
        [0.107981, 0.108702, 0.0975355, ..., 0.100115, 0.107703, -0.0940753],
        [-0.0976532, -0.0906327, 0.0902592, ..., 0.105801, -0.0935963, -0.0959817]], dtype=float32)),
('blocks.layers.3.sa.heads.3.key.norm.bias',
 array([-4.08399e-05, 0.000284808, 0.000142215, ..., -5.54934e-05, -4.5516e-05, 0.00012394], dtype=float32)),
('blocks.layers.3.sa.heads.3.key.norm.weight',
 array([0.963116, 0.756195, 0.668534, ..., 1.02741, 0.775109, 0.894354], dtype=float32)),
('blocks.layers.3.sa.heads.3.key.beta',
 array([0.101786, 0.101786, 0.101786, ..., 0.101786, 0.101786, 0.101786], dtype=float32)),
('blocks.layers.3.sa.heads.3.key.gamma',
 array([6.02686, 6.02686, 6.02686, ..., 6.02686, 6.02686, 6.02686], dtype=float32)),
('blocks.layers.3.sa.heads.3.query.weight',
 array([[0.107905, -0.0926668, 0.090417, ..., 0.10944, -0.105688, 0.105937],
        [-0.0900089, -0.0915872, -0.11004, ..., 0.10957, 0.103964, 0.101921],
        [-0.0944167, 0.0905906, 0.106253, ..., 0.10627, 0.106103, 0.090075],
        ...,
        [-0.110591, 0.105976, -0.107841, ..., -0.0944488, -0.100774, -0.0922852],
        [0.0926664, -0.108998, 0.0927489, ..., -0.105723, -0.10353, -0.105941],
        [0.104668, -0.105758, 0.0924751, ..., -0.0936243, 0.0923288, -0.108523]], dtype=float32)),
('blocks.layers.3.sa.heads.3.query.norm.bias',
```

```
 array([0.22569, -0.337989, -0.0885267, ..., 0.0836504, 0.0149936, -0.0148528], dtype=float32)),
('blocks.layers.3.sa.heads.3.query.norm.weight',
 array([0.762945, 0.727761, 1.03181, ..., 1.23844, 0.61656, 0.916937], dtype=float32)),
('blocks.layers.3.sa.heads.3.query.beta',
 array([0.101758, 0.101758, 0.101758, ..., 0.101758, 0.101758, 0.101758], dtype=float32)),
('blocks.layers.3.sa.heads.3.query.gamma',
 array([5.05596, 5.05596, 5.05596, ..., 5.05596, 5.05596, 5.05596], dtype=float32)),
('blocks.layers.3.sa.heads.3.value.weight',
 array([[-0.1535, 0.167009, -0.170757, ..., -0.201026, -0.105859, 0.209327],
        [-0.214836, -0.208916, -0.222643, ..., 0.134463, -0.205282, 0.16245],
        [-0.132095, 0.19624, 0.129126, ..., 0.187802, 0.225465, -0.128519],
        ...,
        [0.119271, -0.128031, -0.199488, ..., -0.202796, 0.161299, -0.13852],
        [-0.151105, -0.16313, 0.155088, ..., 0.207858, 0.121036, -0.207437],
        [0.195681, -0.126945, 0.109222, ..., -0.188129, 0.151314, -0.169704]], dtype=float32)),
('blocks.layers.3.sa.heads.3.value.norm.bias',
 array([-0.0275592, -0.0471268, -0.342253, ..., 0.104521, -6.42165e-05, 0.163815], dtype=float32)),
('blocks.layers.3.sa.heads.3.value.norm.weight',
 array([0.948246, 1.07121, 1.17526, ..., 1.00292, 0.826003, 1.09684], dtype=float32)),
('blocks.layers.3.sa.heads.3.value.beta',
 array([0.164064, 0.164064, 0.164064, ..., 0.164064, 0.164064, 0.164064], dtype=float32)),
('blocks.layers.3.sa.heads.3.value.gamma',
 array([5.7158, 5.7158, 5.7158, ..., 5.7158, 5.7158, 5.7158], 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([[-0.0405165, -0.0298563, 0.019896, ..., -0.0302522, 0.0395505, 0.0270441],
