Alex Chen

Dr. Clark

12/9/2024

Autonomous C++ Code Assistant Agent Report

The C++ Code Agent is built using the content from the previous 4 labs of the course, capable of fixing user’s C++ files with errors, along with comments of how the errors are being fixed using OpenAI’s gpt4 model. I used langchain to connect to OpenAI’s LLM through simple JavaScript codes with my OpenAI API key.

The Agent is created by passing a prompt to create FlowScript, a Domain Specific Language (DSL) based on DOT language, used to describe process flows and execution dependencies of processes within the multithreaded job system. Each process in FlowScript is defined as a single job. A Job with dependencies can only be executed after the jobs it depended on are finished.

I wanted to generate a FlowScript program with 3 jobs—compileJob1, llmJob, and compileJob2. The first compileJob will compile the files that the user passed in, generate and parse the errors to errors.json. Which then llmJob is going to fix the code by sending AI errors.json, and the second compile job will compile and check if there still have errors.

Below is the zero-shot training for generating the FlowScript program:

Table -Zero Shot Training for FlowScript Generation

|  |
| --- |
| Prompt: FlowScript is a Domain Specific Language based on DOT language, each node in FlowScript represents a single job, and a job can have dependency to another job, represented by ->. Now, create a FlowScript program with three jobs: compileJob1, llmJob, and compileJob2. Ensure that llmJob has a dependency on compileJob1, and compileJob2 has a dependency on llmJob." |
| Result: compileJob1  llmJob -> compileJob1  compileJob2 -> llmJob |

I told AI what FlowScript is, and what kind of FlowScript I wanted the AI to generate, but the AI is generating the wrong thing since I did not give it any example. So in the second try, I gave it an example of FlowScript in the prompt, and it returns the exact FlowScript program that I need, as shown below:

Table -Second Training of FlowScript Generation

|  |
| --- |
| Prompt: FlowScript is a Domain Specific Language based on DOT language, where each node represents a single job, and job dependencies are denoted by the '->' arrow. Consider the following example FlowScript program: digraph G { jobA -> jobB; jobB -> jobC; }. In this instance, jobB depends on jobA, and jobC depends on jobB. Now, create a FlowScript program with three jobs: compileJob1, llmJob, and compileJob2. Ensure that llmJob has a dependency on compileJob1, and compileJob2 has a dependency on llmJob. |
| Result: digraph G {  compileJob1 -> llmJob;  llmJob -> compileJob2;  } |

The Agent received and parsed the FlowScript generated by AI to FS.json, which is going to be used for the FlowScript Interpreter to run the job system.

Inside the Agent, a job system is being created first, then two worker threads are created, one used for compileJob, and one used for llmJob. After that, jobs used in FlowScript are registered with a Json object called input. For compileJobs, the Json input’s content being passed are: newJobType (Name of the compileJob), jobChannel (0xF0000000), folder (errorCode), jobType(any number). When a compileJob is created and executed later in the FlowScript Interpreter, it will run a command on terminal to compile the files in the given folder. When the compileJob is finished, if there are compile errors through the given folder, it will print on the console that there are errors in the folder and parse the errors to errors.json. Otherwise, compileJob will print on the console that no errors exist and create result.json with the word Valid inside it. So that after the second compileJob is done, the agent will check if result.json exist. If the file exists, the program will end. If the file does not exist, the agent will create the jobs, run the job system again until the C++ codes are fixed. Below is how compileJob is registered/created:

Table -Documentation of compileJob

|  |  |
| --- | --- |
| Registration of compileJob in Agent class | Create/Execute/Finish of compileJob in interpreter class |
| input["newJobType"] = "compileJob1";  input["jobChannel"] = 0xF0000000;  input["folder"] = "errorCode";  input["jobType"] = 1;  js->registerNewJobType(input, CustomJob::CreateCustomJob); | input["newJobType"] = job;  input["folder"] = jobFolders[jobID++];  js->CreateNewJob(input);  input["jobID"] = finishJobID++;  js->FinishJob(input); |

For llmJob, the Json input’s content being passed are: newJobType (Name of the compileJob), jobChannel (0x0000000F), folder (Data/errors.json), jobType(any number), IPAddress(127.0.0.1), and prompt. I wanted a prompt that make the AI return a Json file that contains the necessary information that I need to fix the code with comments added to the fixed code, below is my zero-shot training:

Table -Zero Shot Training of llm Prompt

|  |
| --- |
| Prompt: Given a JSON file called errors.json representing errors in C++ code, where each error is accompanied by a Code Snippet providing the error line along with 2 lines above/below it, correct the error lines. Provide a JSON file with two objects: correctedCodes and comments. correctedCodes should have one corrected line as an index, and comments should include a comment for each corrected line, explaining how it was fixed. errors.json content |
| Result: {  "correctedCodes": {  "./Code/errorCode/TestCodes.cpp": [  {  "Row": "5",  "CorrectedCode": " val = value;"  }  ],  "./Code/errorCode/automated.cpp": [  {  "Row": "6",  "CorrectedCode": " return 0;"  },  {  "Row": "7",  "CorrectedCode": " return 0;"  }  ]  },  "comments": {  "./Code/errorCode/TestCodes.cpp": [  {  "Row": "5",  "Comment": "Added a semicolon at the end of the line to fix the 'expected ';' after expression' error."  }  ],  "./Code/errorCode/automated.cpp": [  {  "Row": "6",  "Comment": "Corrected the spelling of 'retrn' to 'return' to fix the 'use of undeclared identifier 'endl'; did you mean 'std::endl'?' error."  },  {  "Row": "7",  "Comment": "Corrected the spelling of 'retrn' to 'return' to fix the 'use of undeclared identifier 'retrn'' error."  }  ]  }  } |

However, the returned Json is not quite the same format that I wanted, so I add another example of the Json format I wanted in the prompt at my second try, and everything works perfectly:

Table -Second training of llm Prompt

|  |
| --- |
| Prompt: Given a JSON file called errors.json representing errors in C++ code, where each error is accompanied by a Code Snippet providing the error line along with 2 lines above/below it, correct the error lines. Provide a JSON file with two objects: correctedCodes and comments. correctedCodes should have one corrected line as an index, and comments should include a comment for each corrected line, explaining how it was fixed. I will show you errors.json first, then the json format that I would like you to return. Your final output should be a properly formatted json object with no additional information. errors.json content  {  "correctedCodes": {  "./Code/errorCode/TestCodes.cpp": {  "RowNumber": "corrected line of code"  },  "./Code/errorCode/automated.cpp": {  "RowNumber": "corrected line of code",  "RowNumber": "corrected line of code"  }  },  "comments": {  "./Code/errorCode/TestCodes.cpp": {  "RowNumber": "comment for the corrected line at same row in same file"  },  "./Code/errorCode/automated.cpp": {  "RowNumber": "comment for the corrected line at same row in same file",  "RowNumber": "comment for the corrected line at same row in same file"  }  }  } |
| Result:  {  "correctedCodes": {  "./Code/errorCode/TestCodes.cpp": {  "5": "val = value;"  },  "./Code/errorCode/automated.cpp": {  "6": "std::cout << \"This is the automated file\" << std::endl;",  "7": "return 0;"  }  },  "comments": {  "./Code/errorCode/TestCodes.cpp": {  "5": "Added semicolon at the end of the line.",  },  "./Code/errorCode/automated.cpp": {  "6": "Changed endl to std::endl.",  "7": "Fixed the spelling of 'return'."  }  }  } |

This is how prompt is created/modified for llmJob. Below is the documentation of llmJob:

Table -llm Job Documentation

|  |  |
| --- | --- |
| Registration of llmJob in Agent class | Create/Execute/Finish of compileJob in interpreter class |
| input["newJobType"] = "llmJob";  input["jobChannel"] = 0x0000000F;  input["folder"] = "Data/errors.json";  input["prompt"] = "Given a JSON file called errors.json representing errors in C++ code, where each error is accompanied by a Code Snippet providing the error line along with 2 lines above/below it, correct the error lines. Provide a JSON file with two objects: correctedCodes and comments. correctedCodes should have one corrected line as an index, and comments should include a comment for each corrected line, explaining how it was fixed. I will show you errors.json first, then the json format that I would like you to return. Your final output should be a properly formatted json object with no additional information. ";  input["jobType"] = 2;  input["IPAddress"] = "127.0.0.1";  js->registerNewJobType(input, CustomLLMJob::CreateCustomJob); | input["newJobType"] = job;  input["folder"] = jobFolders[jobID++];  js->CreateNewJob(input);  input["jobID"] = finishJobID++;  js->FinishJob(input); |

