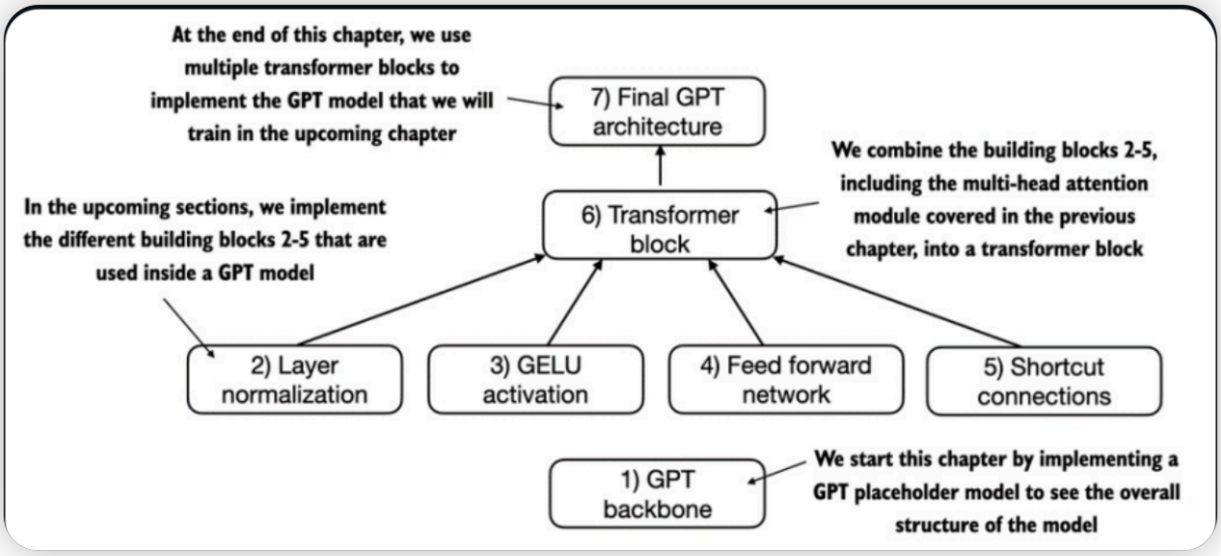
- 编码步骤：



  下文依次介绍 1-7。

GPT Backbone

代码 - DummyGPTModel

```
import torch
import torch.nn as nn
class DummyGPTModel(nn.Module):
    def __init__(self, cfg):
        super().__init__()
        # 标记和位置嵌入
        self.tok_emb = nn.Embedding(cfg["vocab_size"],
cfg["emb_dim"])
        self.pos_emb = nn.Embedding(cfg["context_length"],
cfg["emb_dim"])
        # Dropout
        self.drop_emb = nn.Dropout(cfg["drop_rate"])
        # Transformer 模块
        self.trf_blocks = nn.Sequential(
```

```
            *[DummyTransformerBlock(cfg) for _ in
range(cfg["n_layers"])])
        # 最终归一化
        self.final_norm = DummyLayerNorm(cfg["emb_dim"])
        # 线性输出层
        self.out_head = nn.Linear(
            cfg["emb_dim"], cfg["vocab_size"], bias=False)
    # 数据流
    def forward(self, in_idx):
        batch_size, seq_len = in_idx.shape
        tok_embeds = self.tok_emb(in_idx)
        pos_embeds = self.pos_emb(torch.arange(seq_len,
device=in_idx.device))
        x = tok_embeds + pos_embeds
        x = self.drop_emb(x)
        x = self.trf_blocks(x)
        x = self.final_norm(x)
        logits = self.out_head(x)
        return logits
```

Layer Normalization

[归一化]

代码 - LayerNorm

> Pytorch 其实封装了 LayerNorm 层，下面自己写的原理。

```
import torch
import torch.nn as nn

# 归一化 模块
class LayerNorm(nn.Module):
    def __init__(self, emb_dim):
        super().__init__()
```

```
        self.eps = 1e-5
        self.scale = nn.Parameter(torch.ones(emb_dim))
        self.shift = nn.Parameter(torch.zeros(emb_dim))

    def forward(self, x):
        mean = x.mean(dim=-1, keepdim=True)
        # unbiased=False，是除以 n
        ## 为了与GPT-2的归一化层兼容
                # 有的采用贝塞尔校正，是除以 n-1
        var = x.var(dim=-1, keepdim=True, unbiased=False)
        norm_x = (x - mean) / torch.sqrt(var + self.eps)
        return self.scale * norm_x + self.shift
```