        [0.0363154, -0.0214717, -0.020374, ..., 0.0228209, -0.029186, 0.0228078],
        [-0.018744, -0.0368361, -0.0188484, ..., 0.0345663, 0.0274812, -0.0393289],
        ...,
        [-0.0203475, 0.0483057, 0.0542444, ..., 0.0352907, -0.0222431, -0.0404453],
        [0.034563, -0.0403862, 0.0278441, ..., 0.0482094, -0.0370204, -0.0243083],
        [-0.0357411, -0.0347298, 0.0201353, ..., 0.037389, -0.0241388, -0.0445552]], dtype=float32)),
('blocks.layers.3.sa.proj.bias',
 array([-0.236769, 0.285344, 0.0562842, ..., 0.349183, 0.0996531, -0.139339], dtype=float32)),
('blocks.layers.3.sa.proj.norm.bias',
 array([0.0695602, 0.0973191, -0.0943851, ..., 0.0935514, -0.0338869, 0.0349343], dtype=float32)),
('blocks.layers.3.sa.proj.norm.weight',
 array([0.866112, 1.10847, 0.919634, ..., 0.957016, 1.00307, 0.762937], dtype=float32)),
('blocks.layers.3.sa.proj.beta',
 array([0.0332023, 0.0332023, 0.0332023, ..., 0.0332023, 0.0332023, 0.0332023], dtype=float32)),
('blocks.layers.3.sa.proj.gamma',
 array([4.22984, 4.22984, 4.22984, ..., 0.0898946, -0.0332646, -0.0486258], dtype=float32)),
('blocks.layers.3.ffwd.net.layers.0.weight',
 array([[0.0819429, -0.0128033, -0.045746, ..., -0.0710292, -0.0122253, 0.0956372],
        [0.0835028, -0.0154034, -0.0155366, ..., 0.0564019, 0.00903225, -0.0538648],
        [0.0609073, 0.0827211, 0.0334789, ..., -0.107602, -0.0122394, 0.0331848],
        ...,
        [-0.0354665, -0.0347147, -0.0113224, ..., -0.106858, 0.0268657, -0.0920467],
        [-0.00938938, -0.0350877, -0.0250139, ..., 0.0473458, -0.0215844, -0.057206],
        [0.0270559, -0.0352497, -0.0155412, ..., -0.0127717, 0.0134654, 0.105306]], dtype=float32)),
('blocks.layers.3.ffwd.net.layers.0.bias',
 array([-0.0812247, -0.0253819, -0.336128, ..., 0.0264735, 0.0412333, 0.113603], dtype=float32)),
('blocks.layers.3.ffwd.net.layers.0.norm.bias',
 array([-0.112745, 0.129399, -0.143837, ..., 0.339473, 0.147512, -0.0469358], dtype=float32)),
('blocks.layers.3.ffwd.net.layers.0.norm.weight',
 array([0.943318, 0.996557, 0.771397, ..., 1.05481, 1.18041, 0.33439], dtype=float32)),
('blocks.layers.3.ffwd.net.layers.0.beta',
 array([0.0491226, 0.0491226, 0.0491226, ..., 0.0491226, 0.0491226, 0.0491226], dtype=float32)),
('blocks.layers.3.ffwd.net.layers.0.gamma',
 array([6.32943, 6.32943, 6.32943, ..., 0, 0, 0], dtype=float32)),
('blocks.layers.3.ffwd.net.layers.2.weight',
 array([[0.0196241, 0.0585066, -0.0345372, ..., 0.0372231, 0.035612, -0.0509737],
        [-0.0571367, 0.0257122, 0.0248193, ..., 0.0514272, 0.063557, -0.0542863],
        [0.0572043, -0.0237642, 0.0406408, ..., -0.0562142, 0.0474468, 0.0556876],
        ...,
        [0.0551445, 0.0368881, 0.0623505, ..., 0.0184793, -0.0274255, 0.0477883],
        [-0.049856, -0.0566877, 0.0493129, ..., 0.0574323, 0.0551075, -0.0266973],
        [0.021415, -0.0569785, -0.0347071, ..., 0.0354154, -0.0433217, 0.0262592]], dtype=float32)),
('blocks.layers.3.ffwd.net.layers.2.bias',
 array([-0.0915776, 0.163911, -0.131598, ..., 0.383542, 0.0313572, -0.338458], dtype=float32)),
