First, we apply this approach to the first three examples from the test set that we previously examined:

```
for entry in test_data[:3]:
    prompt = (
        f"Given the input `{format_input(entry)}` "
        f"and correct output `{entry['output']}`, "
        f"score the model response `{entry['model_response']}`"
        f" on a scale from 0 to 100, where 100 is the best score. "
    )
    print("\nDataset response:")
    print(">>", entry['output'])
    print("\nModel response:")
    print(">>", entry["model_response"])
    print("\nScore:")
    print(">>", query_model(prompt))
    print("\n-------------------------")
```

This code prints outputs similar to the following (as of this writing, Ollama is not fully deterministic, so the generated texts may vary):

Dataset response:

>> The car is as fast as lightning.

Model response:

>> The car is as fast as a bullet.

Score:

>> I'd rate the model response "The car is as fast as a bullet." an 85 out of 100.

Here's why:

The response uses a simile correctly, comparing the speed of the car to something else (in this case, a bullet).

The comparison is relevant and makes sense, as bullets are known for their high velocity.

The phrase "as fast as" is used correctly to introduce the simile.

The only reason I wouldn't give it a perfect score is that some people might find the comparison slightly less vivid or evocative than others. For example, comparing something to lightning (as in the original response) can be more dramatic and attention grabbing. However, "as fast as a bullet" is still a strong and effective simile that effectively conveys the idea of the car's speed.

Overall, I think the model did a great job!

Dataset response:

>> The type of cloud typically associated with thunderstorms is cumulonimbus.

Model response:

>> The type of cloud associated with thunderstorms is a cumulus cloud.

Score:

>> I'd score this model response as 40 out of 100.

Here's why:

The model correctly identifies that thunderstorms are related to clouds (correctly identifying the type of phenomenon).

However, it incorrectly specifies the type of cloud associated with thunderstorms. Cumulus clouds are not typically associated with thunderstorms; cumulonimbus clouds are.

The response lacks precision and accuracy in its description.

Overall, while the model attempts to address the instruction, it provides an incorrect answer, which is a significant error.

Dataset response:

>> Jane Austen.

Model response:

>> The author of 'Pride and Prejudice' is Jane Austen.

Score:

>> I'd rate my own response as 95 out of 100. Here's why:

The response accurately answers the question by naming the author of *Pride and Prejudice* as Jane Austen.

The response is concise and clear, making it easy to understand.

There are no grammatical errors or ambiguities that could lead to confusion.

The only reason I wouldn't give myself a perfect score is that the response is slightly redundant—it's not necessary to rephrase the question in the answer. A more concise response would be simply "Jane Austen."

The generated responses show that the Llama 3 model provides reasonable evaluations and is capable of assigning partial points when a model's answer is not entirely correct. For instance, if we consider the evaluation of the "cumulus cloud" answer, the model acknowledges the partial correctness of the response.

The previous prompt returns highly detailed evaluations in addition to the score. We can modify the prompt to just generate integer scores ranging from 0 to 100, where 100 represents the best possible score. This modification allows us to calculate an average score for our model, which serves as a more concise and quantitative assessment of its performance. The `generate_model_scores` function shown in the following listing uses a modified prompt telling the model to `"Respond with the integer number only."`

Listing 7.11 Evaluating the instruction fine-tuning LLM

```
def generate_model_scores(json_data, json_key, model="llama3"):
    scores = []
    for entry in tqdm(json_data, desc="Scoring entries"):
        prompt = (
            f"Given the input `{format_input(entry)}` "
            f"and correct output `{entry['output']}`, "
            f"score the model response `{entry[json_key]}`"
            f" on a scale from 0 to 100, where 100 is the best score. "
            f"Respond with the integer number only."
        )
        score = query_model(prompt, model)
        try:
            scores.append(int(score))
        except ValueError:
            print(f"Could not convert score: {score}")
            continue

    return scores
```

Modified instruction line to only return the score

Let's now apply the `generate_model_scores` function to the entire `test_data` set, which takes about 1 minute on a M3 Macbook Air:

```
scores = generate_model_scores(test_data, "model_response")
print(f"Number of scores: {len(scores)} of {len(test_data)}")
print(f"Average score: {sum(scores)/len(scores):.2f}\n")
```

The results are as follows:

```
Scoring entries: 100%|████████████████████████| 110/110
[01:10<00:00,  1.56it/s]
Number of scores: 110 of 110
Average score: 50.32
```

The evaluation output shows that our fine-tuned model achieves an average score above 50, which provides a useful benchmark for comparison against other models