Running Dev Journal for Lab 4, Job System & LLM communication lab

Run the program from the root directory with:

1. Make compile
2. Make run

Nov 24, 2023

- Working on coding section
    - Restserver.py -- communicates with GPT4ALL application's built in rest API
    - Communicate together to establish a connection with gpt4all on local computer
- Switched to Python 3.9.12 for OPENSSL support
- pip install openai==0.28 ← downgrade openai import for LLMCall.py so that openai.completion can be accessed by the LLMCall.py file. Downgrade over migrate to preserve current code base.
- Used system("Python3 ./Code/LLMCall.py");
    - Calls LLMCall.py from jobsystem with system command. This runs the "python3 ./Code/LLMCall.py in the terminal".
- Outputs the api response in a json file called "ApiResponse.json"

Nov 27, 2023

- Communicating between LLMCall.py and Customjob.cpp with system and args.slice commands
- Generic function in customjob.cpp for file reading and storing as serialized string json object

```
-    std::string read(const std::string& path);
```

```
std::string CustomJob::read(const std::string& filePath){//reads the file and stores the output in a json object
```

```
    // String to store file content
    std::string file_content;

    // Read the JSON file
    std::ifstream json_file(filePath);
    if (json_file.is_open()) {
        file_content.assign((std::istreambuf_iterator<char>(json_file)),
                            std::istreambuf_iterator<char>());
        json_file.close();
    } else {
        std::cerr << "Unable to open JSON file." << std::endl;
    }

    // Parse the JSON data
    json json_data = json::parse(file_content);

    // TODO: Reformat json_data as needed

    // Serialize JSON object to string
    std::string jsonObject = json_data.dump(4); // '4' for pretty printing

    return jsonObject; //return the json object


}
```

Changed job.h from private to public with job threads to enable new compilejob to run:

```
class Job{
    friend class JobSystem;
    friend class JobWorkerThread;
    public:
    Job(unsigned long jobChannels = 0xFFFFFFFF, int jobType = -1) : m_jobChannels(jobChannels),
m_jobType(jobType) {
        static int s_nextJobID = 0;
```

```
        m_jobID = s_nextJobID++;
    }

    virtual ~Job() {}

    virtual void Execute() = 0; //body set = 0 so purely virtual, abstract class. Must inherent the execute function
    virtual void JobCompletedCallback() {} //do not have to implement this function (body not set to zero, so if implemented then does nothing)
                                           //can call if want to.
    int GetUniqueID() const { return m_jobID; } //job type that returns will be a const so cannot be manipulated

    void setUniqueID(int id) { m_jobID = id; } //set job id to id passed in inside the job system API dependency function to match new dependency order. For job status to update correctly.
    void signalCompletion() {//signals job complete
        std::lock_guard<std::mutex> lock(jobMutex);
        condition = true;
        cv.notify_one();
    }
```

CompileJob instantiation in jobsystemapi.cpp:

```
if(jobType == "compilejob"){
            std::cout << "Creating CompileJob" << std::endl;
            std::string directoryToMakefile = "makefile"; //explanatory var for makefile directory
            //created a compile job
            //TODO - [OLD INSTANTIATION] CompileJob* cjb = new CompileJob(0xFFFFFFFF, 1, directoryToMakefile); //pass in directory to makefile to run make command in cjb
            //create a json object with properties for compile job
            json jsonInput;
            jsonInput["threadName"] = "Thread1";
            jsonInput["threadChannel"] = 0xF0000000;
            jsonInput["threadName"] = "Thread2";
            jsonInput["threadChannel"] = 0x0000000F;
            CompileJob* cjb = new CompileJob(jsonInput); //pass in directory to makefile to run make command in cjb
            jobMutex.lock(); //lock mutex for job system API
            //jobsForAPI.push_back(cjb); //add compile job to jobs vector
            jobsForAPI[jobType] = cjb; //add compile job to jobs map
```

```
        jobOrder.push_back(jobType); //add job string name to job order vector
        jobMutex.unlock(); //unlock mutex for job system API
    }
```

11/28 – work

- job.h:

```
-       virtual int Execute(); //TODO - took away 0 so purely virtual
```

