Let's try this `classify_review` function on an example text:

```
text_1 = (
    "You are a winner you have been specially"
    " selected to receive $1000 cash or a $2000 award."
)

print(classify_review(
    text_1, model, tokenizer, device, max_length=train_dataset.max_length
))
```

The resulting model correctly predicts `"spam"`. Let's try another example:

```
text_2 = (
    "Hey, just wanted to check if we're still on"
    " for dinner tonight? Let me know!"
)

print(classify_review(
    text_2, model, tokenizer, device, max_length=train_dataset.max_length
))
```

The model again makes a correct prediction and returns a "not spam" label.

Finally, let's save the model in case we want to reuse the model later without having to train it again. We can use the `torch.save` method:

```
torch.save(model.state_dict(), "review_classifier.pth")
```

Once saved, the model can be loaded:

```
model_state_dict = torch.load("review_classifier.pth, map_location=device")
model.load_state_dict(model_state_dict)
```

Summary

- There are different strategies for fine-tuning LLMs, including classification fine-tuning and instruction fine-tuning.
- Classification fine-tuning involves replacing the output layer of an LLM via a small classification layer.
- In the case of classifying text messages as "spam" or "not spam," the new classification layer consists of only two output nodes. Previously, we used the number of output nodes equal to the number of unique tokens in the vocabulary (i.e., 50,256).
- Instead of predicting the next token in the text as in pretraining, classification fine-tuning trains the model to output a correct class label—for example, "spam" or "not spam."
- The model input for fine-tuning is text converted into token IDs, similar to pretraining.

- Before fine-tuning an LLM, we load the pretrained model as a base model.
- Evaluating a classification model involves calculating the classification accuracy (the fraction or percentage of correct predictions).
- Fine-tuning a classification model uses the same cross entropy loss function as when pretraining the LLM.

7 Fine-tuning to follow instructions

This chapter covers

- The instruction fine-tuning process of LLMs
- Preparing a dataset for supervised instruction fine-tuning
- Organizing instruction data in training batches
- Loading a pretrained LLM and fine-tuning it to follow human instructions
- Extracting LLM-generated instruction responses for evaluation
- Evaluating an instruction-fine-tuned LLM

Previously, we implemented the LLM architecture, carried out pretraining, and imported pretrained weights from external sources into our model. Then, we focused on fine-tuning our LLM for a specific classification task: distinguishing between spam and non-spam text messages. Now we'll implement the process for fine-tuning an LLM to follow human instructions, as illustrated in figure 7.1. Instruction fine-tuning is one of the main techniques behind developing LLMs for chatbot applications, personal assistants, and other conversational tasks.