After jobs are registered in the jobs system in the sequence of compileJob1, llmJob, and compileJob2. The Agent create the FlowScript through the llm and pass the FlowScript to the FlowScript Interpreter to run these jobs in the job System. The agent will check if the code is fixed through whether result.json is created, if not, the job system would register/create/run jobs again until the code is fixed.

Here is a demonstration video about how the process is runned: <https://youtu.be/ZNq-qZoMZyk?si=l_dgk421IpAMrt9I>, and below shows how the job system is runned through terminal.

Table -Documentation of running of job system through terminal

|  |
| --- |
| alexchen@DESKTOP-2U69Q2E:~/final-autonomous-code-assistant-agent-alexchans$ ./app  Creating Job System  Creating Worker Thread  Job 0 Has Been Executed  Compile Job 0 Return Code: 256  Errors detected while compiling errorCode folder and errors has been parsed to errors.json  Job 1 Has Been Executed  Job 2 Has Been Executed  Compile Job 2 Return Code: 0  No Errors detected while compiling errorCode folder  Total jobs completed 3  Job system is destroyed |

The example files with errors and the changes AI applied to fix the errors are shown as below:

Table -Documentaion of Example Files Used

|  |  |
| --- | --- |
| Files with errors before AI fixed the code: | Files with errors after AI fixed the code: |
| automated.cpp:  #include <iostream>  #include "TestCodes.h"  int main()  {  std::cout << "This is the automated file" << endl;  retrn 0;  } | automated.cpp:  #include <iostream>  #include "TestCodes.h"  int main()  {  std::cout << "This is the automated file" << std::endl; //Added 'std::' before 'endl' to fix use of undeclared identifier error  return 0; //Changed 'retrn' to 'return' to fix use of undeclared identifier error  } |
| TestCodes.cpp:  #include "TestCodes.h"  void TestCodes::setVal(int value)  {  val = value  }  int TestCodes::getVal()  {  return val;  } | TestCodes.cpp:  void TestCodes::setVal(int value)  {  val = value; //Added semicolon at end of line to fix expected ';' error  }  int TestCodes::getVal()  {  return val;  } |

Below is the errors.json produced by compileJob:

|  |
| --- |
| {  "./Code/errorCode/TestCodes.cpp": [  {  "Code Snippet": [  [  "void TestCodes::setVal(int value)"  ],  [  "{"  ],  [  " val = value"  ],  [  "}"  ],  [  ""  ]  ],  "Column": [  "16"  ],  "Error": [  "expected ';' after expression"  ],  "File": [  "./Code/errorCode/TestCodes.cpp"  ],  "Row": [  "5"  ]  }  ],  "./Code/errorCode/automated.cpp": [  {  "Code Snippet": [  [  "int main()"  ],  [  "{"  ],  [  " std::cout << \"This is the automated file\" << endl;"  ],  [  " retrn 0;"  ],  [  "}"  ]  ],  "Column": [  "50"  ],  "Error": [  "use of undeclared identifier 'endl'; did you mean 'std::endl'?"  ],  "File": [  "./Code/errorCode/automated.cpp"  ],  "Row": [  "6"  ]  },  {  "Code Snippet": [  [  "{"  ],  [  " std::cout << \"This is the automated file\" << endl;"  ],  [  " retrn 0;"  ],  [  "}"  ]  ],  "Column": [  "5"  ],  "Error": [  "use of undeclared identifier 'retrn'"  ],  "File": [  "./Code/errorCode/automated.cpp"  ],  "Row": [  "7"  ]  }  ]  } |

In conclusion, the C++ agent I build is capable of fixing errors of C++ files that user put in error folder and provide useful feedback on how the code is fixed. The agent use OpenAI’s LLM to communicate with the gpt-4 model, and the agent fix the code by running compileJob and llmJob through a multithreaded job system until the code is being fixed.