Giovanni Boscan 11/29/2023

Lab 4 Report

Prompts utilized for Mistra Instruct

To create this prompt, I decided to use the single-shot training approach. I first developed a paragraph to tell the LLM the detailed description of the problem, along with some constraints like the JSON formatting and some hints like using the "note" label. Then, I added an example input and a corresponding valid output in order to guide the LLM on how the result should be formatted. Lastly, I added the actual input to test the LLM:

```
The task is to fix the following code that is formatted in JSON with the provided
error message and provide a description of the resolution of the error.
The note message will help a lot in order to figure out the error.
The line and column numbers are also provided in the JSON object.
The source code is provided and it is contained in the "code" label of the JSON object
provided.
The error message is in the "error" label of the JSON object provided.
The note message is in the "note" label of the JSON object provided, if any.

Example:
{"./Project1/Dog.cpp": [{"code": "    ptr2 = &val;\n\n    std::cout << \"Hello
World!!!\" << std::endl\n    return 0;\n}\n","column": 47,"error": "expected ';'
before 'return'","file": "./compilecode/Project1/hello_world.cpp","line": 11}]}

Response:
{"code": "    ptr2 = &val;\n\n    std::cout << \"Hello World!!!\" << std::endl;\n
return 0;\n}\n"}

The error was caused by a missing semicolon in line 11, column 47. As a result, a
semicolon was added after the std::endl statement in order to solve the compiler
error.

End of Example

Here is the task:
```

```
{"./Project1/Dog.cpp": [{"code": "{\n    int woof = 1\n}\n","column": 1,"error":
"expected ',' or ';' before '}' token","file":
"./compilecode/Project1/Dog.cpp","line": 8}]}
```

Mistra Instruct Output:

```
The error message is indicating that there should be a comma or semicolon before the
closing curly brace `}` on line 8 of file `./compilecode/Project1/Dog.cpp`.
To fix this error, you can add either a comma or a semicolon after the integer value
`woof = 1` on line 7. Here's an example with both options:
json
{"./Project1/Dog.cpp": [{"code": "{\n    int woof = 1,\n}\n","column": 1,"error":
"expected ',' or ';' before '}' token","file":
"./compilecode/Project1/Dog.cpp","line": 8}]}
Or
json
{"./Project1/Dog.cpp": [{"code": "{\n    int woof = 1;\n}\n","column": 1,"error":
"expected ',' or ';' before '}' token","file":
"./compilecode/Project1/Dog.cpp","line": 8}]}
Both of these options should resolve the error and allow the code to compile
successfully.
```

This prompt provided the correct output but added a second solution that contained a syntax error, I think it got confused with JSON and C++ since I wrote JSON before C++, which is not the intended result. To fix this, I reworded the first sentence in order to clarify the task and specify the target programming language (C++):

```
The task is to fix the following C++ code with the given error message and provide a
description of the resolution of the error.
The input will be formatted in JSON.
The note message will help a lot in order to figure out the error.
The line and column numbers are also provided in the JSON object.
The source C++ code is provided and it is contained in the "code" label of the JSON
object provided.
The error message is in the "error" label of the JSON object provided.
The note message is in the "note" label of the JSON object provided, if any.

Example:
```

```
{"./Project1/Dog.cpp": [{"code": "    ptr2 = &val;\n\n    std::cout << \"Hello World!!!\" << std::endl\n    return 0;\n}\n","column": 47,"error": "expected ';' before 'return'","file": "./compilecode/Project1/hello_world.cpp","line": 11}]}
```

Response:

```
{"code": "    ptr2 = &val;\n\n    std::cout << \"Hello World!!!\" << std::endl;\n return 0;\n}\n"}
```

The error was caused by a missing semicolon in line 11, column 47. As a result, a semicolon was added after the std::endl statement in order to solve the compiler error.

End of Example

Here is the task:

```
{"./Project1/Dog.cpp": [{"code": "{\n    int woof = 1\n}\n","column": 1,"error": "expected ',' or ';' before '}' token","file": "./compilecode/Project1/Dog.cpp","line": 8}]}
```

Mistra Instruct Output:

The error message indicates that there is an issue with the syntax of the code at line 8, column 1 in the file "./compilecode/Project1/Dog.cpp". Specifically, it seems that there is a problem with the curly braces ({}).

To fix this error, we need to add a semicolon after the int statement on line 7, before the opening brace of the block. This will ensure that the syntax is correct and the compiler can compile the code without any issues. Here's what the corrected code should look like:

```
c++
{
int woof = 1;
}
```

By adding this semicolon, we are telling the compiler to expect a block of statements after the int statement, but not before it. This will ensure that the curly braces on lines 8 and 9 are properly matched and the code compiles without any errors.

It interpreted the code right and produced the corrected output, however, it didn't provide a JSON object as output, just a code snippet. To solve this, I tried to add a sentence to state that the response has to be in JSON format:

```
The task is to fix the following C++ code with the given error message and provide a
description of the resolution of the error in JSON format.
The input will be formatted in JSON.
The note message will help a lot in order to figure out the error.
The line and column numbers are also provided in the JSON object.
The source C++ code is provided and it is contained in the "code" label of the JSON
object provided.
The error message is in the "error" label of the JSON object provided.
The note message is in the "note" label of the JSON object provided, if any.
Your output response has to be in JSON format.

Example:
{"./Project1/Dog.cpp": [{"code": "    ptr2 = &val;\n\n    std::cout << \"Hello
World!!!\" << std::endl\n    return 0;\n}\n","column": 47,"error": "expected ';'
before 'return'","file": "./compilecode/Project1/hello_world.cpp","line": 11}]}

Response:
{"code": "    ptr2 = &val;\n\n    std::cout << \"Hello World!!!\" << std::endl;\n
return 0;\n}\n"}

The error was caused by a missing semicolon in line 11, column 47. As a result, a
semicolon was added after the std::endl statement in order to solve the compiler
error.

End of Example

Here is the task:
{"./Project1/Dog.cpp": [{"code": "{\n    int woof = 1\n}\n","column": 1,"error":
"expected ',' or ';' before '}' token","file":
"./compilecode/Project1/Dog.cpp","line": 8}]}
```

Mistra Instruct Ouput:

```
{"./Project1/Dog.cpp": [{"code": "{\n    int woof = 1\n}\n","column": 2,"error":
"expected ',' or ';' before '}' token","file":
"./compilecode/Project1/Dog.cpp","line": 8}]}
```

This produced the right result but it didn't provide any feedback on the error fix. Therefore, I removed the last prompt sentence to not overwhelm the LLM with the JSON output:

```
The task is to fix the following C++ code with the given error message and provide a
description of the resolution of the error in JSON format.
The input will be formatted in JSON.
The note message will help a lot in order to figure out the error.
The line and column numbers are also provided in the JSON object.
The source C++ code is provided and it is contained in the "code" label of the JSON
object provided.
The error message is in the "error" label of the JSON object provided.
The note message is in the "note" label of the JSON object provided, if any.

Example:
{"./Project1/Dog.cpp": [{"code": "    ptr2 = &val;\n\n    std::cout << \"Hello
World!!!\" << std::endl\n    return 0;\n}\n","column": 47,"error": "expected ';'
before 'return'","file": "./compilecode/Project1/hello_world.cpp","line": 11}]}

Response:
{"code": "    ptr2 = &val;\n\n    std::cout << \"Hello World!!!\" << std::endl;\n
return 0;\n}\n"}

The error was caused by a missing semicolon in line 11, column 47. As a result, a
semicolon was added after the std::endl statement in order to solve the compiler
error.

End of Example

Here is the task:
{"./Project1/Dog.cpp": [{"code": "{\n    int woof = 1\n}\n","column": 1,"error":
"expected ',' or ';' before '}' token","file":
"./compilecode/Project1/Dog.cpp","line": 8}]}
```

Mistral Instruct Output:

```
{"code": "{\n    int woof = 1;\n}", "column": 1, "resolution": "The error was caused
by a missing semicolon after the int variable declaration in line 7. As a result, a
semicolon was added after the variable declaration to solve the compiler error."}
```

This produced a really convincing and helpful output, considering the variations the model can have on the output and also considering the complexity of the problem. However, on some occasions, it was not providing the fixed source code in the output JSON, so I added a statement to make sure the LLM included it in the response. Also, I added the response inside the JSON object so the LLM adds it inside the response JSON object as well:

```
The task is to fix the following C++ code with the given error message and provide a
description of the resolution of the error in JSON format.
The fixed code should be contained in the JSON output.
The input will be formatted in JSON.
The note message will help a lot in order to figure out the error.
The line and column numbers are also provided in the JSON object.
The source C++ code is provided and it is contained in the "code" label of the JSON
object provided.
The error message is in the "error" label of the JSON object provided.
The note message is in the "note" label of the JSON object provided, if any.

Example:
{"./Project1/Dog.cpp": [{"code": "    ptr2 = &val;\n\n    std::cout << \"Hello
World!!!\" << std::endl\n    return 0;\n}\n","column": 47,"error": "expected ';'
before 'return'","file": "./compilecode/Project1/hello_world.cpp","line": 11}]}

Response:
{"code": "    ptr2 = &val;\n\n    std::cout << \"Hello World!!!\" << std::endl;\n
return 0;\n}\n", "resolution": "The error was caused by a missing semicolon in line
11, column 47. As a result, a semicolon was added after the std::endl statement in
order to solve the compiler error."}

End of Example

Here is the task:
{"./Project1/Dog.cpp": [{"code": "{\n    int woof = 1\n}\n","column": 1,"error":
"expected ',' or ';' before '}' token","file":
"./compilecode/Project1/Dog.cpp","line": 8}]}
```