Feed Forward Network

> 实现使用 GELU 激活函数的前馈网络。

激活函数

代码 - GELU

GELU函数（高斯误差线性单元函数）

$$GELU(x) \approx 0.5 \cdot x \cdot (1 + \tanh[\sqrt{(2/\pi)} \cdot (x + 0.044715 \cdot x^3)])$$

```
import torch
from torch import nn

# GELU 激活函数
class GELU(nn.Module):
    def __init__(self):
        super().__init__()
    def forward(self, x):
        return 0.5 * x * (1 + torch.tanh(
            torch.sqrt(torch.tensor(2.0 / torch.pi)) *
```

```
            (x + 0.044715 * torch.pow(x, 3))
        ))
```

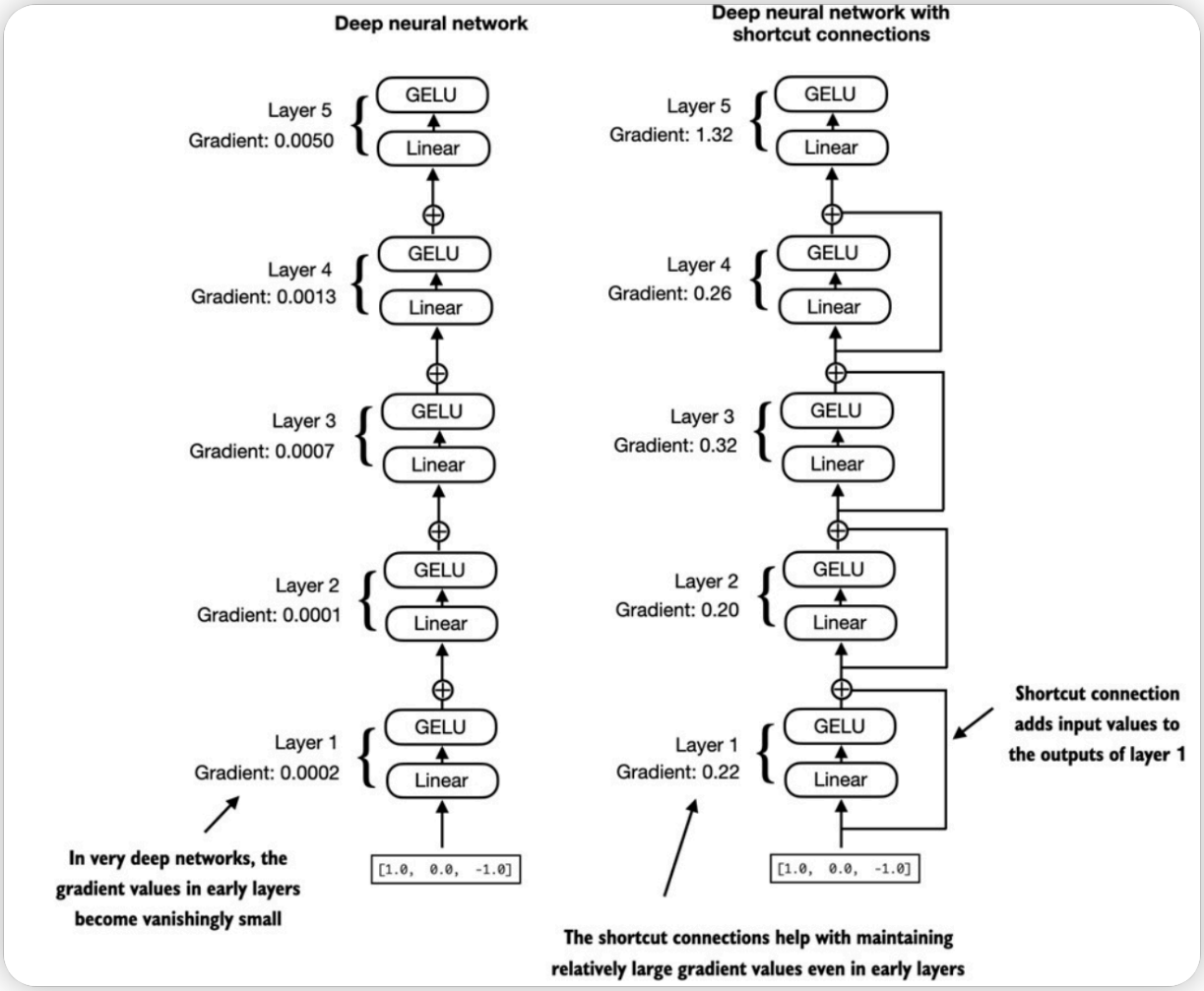
代码 - FeedForward

```
import torch.nn as nn

# FeedForward 前馈网络块
class FeedForward(nn.Module):
    def __init__(self, cfg):
        super().__init__()
        self.linear1 = nn.Linear(cfg["emb_dim"], cfg["emb_dim"] *
4)
        self.relu = nn.ReLU()
        self.linear2 = nn.Linear(cfg["emb_dim"] * 4,
cfg["emb_dim"])
        self.dropout = nn.Dropout(cfg["drop_rate"])

    def forward(self, x):
        x = self.relu(self.linear1(x))
        x = self.dropout(x)
        x = self.linear2(x)
        return x
```