- Compilejob.cpp:

```
- int CompileJob::JobCompletedCallback(){
-     std::cout << "CompileJob done" << std::endl;
-     //notify completion
-    //signalCompletionForBusyWhile(); //TODO - this causes seg fault because of new way compilejob is done... fix this.... or wait in main with a busy while etc.
-     int signalCompletion = 1;
-     return signalCompletion;
-
- }
```

- Realized that output.json in data actually has the errors with source code so that work with compilejob was irrelevant. Returned compilejob.h/.cpp to normal below:
    - .h:

```
- #include "job.h"
- #include <future>
- #include "jobsystem.h"
- #include <queue> //for basic thread control
- //could have taken everything in compilejob.cpp and put it on top of compilejob.h instead
- //in general should put dependencies in .cpp files instead of .h files
- //if you put dependencies in .h files, then everything that includes that .h file will have to include those dependencies
- //pragma once prevents children from calling imports again
- //when working on large code bases, when making changes in .h have to change every single .cpp file that includes that .h file
- //if dependencies are in .cpp files, then only have to change .cpp file that includes that .h file -> faster compile times
- //when dont need it there then dont include it, use forward declarations etc.
```

```
-    //keep dependencies in .cpp files (unless creating object in .h, then need them in .h)
-    class CompileJob : public Job{
-        private:
-        std::string makefile; //string for grabbing makefile path
-
-        public:
-        CompileJob(unsigned long jobChannels, int jobType, const std::string & passedInMakeFile):
     Job(jobChannels, jobType), makefile(passedInMakeFile) {}; //constructor - TODO - added passing jobs
     vector to constructor to add parsing job to jobs vector
-        ~CompileJob() {};
-
-        std::string output; //string for output added
-        int returnCode;  //return code added
-        void Execute() override;
-        void JobCompletedCallback() override;
-
-
-    };
```

- .cpp:

```
#include "compilejob.h"
#include <iostream>
#include <string>
#include <array>
#include "job.h"
#include <sstream> //for stringstream to split output into words by spaces
#include "fstream" //for writing to file
#include "json.hpp"  //for json file writing
using json = nlohmann::json; //for json manipulation

void CompileJob::Execute(){
    std::array< char, 128 > buffer; //buffer to store output
    std::string command = "make -f " + makefile + " automated"; //path to makefile and command to run makefile based
on file path

    //Redirect cerr to cout - easy way to see errors when working with multiple threads
```

```
    //then can capture cout and get all errors
    command.append(" 2>&1"); //redirects stderr to stdout
    //spin up a thread that is going to grab code, compile it, and return a result
    //Terminal tells you there is an error in code, but with threads you cannot tell where the error is
    //cout to terminal is not thread safe (standard commands like that are not thread safe)

    //open pipe and run command
    FILE* pipe = popen(command.c_str(), "r"); //opens a pipe and will send something to it
    //gives terminal to work on, but this terminal is open inside a thread
    //turns command into a c string and sends it to the pipe
    //you can only open pipe to read in this case "r"

    if(!pipe){
        std::cout << "Pipe failed to open" << std::endl;
        return;
    }

    //read till end of process:
    while(fgets(buffer.data(), 128, pipe) != NULL){//reading buffer until get to end of file - same concept as finishing
reading in terminal
        this->output.append(buffer.data()); //append to output string
    }
    //close pipe and get return code
    this->returnCode = pclose(pipe); //close pipe and get return code. Fine = 0, error = returns offset other than 0

    //split output and feed into stream //TODO - take out stream variable? Not needed anymore if we are writing to file
    std::stringstream ss(this->output); // Create stringstream from output string
    nlohmann::json j; // Create JSON object using nlohmann::json

    std::string transcribeOut; // String to store lines from output string
    while(std::getline(ss, transcribeOut, '\n')) { // While there are lines in the stringstream
        j["outputLines"].push_back(transcribeOut);  // Add line to JSON object under "outputLines" key
    }

    // Write JSON object to a file named compileJobOutput.json
    std::ofstream fileOutput("Data/compileJobOutput.json");
    fileOutput << j.dump(4);  // 4 spaces as indent
```

```
    fileOutput.close();

    std::cout << "CompileJob [ID: " << this->GetUniqueID() << "] has been executed." << std::endl;
    JobCompletedCallback();
}

void CompileJob::JobCompletedCallback(){
    std::cout << "CompileJob done" << std::endl;
    //notify completion
    signalCompletion(); //TODO - encapsulated version of thread control with condvar

}




//code automatically compiling means you can:
//set code off and do other things
//automated regression testing -> compile code and automatically run tests on code.
//when errors output it tells you error number and error messages -> put error messages into LLM's to fix errors
//take code file and web info from LLM and say "fix it for me"
//AI TOOL TO FIX CODE FOR YOU -> BETTER STANDARDIZATION = BETTER FIXES
//Can write as terminal job instead of compile job (written as compile job here)
```