Mistral Instruct Final Output:

```
Here is the fixed code in JSON format:\n{\n    \"./Project1/Dog.cpp\": {\n
\"code\": \"{ \\n\\tint woof = 1; \\n}\",\n        \"resolution\": \"The error was
caused by a missing semicolon after the line containing only an integer value (woof).
As a result, the necessary semicolons were added before and after the line in order to
solve the compiler error.\"\n    }\n}
```

This last prompt was the one that produced the most consistent and correct outputs of all those I tried for Mistra Instruct, therefore, I stuck with it and added it to my code in order to generate the fixes for the other example codes I had.

Prompts utilized for GTP4All Falcon

For the GTP4All Falcon LLM prompt, I started with the same prompt I used for Mistral Instruct in order to test if the same prompt would produce the same desired output:

```
The task is to fix the following C++ code with the given error message and provide a
description of the resolution of the error in JSON format.
The input will be formatted in JSON.
The note message will help a lot in order to figure out the error.
The line and column numbers are also provided in the JSON object.
The source C++ code is provided and it is contained in the "code" label of the JSON
object provided.
The error message is in the "error" label of the JSON object provided.
The note message is in the "note" label of the JSON object provided, if any.

Example:
{"./Project1/Dog.cpp": [{"code": "    ptr2 = &val;\n\n    std::cout << \"Hello
World!!!\" << std::endl\n    return 0;\n}\n","column": 47,"error": "expected ';'
before 'return'","file": "./compilecode/Project1/hello_world.cpp","line": 11}]}

Response:
{"code": "    ptr2 = &val;\n\n    std::cout << \"Hello World!!!\" << std::endl;\n
return 0;\n}\n"}

The error was caused by a missing semicolon in line 11, column 47. As a result, a
semicolon was added after the std::endl statement in order to solve the compiler
error.

End of Example

Here is the task:
{"./Project1/Dog.cpp": [{"code": "{\n    int woof = 1\n}\n","column": 1,"error":
"expected ',' or ';' before '}' token","file":
"./compilecode/Project1/Dog.cpp","line": 8}]}
```

GPT4All Falcon Output:

```
{"code": "{\n int woof = 1\n}\n"}
The 'expected ',' or ';' before '}' token error was caused by a missing semicolon in
the line 8 of the file "./compilecode/Project1/Dog.cpp".
```

This prompt generated a correct description of the error, however, it didn't fix the code error, so I changed a bit of the description in order to make it work. Therefore, I reworded some of the instructions in order to make them very clear. Also, I removed some of the ambiguous sentences lying around the example and added some redundancy since I noted that this model is really strict in following instructions:

```
The task is to fix the following C++ source code with the given error message and
provide a description of the resolution of the error in JSON format for the given
task.
The response should contain the fixed source code for the given task and solve the
error.
The input will be formatted in JSON.
The note message will help a lot in order to figure out the error.
The line and column numbers are also provided in the JSON object.
The source C++ code is provided and it is contained in the "code" label of the JSON
object provided.
The error message is in the "error" label of the JSON object provided.
The note message is in the "note" label of the JSON object provided, if any.

Example input:
{"./Project1/Dog.cpp": [{"code": "    ptr2 = &val;\n\n    std::cout << \"Hello
World!!!\" << std::endl\n    return 0;\n}\n","column": 47,"error": "expected ';'
before 'return'","file": "./compilecode/Project1/hello_world.cpp","line": 11}]}

Example response:
{"code": "    ptr2 = &val;\n\n    std::cout << \"Hello World!!!\" << std::endl;\n
return 0;\n}\n", "resolution": "The error was caused by a missing semicolon in line
11, column 47. As a result, a semicolon was added after the std::endl statement in
order to solve the compiler error."}

Given task:
{"./Project1/Dog.cpp": [{"code": "{\n    int woof = 1\n}\n","column": 1,"error":
"expected ',' or ';' before '}' token","file":
"./compilecode/Project1/Dog.cpp","line": 8}]}
```

GPT4All Falcon Output:

```
json
{
    "code": "    ptr2 = &val;\n\n    std::cout << \"Hello World!!!\" << std::endl;\n
return 0;\n",
    "resolution": "The  error was caused by a missing semicolon in line 11, column 47.
As a result, a semicolon was added after the std::endl statement in order to solve the
compiler error."
}
```

This prompt resolved the example input instead of the actual given task. Therefore, I tried switching strategies with my single-shot training and tested out zero shot since it was trying to solve the example given instead of the actual task, so I removed the examples and reworded the prompt a bit to be more specific:

```
The task is to fix the following C++ source code by modifying the original code and
using the given error message to provide a description of the resolution of the error
in JSON format for the given task.
The JSON output should only contain the modified code that solves the error along with
a short and intuitive description of how the error was solved.
The response should contain the fixed source code for the given task and solve the
error.
The input will be formatted in JSON.
The note message will help a lot in order to figure out the error.
The line and column numbers are also provided in the JSON object.
The source C++ code is provided and it is contained in the "code" label of the JSON
object provided.
The error message is in the "error" label of the JSON object provided.
The note message is in the "note" label of the JSON object provided, if any.

Solve the error for the following given task:
{"./Project1/Dog.cpp": [{"code": "{\n    int woof = 1\n}\n","column": 1,"error":
"expected ',' or ';' before '}' token","file":
"./compilecode/Project1/Dog.cpp","line": 8}]}
```

GPT4All Falcon Output:

```
json
  {
  "code": "{\n    int woof = 1\n}\n",
  "column": 1,
  "error": "expected ','  or ';'  before '}' token",
  "file": "./compilecode/Project1/Dog.cpp",
  "line": 8
}
### 1.cpp:8: error: expected ',' or ';' before '}' token
json
{
  "code": "{\n    int woof = 1\n}\n",
  "column": 1,
  "error": "expected ','  or ';'  before '}' token",
  "file": "./compilecode/Project1/Dog.cpp",
  "line": 8
}
```

This time, the LLM was trying to solve the error, however, it didn't manage correctly to solve the issue for the given source code. Lastly, I tried to remove a lot of redundant information that might make the task more ambiguous for the LLM like specifying the error and note labels:

```
The task is to fix the following C++ source code by modifying the original code and
using the given error message to provide a description of the resolution of the error
in JSON format for the given task.
The JSON output should only contain the modified code that solves the error along with
a short and intuitive description of how the error was solved.
The fixed code should be contained in the JSON output.
The response should contain the fixed source code for the given task and solve the
error.

Solve the error for the following given task:
{"./Project1/Dog.cpp": [{"code": "{\n    int woof = 1\n}\n","column": 1,"error":
"expected ',' or ';' before '}' token","file":
"./compilecode/Project1/Dog.cpp","line": 8}]}
```

GPT4All Falcon Final Output:

```
json
  {"./Project1/Dog.cpp": [{"code": "{\n    int woof = 1\n}\n","column": 1,"error":
"expected ','  or ';'  before '}'  token","file":
"./compilecode/Project1/Dog.cpp","line": 8}]}
```

Despite my efforts to try to make the LLM correct a simple missing semicolon in the source code, I couldn't manage to generate a convincing enough and consistent output for this LLM. I think this might be because of the way the model was trained. I believe that the way the model was trained was meant to strictly follow patterns and instructions, which I could note in the responses, however, this capability will not be really useful for the task I want to accomplish.

Details of the Custom Job

The custom Job I developed called “callLLM” takes a string input in JSON format and returns a string in the same format, as any regular job designed for the Job System. The input consists of an IP address which makes the Job capable of calling any LLM that has a REST API, a prompt, the LLM model name to use, and an optional API key if needed to connect to the API. The Job first parses the JSON input provided (containing the IP, prompt, model name, and the optional API key) then creates an instance of a command prompt and uses a pipe in order to return the output (either the LLM response or any error occurred) of a script I made in Python. This script was made to simplify the call to the LLM thanks to the Open AI API that is available in this programming language. It simply parses the arguments I provided from the command prompt call made from the Job in C++, then uses them to create an Open AI client object, call it with the prompt and model name, and catch any errors during the call. Lastly, it prints the output/error to the console which gets returned to the C++ callLLM job as described above.

The callLLM job gets executed in parallel thanks to the capabilities previously designed for the Job System, therefore, you can create multiple LLM calls at the same time and save a lot of time. Also, since I am using the Job System library, I create this custom Job by just providing the function pointer and a name, which gets stored in the Job System and referenced later in order to execute the Job.

In order to provide the prompts and errors for each LLM, I created a prompt file for each LLM I tested (Mistra Instruct and GPT4All Falcon) that already provided the single and zero-shot training, and some error file examples in JSON format that I generated in Lab 1. So, to read those files and not break the command prompt call or JSON objects, I created a custom function to read the file and escape any backslashes (\) or double quotations (“) to avoid any of these problems. Lastly, I just called each Job with their respective prompts, errors, model name, and IP values and printed their outputs for now, since JSON parsing was complicated due to the inconsistencies that each LLM produced.