Shortcut Connections

> 快捷连接（ResNet 残差网络提出），用于解决梯度消失问题：为梯度在网络中流动创造更短的备用路径，保证梯度的流动。

- 有无快捷连接的深度神经网络对比



代码 - 图例实现

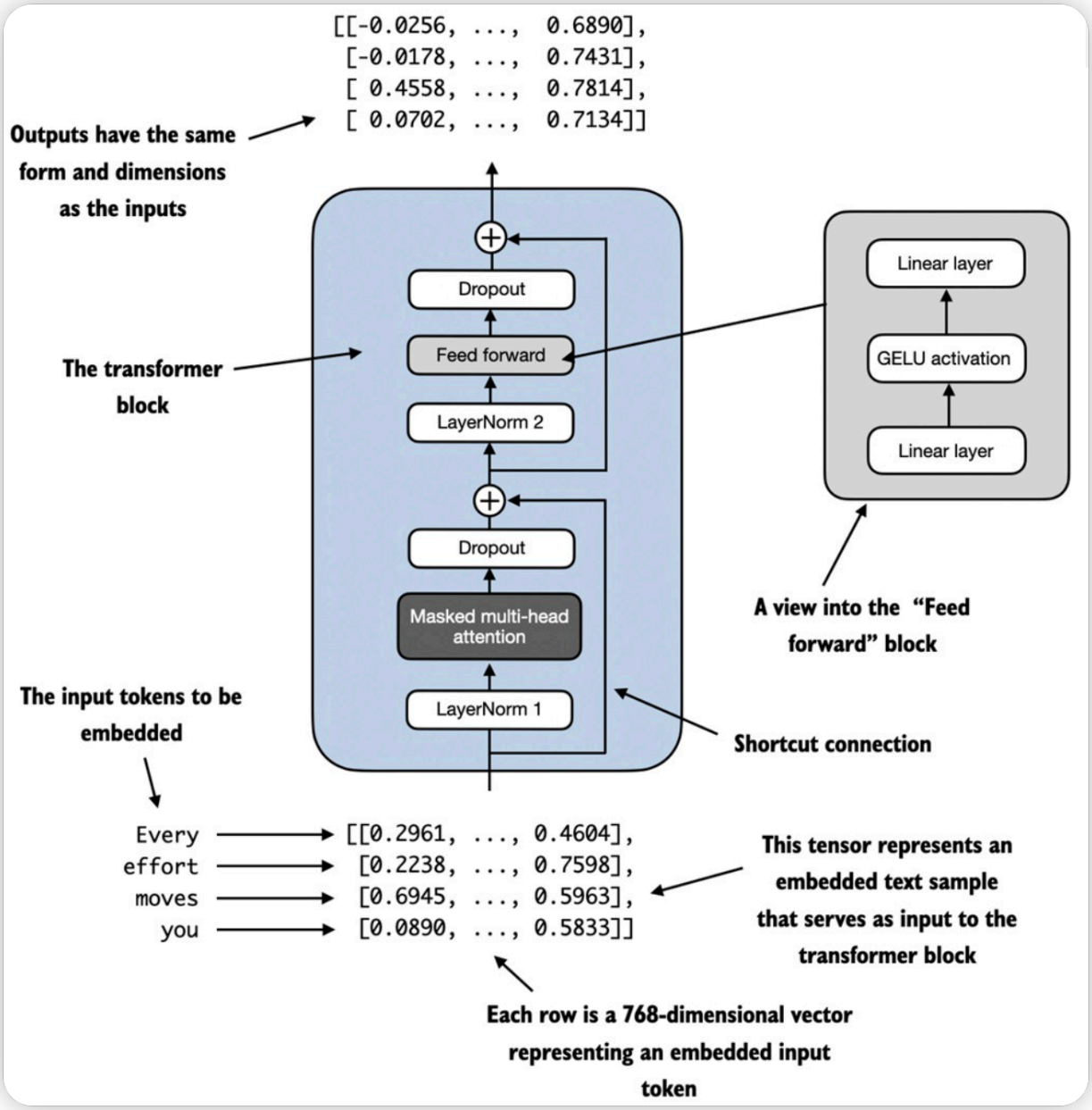
```
import torch.nn as nn
from torch.nn import GELU

class ExampleDeepNeuralNetwork(nn.Module):
    def __init__(self, layer_sizes, use_shortcut):
        super().__init__()
        self.use_shortcut = use_shortcut
        # Sequential 自动顺序执行;
        # ModuleList 手动指定执行;
        self.layers = nn.ModuleList([
            nn.Sequential(nn.Linear(layer_sizes[0],
```

```
layer_sizes[1]), GELU()),
            nn.Sequential(nn.Linear(layer_sizes[1],
layer_sizes[2]), GELU()),
            nn.Sequential(nn.Linear(layer_sizes[2],
layer_sizes[3]), GELU()),
            nn.Sequential(nn.Linear(layer_sizes[3],
layer_sizes[4]), GELU()),
            nn.Sequential(nn.Linear(layer_sizes[4],
layer_sizes[5]), GELU()),
        ])
    def forward(self, x):
        for layer in self.layers:
            layer_output = layer(x)
            if self.use_shortcut and x.shape == layer_output.shape:
                x = x + layer_output
            else:
                x = layer_output
        return x
```

