The class also implements the forward pass, where each component is followed by a shortcut connection that adds the input of the block to its output. This critical feature helps gradients flow through the network during training and improves the learning of deep models (see section 4.4).

Using the `GPT_CONFIG_124M` dictionary we defined earlier, let's instantiate a transformer block and feed it some sample data:

```
torch.manual_seed(123)
x = torch.rand(2, 4, 768)                          # Creates sample input of shape [batch_size, num_tokens, emb_dim]
block = TransformerBlock(GPT_CONFIG_124M)
output = block(x)

print("Input shape:", x.shape)
print("Output shape:", output.shape)
```

The output is

```
Input shape: torch.Size([2, 4, 768])
Output shape: torch.Size([2, 4, 768])
```

As we can see, the transformer block maintains the input dimensions in its output, indicating that the transformer architecture processes sequences of data without altering their shape throughout the network.

The preservation of shape throughout the transformer block architecture is not incidental but a crucial aspect of its design. This design enables its effective application across a wide range of sequence-to-sequence tasks, where each output vector directly corresponds to an input vector, maintaining a one-to-one relationship. However, the output is a context vector that encapsulates information from the entire input sequence (see chapter 3). This means that while the physical dimensions of the sequence (length and feature size) remain unchanged as it passes through the transformer block, the content of each output vector is re-encoded to integrate contextual information from across the entire input sequence.

With the transformer block implemented, we now have all the building blocks needed to implement the GPT architecture. As illustrated in figure 4.14, the transformer block combines layer normalization, the feed forward network, GELU activations, and shortcut connections. As we will eventually see, this transformer block will make up the main component of the GPT architecture.

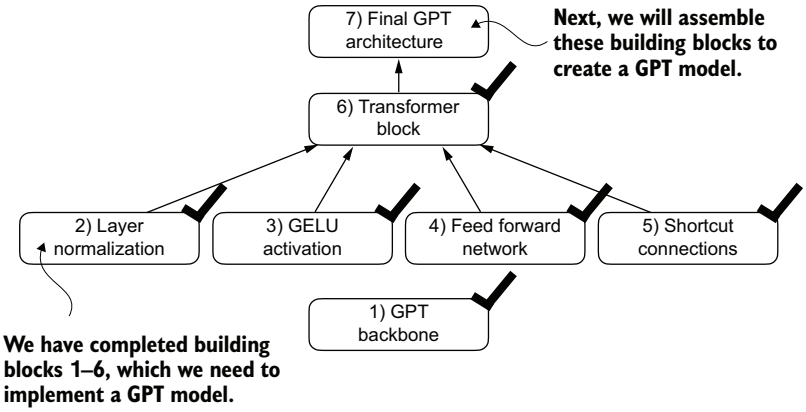

Figure 4.14 The building blocks necessary to build the GPT architecture. The black checks indicate the blocks we have completed.

4.6 Coding the GPT model

We started this chapter with a big-picture overview of a GPT architecture that we called `DummyGPTModel`. In this `DummyGPTModel` code implementation, we showed the input and outputs to the GPT model, but its building blocks remained a black box using a `DummyTransformerBlock` and `DummyLayerNorm` class as placeholders.

Let's now replace the `DummyTransformerBlock` and `DummyLayerNorm` placeholders with the real `TransformerBlock` and `LayerNorm` classes we coded previously to assemble a fully working version of the original 124-million-parameter version of GPT-2. In chapter 5, we will pretrain a GPT-2 model, and in chapter 6, we will load in the pretrained weights from OpenAI.

Before we assemble the GPT-2 model in code, let's look at its overall structure, as shown in figure 4.15, which includes all the concepts we have covered so far. As we can see, the transformer block is repeated many times throughout a GPT model architecture. In the case of the 124-million-parameter GPT-2 model, it's repeated 12 times, which we specify via the `n_layers` entry in the `GPT_CONFIG_124M` dictionary. This transform block is repeated 48 times in the largest GPT-2 model with 1,542 million parameters.

The output from the final transformer block then goes through a final layer normalization step before reaching the linear output layer. This layer maps the transformer's output to a high-dimensional space (in this case, 50,257 dimensions, corresponding to the model's vocabulary size) to predict the next token in the sequence.

Let's now code the architecture in figure 4.15.