('blocks.layers.3.ffwd.net.layers.2.norm.bias',
 array([-0.0801262, 0.137373, 0.0347545, ..., -0.0785576, 0.00376761, -0.0493959], dtype=float32)),
('blocks.layers.3.ffwd.net.layers.2.norm.weight',
 array([0.956539, 0.946648, 0.844155, ..., 1.07325, 0.980241, 1.01682], dtype=float32)),
('blocks.layers.3.ffwd.net.layers.2.beta',
 array([0.0419804, 0.0419804, 0.0419804, ..., 0.0419804, 0.0419804, 0.0419804], dtype=float32)),
('blocks.layers.3.ffwd.net.layers.2.gamma',
 array([14.5634, 14.5634, 14.5634, ..., 0.127619, 0.120073, -0.106548], dtype=float32)),
('blocks.layers.3.ln1.bias',
 array([0.0226316, -0.121078, -0.284614, ..., 0.0339937, 0.00743196, 0.0847285], dtype=float32)),
('blocks.layers.3.ln1.weight',
 array([0.950498, 0.934256, 1.02315, ..., 1.22951, 0.629743, 0.982902], dtype=float32)),
('blocks.layers.3.ln2.bias',
 array([-0.128902, 0.106691, -0.170425, ..., 0.30981, 0.128258, -0.0605226], dtype=float32)),
```

```
 ('blocks.layers.3.ln2.weight',
  array([0.961863, 0.976864, 0.739212, ..., 0.883239, 1.21749, 0.294405], dtype=float32)),
 ('ln_f.bias',
  array([-0.107084, 0.144412, -0.043709, ..., 0.262063, 0.0294899, 0.00232565], dtype=float32)),
 ('ln_f.weight',
  array([1.02813, 0.93274, 0.944995, ..., 1.41795, 1.02816, 1.49907], dtype=float32)),
 ('lm_head.weight',
  array([[-0.542533, 0.548549, -0.570712, ..., 0.514647, -0.524416, 0.576259],
         [-0.589967, 0.585115, -0.567441, ..., 0.574367, -0.491216, 0.533877],
         [0.517043, -0.506139, 0.487949, ..., 0.589585, -0.573156, -0.570704],
         ...,
         [-0.488116, -0.57287, -0.539515, ..., -0.542775, -0.576967, 0.557261],
         [0.519569, 0.526767, 0.595124, ..., 0.524869, 0.495257, 0.503809],
         [0.563177, 0.515242, 0.525022, ..., 0.586133, -0.551134, -0.595954]], dtype=float32)),
 ('lm_head.bias',
  array([0.269059, 0.0507124, -0.444161, ..., 1.96958, -0.638336, 1.75739], dtype=float32)),
 ('lm_head.norm.bias',
  array([0.00154793, 0.0698593, 0.00440143, ..., 0.192655, 0.045572, 0.0413605], dtype=float32)),
 ('lm_head.norm.weight',
  array([0.917719, 0.945217, 1.00734, ..., 1.26668, 1.03557, 1.24], dtype=float32)),
 ('lm_head.beta',
  array([0.535704, 0.535704, 0.535704, ..., 0.535704, 0.535704, 0.535704], dtype=float32)),
 ('lm_head.gamma',
  array([2.73138, 2.73138, 2.73138, ..., -1.29008, -0.600934, -1.10448], dtype=float32))]
```

In [ ]:
```
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 [ ]:
```
# 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..


SRy,
SAlend my.
Burs pol, d baswe, hstieease, sigy;
AsuroLavis be.
ss!

Bho ikirer.

ALAblouUESAntowaplaius ved.
S:
Nodenchimer ts cantatVIroforseve.

Aponosan, hand hecomy, wa OUqcedrdeknd oth lom.
Icthar:

AnWrMbF il MXlce.
llOLII deweouloffreut ,
d sirsis kistosWAPpus u, so ngoor.

Fus, cheDl.
Kver dora'e hoMIyoue tcWelo hewiflaG, wiarmedce.

Noo t;
pgr or akeriurrfoit, ipearagEyoongit th.
m oroda LAKEfh.
AAder, Irig thedonare thid, s stoomuse.

LLAABendeatikewoise it,
Aloorg, att sedorgtE:
```

In [ ]: