

# *Implementing a GPT model from scratch to generate text*

## **This chapter covers**

- Coding a GPT-like large language model (LLM) that can be trained to generate human-like text
- Normalizing layer activations to stabilize neural network training
- Adding shortcut connections in deep neural networks
- Implementing transformer blocks to create GPT models of various sizes
- Computing the number of parameters and storage requirements of GPT models

You've already learned and coded the *multi-head attention* mechanism, one of the core components of LLMs. Now, we will code the other building blocks of an LLM and assemble them into a GPT-like model that we will train in the next chapter to generate human-like text.

The LLM architecture referenced in figure 4.1, consists of several building blocks. We will begin with a top-down view of the model architecture before covering the individual components in more detail.

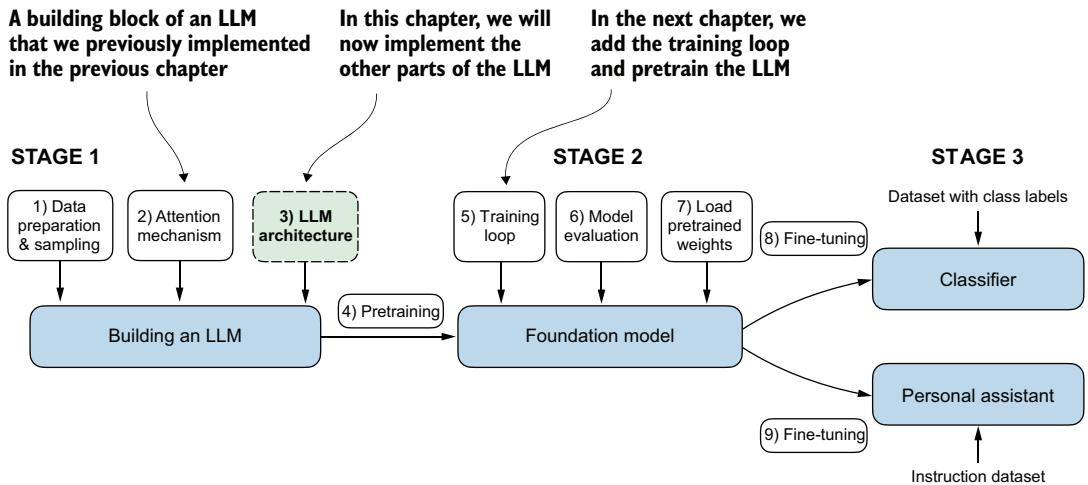


Figure 4.1 The three main stages of coding an LLM. This chapter focuses on step 3 of stage 1: implementing the LLM architecture.

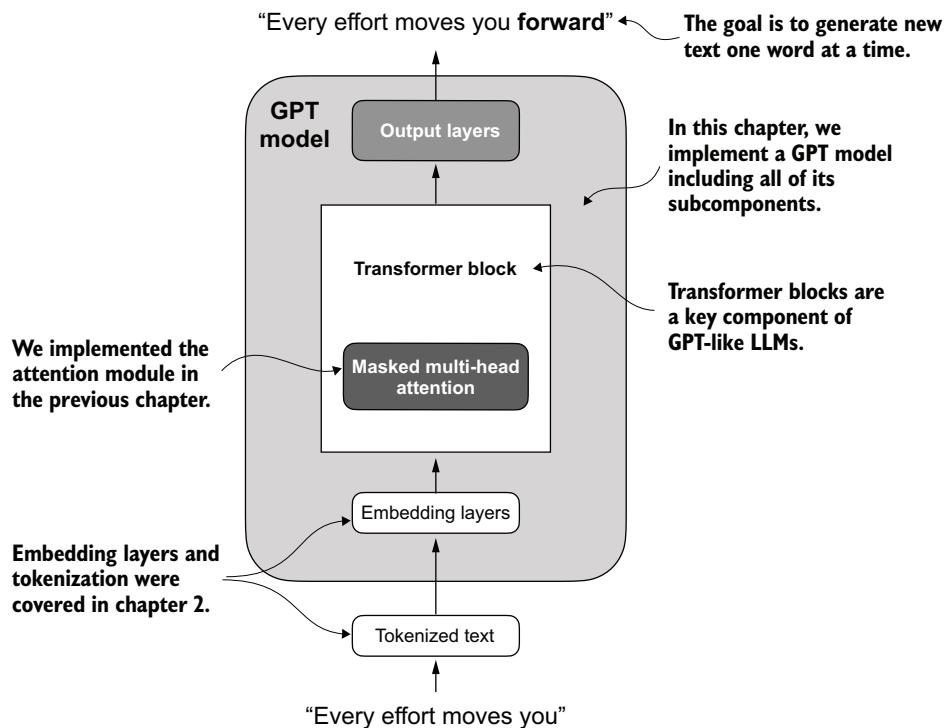
## 4.1 Coding an LLM architecture

LLMs, such as GPT (which stands for *generative pretrained transformer*), are large deep neural network architectures designed to generate new text one word (or token) at a time. However, despite their size, the model architecture is less complicated than you might think, since many of its components are repeated, as we will see later. Figure 4.2 provides a top-down view of a GPT-like LLM, with its main components highlighted.

We have already covered several aspects of the LLM architecture, such as input tokenization and embedding and the masked multi-head attention module. Now, we will implement the core structure of the GPT model, including its *transformer blocks*, which we will later train to generate human-like text.

Previously, we used smaller embedding dimensions for simplicity, ensuring that the concepts and examples could comfortably fit on a single page. Now, we are scaling up to the size of a small GPT-2 model, specifically the smallest version with 124 million parameters, as described in “Language Models Are Unsupervised Multitask Learners,” by Radford et al. (<https://mng.bz/yoBq>). Note that while the original report mentions 117 million parameters, this was later corrected. In chapter 6, we will focus on loading pretrained weights into our implementation and adapting it for larger GPT-2 models with 345, 762, and 1,542 million parameters.

In the context of deep learning and LLMs like GPT, the term “parameters” refers to the trainable weights of the model. These weights are essentially the internal variables of the model that are adjusted and optimized during the training process to minimize a specific loss function. This optimization allows the model to learn from the training data.



**Figure 4.2** A GPT model. In addition to the embedding layers, it consists of one or more transformer blocks containing the masked multi-head attention module we previously implemented.

For example, in a neural network layer that is represented by a  $2,048 \times 2,048$ -dimensional matrix (or tensor) of weights, each element of this matrix is a parameter. Since there are 2,048 rows and 2,048 columns, the total number of parameters in this layer is 2,048 multiplied by 2,048, which equals 4,194,304 parameters.

### GPT-2 vs. GPT-3

Note that we are focusing on GPT-2 because OpenAI has made the weights of the pretrained model publicly available, which we will load into our implementation in chapter 6. GPT-3 is fundamentally the same in terms of model architecture, except that it is scaled up from 1.5 billion parameters in GPT-2 to 175 billion parameters in GPT-3, and it is trained on more data. As of this writing, the weights for GPT-3 are not publicly available. GPT-2 is also a better choice for learning how to implement LLMs, as it can be run on a single laptop computer, whereas GPT-3 requires a GPU cluster for training and inference. According to Lambda Labs (<https://lambdalabs.com/>), it would take 355 years to train GPT-3 on a single V100 datacenter GPU and 665 years on a consumer RTX 8000 GPU.