In customjob.cpp:

```
//TODO no longer need this line if code contains sourceCode? - std::string sourceCode = readAllFilesInDirectory("./test-code/");
```

Current AI prompt:

```
std::string json_file_path = "./Data/output.json";

   std::string code = processFile(json_file_path); //read the json file and store the output in a string called code
```

```
   std::string prompt = "Answer question based on the context. Context: Here is code in a json file format with errors. Please fix these errors and add a description of what you did to fix the errors. Question: " + code;
```

calling LLM with slicing:

```
std::string command = "python3 ./Code/LLMCall.py ./Code/llm \"" + prompt + "\"";
   system(command.c_str()); //call to python script to run LLM and output results to a file called "LLMOutput.txt" in the Data folder.
```

LLMCall.py:

```
import openai
import json # for getting errors json file from lab one
import sys # for getting prompt from customjob.cpp

# Set the base URL to your local server
openai.api_base = "http://localhost:4891/v1" #TODO - pass local host 4891 to LLMCall.py from customjob.cpp

# Since it's a local server, API key is not required
openai.api_key = "not needed for a local LLM"



# print("\n CODE: " + code + "\n"); #TODO - remove this line. For testing purposes only
# Set up the prompt and other parameters for the API request
#prompt = "Answer question based on the context. Context: Here is code in a json file format with errors. Please fix these errors and add a description of what you did to fix the errors. Question: " + code;
args = sys.argv[2:]
```

```
print("\nthis is the args:", args, "\n")

prompt = args; #set prompt to args from customjob.cpp
# Specify the model you want to use
model = "Minstrel OpenOrca"

# Make the API request
response = openai.Completion.create(
    model=model,
    prompt=prompt,
    max_tokens=5000,
    temperature=0.28,
    top_p=0.95,
    n=1,
    echo=True,
    stream=False
)

# Now, write the response to a JSON file
output_file_path = "./Data/ApiResponse.json"
with open(output_file_path, "w") as output_file:
    json.dump(response, output_file, indent=4)
```

-passing output.json to LLM so that it has the source code to read

LLMCall.py update slicing to grab prompt:

# Get the prompt from command line arguments
prompt = sys.argv[1] if len(sys.argv) > 1 else "Default prompt if not provided"

- Fix “code” appended to prompt with this function in customjob.cpp:

```
-
-    std::string CustomJob::escapeShellArgument(const std::string& arg) {
```

```
-         std::string escaped;
-         for (char c : arg) {
-             // Add a backslash before special characters
-             if (c == '"' || c == '\\' || c == '$') {
-                 escaped += '\\';
-             }
-             escaped += c;
-         }
-         return escaped;
-     }
```

Deals with special characters

Nov 29 – Wednesday

Added function to LLMCall.py to easily run multiple models. Takes model name as input and prompt and returns response to be printed/logged in .txt file:

```
def runModel(modelName, prompt):
    # Specify the model you want to use and make the API request
    model = modelName
    try:
        response = openai.Completion.create(
            model=model,
            prompt=prompt,
            max_tokens=5000,
            temperature=0.28,
            top_p=0.95,
            n=1,
            echo=True,
            stream=False
        )
        return response
    except Exception as e:
        print("An error occurred:", e)
        return None
```

Main.cpp set ip and prompt in customjob LLM with getLLMJob function and setters…

```
jobSystem.getLLMJob()->SetPrompt(promptToPassIn); //returns LLM job type for manipulation
    jobSystem.getLLMJob()->SetIP("Dummy-there-is-no-ip"); //sets the IP value for the LLM job type. In this case localHost 4891 port #.
```