Transformer Block

> 在transfomer模块中连接注意力层和线性层。

- Transformer块示意图：包括层归一化，多头注意力，Dropout，前馈网络层（GELU 激活函数）和快捷链接



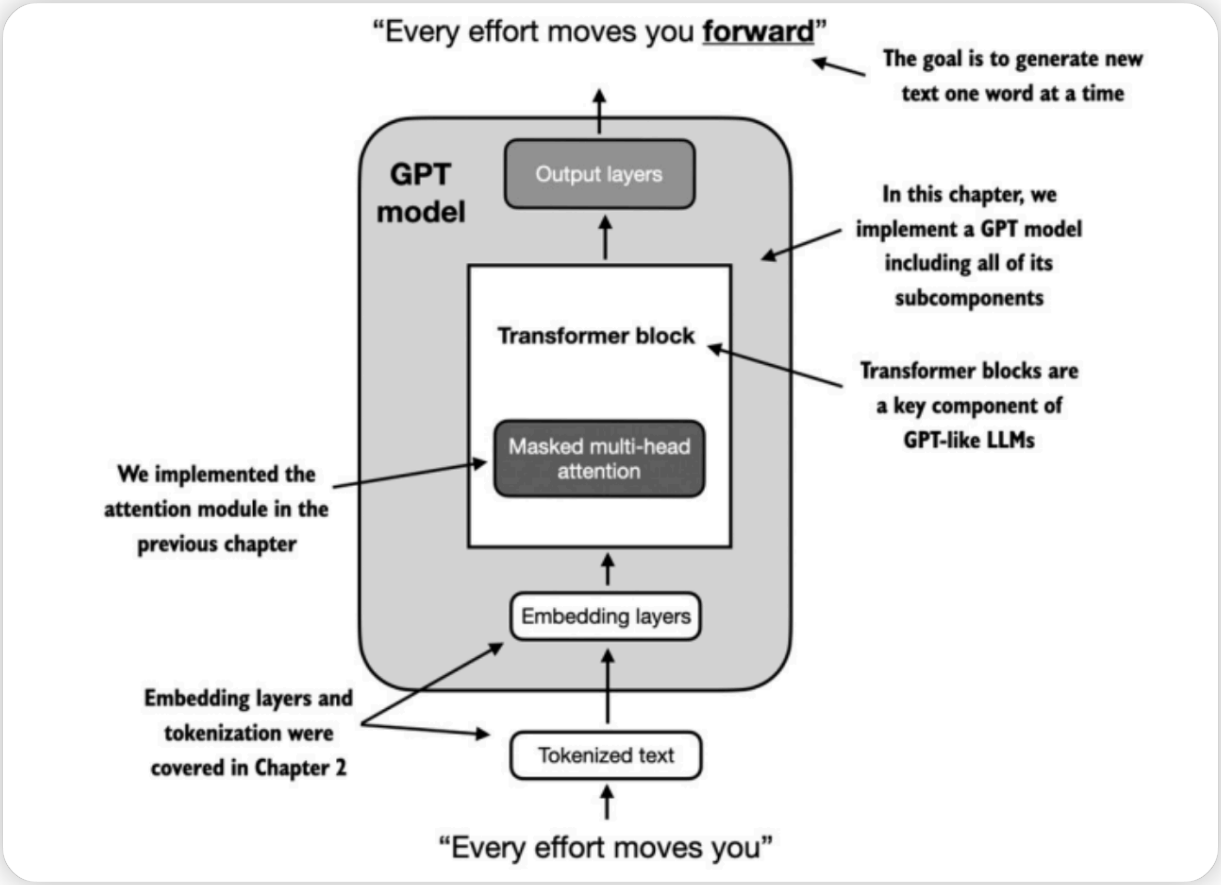
代码 - TransformerBlock

```
import torch
import torch.nn as nn
from torch.nn import LayerNorm

class TransformerBlock(nn.Module):
    def __init__(self, cfg):
        super().__init__()
        self.att = MultiHeadAttention(
```