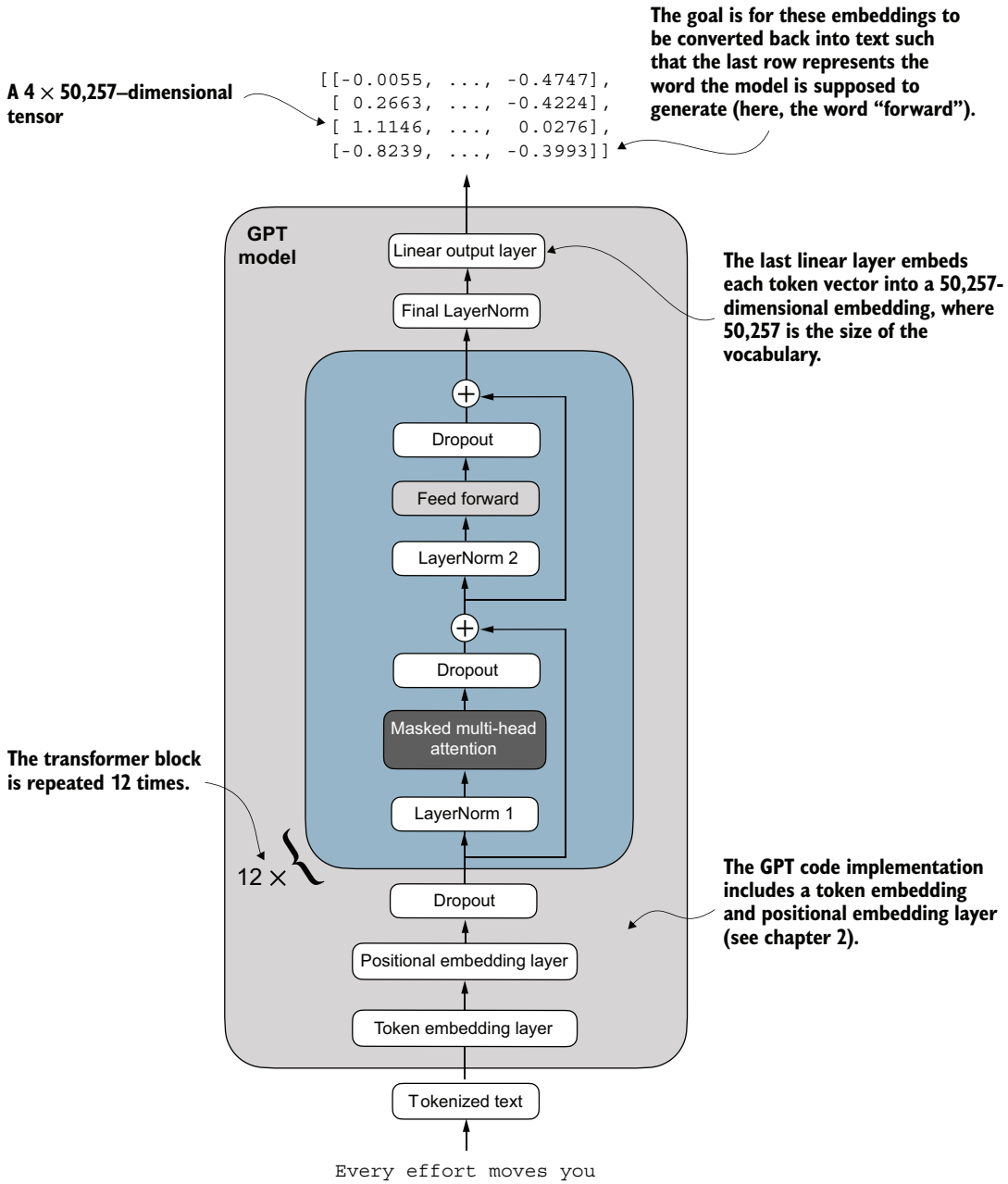

Figure 4.15 An overview of the GPT model architecture showing the flow of data through the GPT model. Starting from the bottom, tokenized text is first converted into token embeddings, which are then augmented with positional embeddings. This combined information forms a tensor that is passed through a series of transformer blocks shown in the center (each containing multi-head attention and feed forward neural network layers with dropout and layer normalization), which are stacked on top of each other and repeated 12 times.