Job.h: added setters and variables for ip/prompt

```
//promptFromMain & IPFromMain are variables that are private to the job.h class, but that are set from the prompt/ip values in main.cpp.
    void SetPrompt(const std::string& promptPassedIn) {promptFromMain = promptPassedIn; } //set prompt for custom job

    void SetIP(const std::string& ipPassedIn) {ipFromMain = ipPassedIn; } //set ip for custom job

    //prompt/ip public so that customjob.cpp can access them
    std::string promptFromMain; //for storing prompt string
    std::string ipFromMain; //for storing ip string
```

added response.txt files for both the mistral instruct and mistral OpenOrca AI responses in the ./Data/ directory

Now writing both AI model outputs to .json file:

```
# can either use "Minstrel OpenOrca" or "GPT4All Falcon"
response = runModel(modelName, prompt)

if(modelName == "Mistral OpenOrca"):
    output_file_path = "./Data/OpenOrca-Response.json"
    print("\n\nMistral OpenOrca model output path!\n\n")
```

```
elif(modelName == "Mistral Instruct"):
    output_file_path = "./Data/MistralInstruct-Response.json"
    print("\n\nMistral Instruct model output path!\n\n")
else:
    print("\n\nERROR: Model name not recognized!\n\n")
# Now, write the response to a JSON file
with open(output_file_path, "w") as output_file:
    json.dump(response, output_file, indent=4)
```

Now trying to wrap json text with text_wrap_json:

```
 # wrap function to wrap text field in json file to 80 characters per line
def wrap_text_in_json(response, width=80):
    if 'choices' in response and response['choices']:
        for choice in response['choices']:
            if 'text' in choice:
                original_text = choice['text']
                wrapped_text = textwrap.fill(original_text, width=width)
                choice['text'] = wrapped_text
    return response
```

Does not consider new lines so does not wrap AI response in json, change text_wrap_function to consider newlines:

```
# wrap function to wrap text field in json file to 80 characters per line - considers newlines
def wrap_text_in_json(response, width=80):
    if 'choices' in response and response['choices']:
        for choice in response['choices']:
            if 'text' in choice:
                # Split the text into lines, wrap each line, and then rejoin with newlines
                original_lines = choice['text'].split('\n')
                wrapped_lines = [textwrap.fill(line, width=width) for line in original_lines]
```

```
            choice['text'] = '\n'.join(wrapped_lines)
    return response
```

Scrapping text_wrap_function and can just wrap the .json if it is ever output later on or displayed elsewhere before displaying.

Can now run both models. Uncomment either line 45 or 46 to run whichever model you want in LLMCall.py file:

```
modelName = "mistral-7b-openorca.Q4_0.gguf"
# modelName = "mistral-7b-instruct-v0.1.Q4_0.gguf"
```

By calling the models with the unique file names (.gguf), GPT4ALL switches AI models depending on which is called.

The prompt I am using is:

```
    std::string promptToPassIn = "Answer question based on the context. Context: Here is code in a json file format with errors. Please fix these errors and add a description of what you did to fix the errors. Question: ";
```

This prompt breaks down the question into context and a question. The context section gives the AI more knowledge to work with by telling I thtat the question it will be asked is a coding segment with errors and that it needs to fix the errors. This is an attempt to make the prompt universally applicable across all coding errors because the only element that needs to change is the “code” variable that is appended later. This prompt is passed into customjob.cpp.

This prompt has the code appended to it in customjob.cpp here:

```
    // JSON file path
    std::string json_file_path = "./Data/output.json";
```

```
std::string code = processFile(json_file_path); //read the json file and store the output in a string called code
code = escapeShellArgument(code); //escape the code string to be used in the command string below
```

```
std::string prompt = promptFromMain + code; //append code to prompt string
```