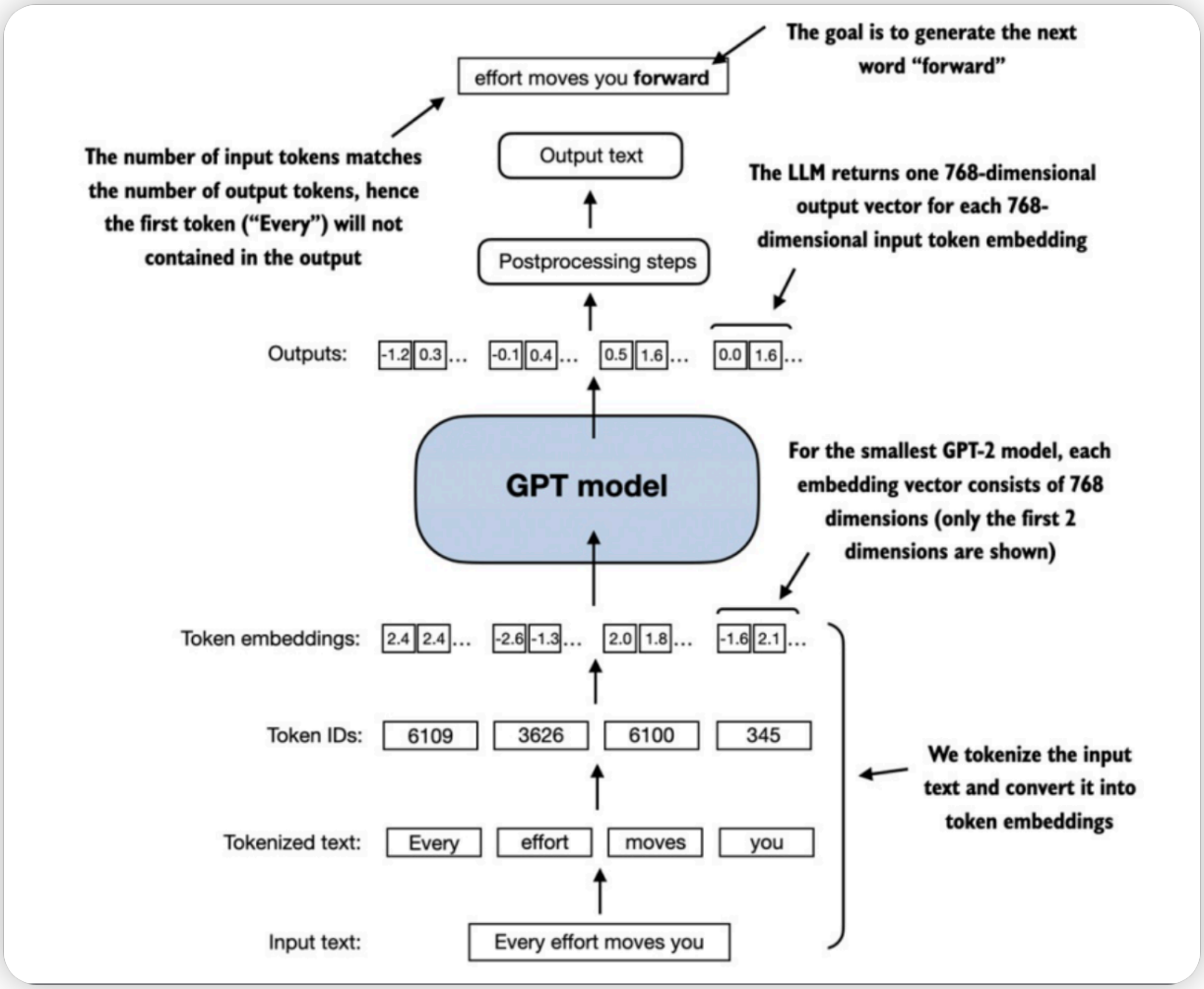
```
            d_in=cfg["emb_dim"],
            d_out=cfg["emb_dim"],
            context_length=cfg["context_length"],
            dropout=cfg["drop_rate"],
            num_heads=cfg["n_heads"],
            qkv_bias=cfg["qkv_bias"],
        )

        self.ff = FeedForward(cfg)
        self.norm1 = LayerNorm(cfg["emb_dim"])
        self.norm2 = LayerNorm(cfg["emb_dim"])
        self.drop_resid = nn.Dropout(cfg["drop_rate"])

    def forward(self, x):
        shortcut = x
        x = self.norm1(x)
        x = self.att(x)
        x = self.drop_resid(x)
        x = x + shortcut

        shortcut = x
        x = self.norm2(x)
        x = self.ff(x)
        x = self.drop_resid(x)
        x = x + shortcut
        return x
```

总结

- GPT模型：


"Every effort moves you forward"
The goal is to generate new text one word at a time
GPT model
Output layers
In this chapter, we implement a GPT model including all of its subcomponents
Transformer block
Transformer blocks are a key component of GPT-like LLMs
Masked multi-head attention
We implemented the attention module in the previous chapter
Embedding layers
Embedding layers and tokenization were covered in Chapter 2
Tokenized text
"Every effort moves you"

- GPT-2的数据流图：


The goal is to generate the next word "forward"
effort moves you forward
The number of input tokens matches the number of output tokens, hence the first token ("Every") will not contained in the output
Output text
The LLM returns one 768-dimensional output vector for each 768-dimensional input token embedding
Postprocessing steps
Outputs:
-1.2 0.3 ... -0.1 0.4 ... 0.5 1.6 ... 0.0 1.6 ...
GPT model
For the smallest GPT-2 model, each embedding vector consists of 768 dimensions (only the first 2 dimensions are shown)
Token embeddings:
2.4 2.4 ... -2.6 -1.3 ... 2.0 1.8 ... -1.6 2.1 ...
Token IDs:
6109
3626
6100
345
We tokenize the input text and convert it into token embeddings
Tokenized text:
Every
effort
moves
you
Input text:
Every effort moves you