

**Figure 7.10** The input and target token alignment used in the instruction fine-tuning process of an LLM. For each input sequence, the corresponding target sequence is created by shifting the token IDs one position to the right, omitting the first token of the input, and appending an end-of-text token.

The following updated collate function generates the target token IDs from the input token IDs:

```
def custom_collate_draft_2(
    batch,
    pad_token_id=50256,
    device="cpu"
):
    batch_max_length = max(len(item)+1 for item in batch)
    inputs_lst, targets_lst = [], []

    for item in batch:
        new_item = item.copy()
        new_item += [pad_token_id]
```

```

padded = (
    new_item + [pad_token_id] *
    (batch_max_length - len(new_item))
)
inputs = torch.tensor(padded[:-1])           ← Truncates the last token for inputs
targets = torch.tensor(padded[1:])           ← Shifts +1 to the right for targets
inputs_lst.append(inputs)
targets_lst.append(targets)

inputs_tensor = torch.stack(inputs_lst).to(device)
targets_tensor = torch.stack(targets_lst).to(device)
return inputs_tensor, targets_tensor

inputs, targets = custom_collate_draft_2(batch)
print(inputs)
print(targets)

```

Applied to the example `batch` consisting of three input lists we defined earlier, the new `custom_collate_draft_2` function now returns the input and the target batch:

```

tensor([[ 0,      1,      2,      3,      4],   ← The first tensor represents inputs.
        [ 5,      6, 50256, 50256, 50256],
        [ 7,      8,      9, 50256, 50256]])
tensor([[ 1,      2,      3,      4, 50256],   ← The second tensor represents the targets.
        [ 6, 50256, 50256, 50256, 50256],
        [ 8,      9, 50256, 50256, 50256]])

```

In the next step, we assign a `-100` placeholder value to all padding tokens, as highlighted in figure 7.11. This special value allows us to exclude these padding tokens from contributing to the training loss calculation, ensuring that only meaningful data influences model learning. We will discuss this process in more detail after we implement this modification. (When fine-tuning for classification, we did not have to worry about this since we only trained the model based on the last output token.)

However, note that we retain one end-of-text token, ID `50256`, in the target list, as depicted in figure 7.12. Retaining it allows the LLM to learn when to generate an end-of-text token in response to instructions, which we use as an indicator that the generated response is complete.

In the following listing, we modify our custom collate function to replace tokens with ID `50256` with `-100` in the target lists. Additionally, we introduce an `allowed_max_length` parameter to optionally limit the length of the samples. This adjustment will be useful if you plan to work with your own datasets that exceed the 1,024-token context size supported by the GPT-2 model.

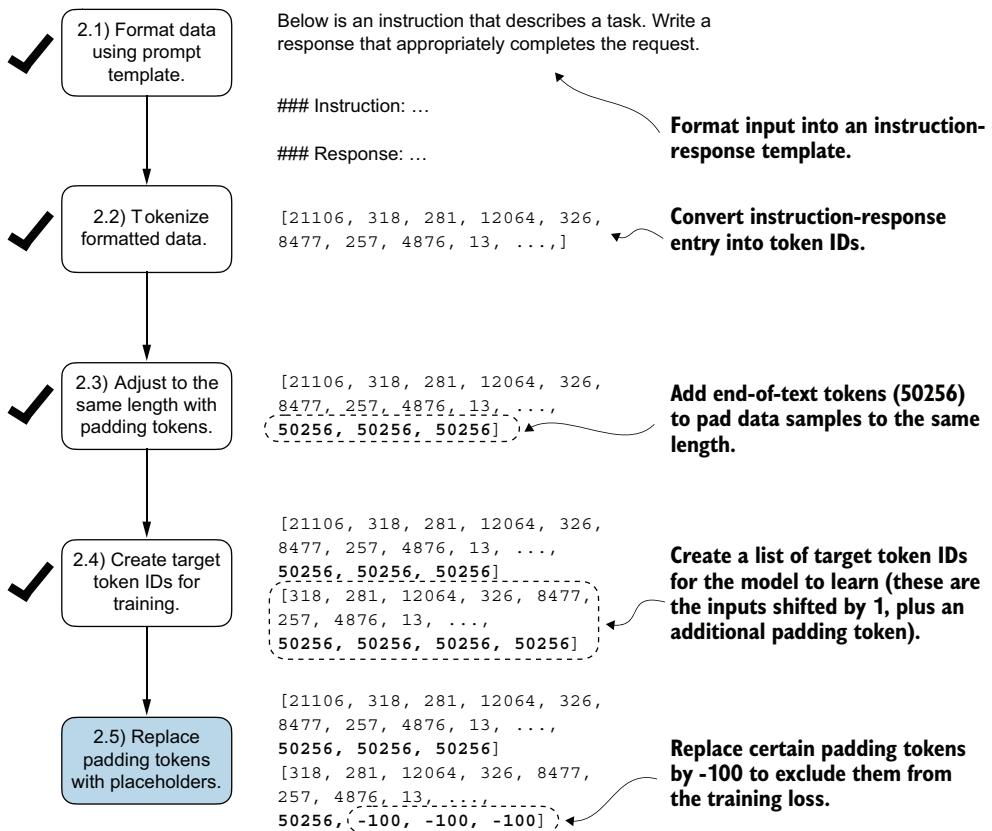


Figure 7.11 The five substeps involved in implementing the batching process. After creating the target sequence by shifting token IDs one position to the right and appending an end-of-text token, in step 2.5, we replace the end-of-text padding tokens with a placeholder value (-100).

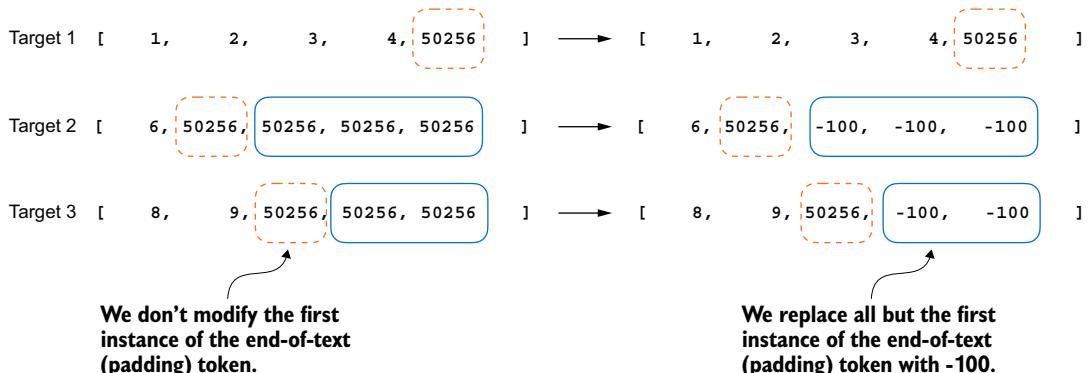


Figure 7.12 Step 2.4 in the token replacement process in the target batch for the training data preparation. We replace all but the first instance of the end-of-text token, which we use as padding, with the placeholder value -100, while keeping the initial end-of-text token in each target sequence.