The code holds this output.json file content here:


```
[
  [
    {
      "Column Number": 4,
      "Error Description": "expected expression",
      "File Path": "test-code/test-set-one.cpp",
      "Line Number": 6,
      "Source Code Causing Error": [
        "",
        "int main(){",
        "   << \"test_set_one\" << std::endl",
        "    //another_function();",
        ""
      ]
    }
  ],
  [
    {
      "Column Number": 2,
      "Error Description": "invalid preprocessing directive",
      "File Path": "test-code/test-set-three.cpp",
      "Line Number": 1,
      "Source Code Causing Error": [
        "#incl",
        "#include \"test-set-three.h\"",
        "void test_set_three() {"
      ]
```

```
    },
    {
      "Column Number": 5,
      "Error Description": "use of undeclared identifier 'std'",
      "File Path": "test-code/test-set-three.cpp",
      "Line Number": 4,
      "Source Code Causing Error": [
        "#include \"test-set-three.h\"",
        "void test_set_three() {",
        "    std::cout << \"test_set_three\" << std::endl;",
        "}"
      ]
    },
    {
      "Column Number": 38,
      "Error Description": "use of undeclared identifier 'std'",
      "File Path": "test-code/test-set-three.cpp",
      "Line Number": 4,
      "Source Code Causing Error": [
        "#include \"test-set-three.h\"",
        "void test_set_three() {",
        "    std::cout << \"test_set_three\" << std::endl;",
        "}"
      ]
    }
  ],
  [
    {
      "Column Number": 1,
      "Error Description": "expected '}'",
      "File Path": "test-code/test-set-two.cpp",
      "Line Number": 7,
      "Source Code Causing Error": [
        "",
        "   std::cout << \"test_set_two\" << std::endl; ",
        ""
      ]
```

```
    },
    {
      "Column Number": 20,
      "Error Description": "to match this '{'",
      "File Path": "test-code/test-set-two.cpp",
      "Line Number": 4,
      "Source Code Causing Error": [
        "#include <iostream>  ",
        "",
        "void test_set_two(){",
        "",
        "   std::cout << \"test_set_two\" << std::endl; "
      ]
    }
  ]
]
```

Then the prompt is put into the system command to call the LLM

```
std::string command = "python3 ./Code/LLMCall.py \"" + prompt + "\"";
```

The LLMCall.py file writes the model output to either OopenOrca-Response.json or MistralInstruct-Response.json depending on which model is used:

OpenOrca-Response.json:

```
{
  "choices": [
    {
      "finish_reason": "stop",
      "index": 0,
      "logprobs": null,
      "references": [],
```

```
"text": "Answer question based on the context. Context: Here is code in a json file format with errors. Please fix these errors and add a description of what you did to fix the errors. Question: [\n  [\n  {\n  \"Column Number\": 4,\n  \"Error Description\": \"expected expression\",\n  \"File Path\": \"test-code/test-set-one.cpp\",\n  \"Line Number\": 6,\n  \"Source Code Causing Error\": [\n  \"\",\n  \"int main(){\",\n  \"  << \\\"test_set_one\\\" << std::endl\",\n  \"  //another_function();\",\n  \"\"\n ]\n  }\n ],\n [\n  {\n  \"Column Number\": 2,\n  \"Error Description\": \"invalid preprocessing directive\",\n  \"File Path\": \"test-code/test-set-three.cpp\",\n  \"Line Number\": 1,\n  \"Source Code Causing Error\": [\n  \"#incl\",\n  \"#include \\\"test-set-three.h\\\"\",\n  \"void test_set_three() {\"\n  ]\n  },\n  {\n  \"Column Number\": 5,\n  \"Error Description\": \"use of undeclared identifier 'std'\",\n  \"File Path\": \"test-code/test-set-three.cpp\",\n  \"Line Number\": 4,\n  \"Source Code Causing Error\": [\n  \"#include \\\"test-set-three.h\\\"\",\n  \"void test_set_three() {\",\n  \"  std::cout << \\\"test_set_three\\\" << std::endl;\",\n  \"}\"\n  ]\n  },\n  {\n  \"Column Number\": 38,\n  \"Error Description\": \"use of undeclared identifier 'std'\",\n  \"File Path\": \"test-code/test-set-three.cpp\",\n  \"Line Number\": 4,\n  \"Source Code Causing Error\": [\n  \"#include \\\"test-set-three.h\\\"\",\n  \"void test_set_three() {\",\n  \"  std::cout << \\\"test_set_three\\\" << std::endl;\",\n  \"}\"\n  ]\n  }\n ],\n [\n  {\n  \"Column Number\": 1,\n  \"Error Description\": \"expected '}'\",\n  \"File Path\": \"test-code/test-set-two.cpp\",\n  \"Line Number\": 7,\n  \"Source Code Causing Error\": [\n  \"\",\n  \"  std::cout << \\\"test_set_two\\\" << std::endl; \",\n  \"\"\n  ]\n  },\n  {\n  \"Column Number\": 20,\n  \"Error Description\": \"to match this '{'\",\n  \"File Path\": \"test-code/test-set-two.cpp\",\n  \"Line Number\": 4,\n  \"Source Code Causing Error\": [\n  \"#include <iostream>  \",\n  \"\",\n  \"void test_set_two(){\",\n  \"\",\n  \"  std::cout << \\\"test_set_two\\\" << std::endl; \"\n  ]\n  }\n ]\n]\nHere is the corrected JSON file with descriptions of what was fixed in each error.\n\n[\n  [\n  {\n  \"Column Number\": 4,\n  \"Error Description\": \"expected expression\",\n  \"File Path\": \"test-code/test-set-one.cpp\",\n  \"Line Number\": 6,\n  \"Source Code Causing Error\": [\n  \"\",\n  \"int main(){\",\n  \"  << \\\"test_set_one\\\" << std::endl;\",\n  \"  //another_function();\",\n  \"\"\n  ],\n  \"Description of Fix\": \"Removed the extra space before '<<' and added a semicolon at the end.\"\n  }\n ],\n [\n  {\n  \"Column Number\": 2,\n  \"Error Description\": \"invalid preprocessing directive\",\n  \"File Path\": \"test-code/test-set-three.cpp\",\n  \"Line Number\": 1,\n  \"Source Code Causing Error\": [\n  \"#incl\",\n  \"#include \\\"test-set-three.h\\\"\",\n  \"void test_set_three() {\"\n  ],\n  \"Description of Fix\": \"Corrected the preprocessing directive by changing 'incl' to '#include'\"\n  },\n  {\n  \"Column Number\": 5,\n  \"Error Description\": \"use of undeclared identifier 'std'\",\n  \"File Path\": \"test-code/test-set-three.cpp\",\n  \"Line Number\": 4,\n  \"Source Code Causing Error\": [\n  \"#include \\\"test-set-three.h\\\"\",\n  \"void test_set_three() {\",\n  \"  std::cout << \\\"test_set_three\\\" << std::endl;\",\n  \"}\"\n  ],\n  \"Description of Fix\": \"Added the missing namespace 'std' before 'cout'\"\n  },\n  {\n  \"Column Number\": 38,\n  \"Error Description\": \"use of undeclared identifier 'std'\",\n  \"File Path\": \"test-code/test-set-three.cpp\",\n  \"Line Number\": 4,\n  \"Source Code Causing Error\": [\n
```

\"#include \\\"test-set-three.h\\\"\",\n \"void test_set_three() {\",\n \" std::cout << \\\"test_set_three\\\" << std::endl;\",\n \"}\"\n],\n \"Description of Fix\": \"Added the missing namespace 'std' before 'cout'\"\n }\n],\n [\n {\n \"Column Number\": 1,\n \"Error Description\": \"expected '}'\",\n \"File Path\": \"test-code/test-set-two.cpp\",\n \"Line Number\": 7,\n \"Source Code Causing Error\": [\n \"\",\n \" std::cout << \\\"test_set_two\\\" << std::endl; \",\n \"\"\n],\n \"Description of Fix\": \"Added a closing '}' after the last line.\"\n },\n {\n \"Column Number\": 20,\n \"Error Description\": \"to match this '{'\",\n \"File Path\": \"test-code/test-set-two.cpp\",\n \"Line Number\": 4,\n \"Source Code Causing Error\": [\n \"#include <iostream>\",\n \"\",\n \"void test_set_two(){\",\n \"\",\n \" std::cout << \\\"test_set_two\\\" << std::endl; \"\n],\n \"Description of Fix\": \"Added a missing opening '{' after the function declaration.\"\n }\n]\n]"
        }
    ],
    "created": 1701298875,
    "id": "foobarbaz",
    "model": "Mistral OpenOrca",
    "object": "text_completion",
    "usage": {
        "completion_tokens": 864,
        "prompt_tokens": 768,
        "total_tokens": 1632
    }
}

MistralInstruct-Response.json:

{
    "choices": [
        {
            "finish_reason": "stop",
            "index": 0,
            "logprobs": null,
            "references": [],
            "text": "Answer question based on the context. Context: Here is code in a json file format with errors. Please fix these errors and add a description of what you did to fix the errors. Question: [\n [\n {\n \"Column Number\": 4,\n \"Error Description\": \"expected expression\",\n \"File Path\": \"test-code/test-set-one.cpp\",\n \"Line Number\": 6,\n \"Source Code Causing Error\": [\n \"\",\n \"int

main(){\",\n \" << \\\"test_set_one\\\" << std::endl\",\n \" //another_function();\",\n \"\"\n]\n }\n],\n [\n {\n \"Column Number\": 2,\n \"Error Description\": \"invalid preprocessing directive\",\n \"File Path\": \"test-code/test-set-three.cpp\",\n \"Line Number\": 1,\n \"Source Code Causing Error\": [\n \"#incl\",\n \"#include \\\"test-set-three.h\\\"\",\n \"void test_set_three() {\"\n]\n },\n {\n \"Column Number\": 5,\n \"Error Description\": \"use of undeclared identifier 'std'\",\n \"File Path\": \"test-code/test-set-three.cpp\",\n \"Line Number\": 4,\n \"Source Code Causing Error\": [\n \"#include \\\"test-set-three.h\\\"\",\n \"void test_set_three() {\",\n \" std::cout << \\\"test_set_three\\\" << std::endl;\",\n \"}\"\n]\n },\n {\n \"Column Number\": 38,\n \"Error Description\": \"use of undeclared identifier 'std'\",\n \"File Path\": \"test-code/test-set-three.cpp\",\n \"Line Number\": 4,\n \"Source Code Causing Error\": [\n \"#include \\\"test-set-three.h\\\"\",\n \"void test_set_three() {\",\n \" std::cout << \\\"test_set_three\\\" << std::endl;\",\n \"}\"\n]\n }\n],\n [\n {\n \"Column Number\": 1,\n \"Error Description\": \"expected '}'\",\n \"File Path\": \"test-code/test-set-two.cpp\",\n \"Line Number\": 7,\n \"Source Code Causing Error\": [\n \"\",\n \" std::cout << \\\"test_set_two\\\" << std::endl; \",\n \"\"\n]\n },\n {\n \"Column Number\": 20,\n \"Error Description\": \"to match this '{'\",\n \"File Path\": \"test-code/test-set-two.cpp\",\n \"Line Number\": 4,\n \"Source Code Causing Error\": [\n \"#include <iostream> \",\n \"\",\n \"void test_set_two(){\",\n \"\",\n \" std::cout << \\\"test_set_two\\\" << std::endl; \"\n]\n }\n]\n]\nHere is the fixed code:\n```json\n[\n {\n \"Column Number\": 4,\n \"Error Description\": \"expected expression\",\n \"File Path\": \"test-code/test-set-one.cpp\",\n \"Line Number\": 6,\n \"Source Code Causing Error\": [\n \"\",\n \"int main(){\",\n \" << \\\"test_set_one\\\" << std::endl\",\n \" //another_function();\",\n \"\"\n]\n },\n {\n \"Column Number\": 2,\n \"Error Description\": \"invalid preprocessing directive\",\n \"File Path\": \"test-code/test-set-three.cpp\",\n \"Line Number\": 1,\n \"Source Code Causing Error\": [\n \"#incl\",\n \"#include \\\"test-set-three.h\\\"\",\n \"void test_set_three() {\"\n]\n },\n {\n \"Column Number\": 5,\n \"Error Description\": \"use of undeclared identifier 'std'\",\n \"File Path\": \"test-code/test-set-three.cpp\",\n \"Line Number\": 4,\n \"Source Code Causing Error\": [\n \"#include \\\"test-set-three.h\\\"\",\n \"void test_set_three() {\",\n \" std::cout << \\\"test_set_three\\\" << std::endl;\",\n \"}\"\n]\n },\n {\n \"Column Number\": 38,\n \"Error Description\": \"use of undeclared identifier 'std'\",\n \"File Path\": \"test-code/test-set-three.cpp\",\n \"Line Number\": 4,\n \"Source Code Causing Error\": [\n \"#include \\\"test-set-three.h\\\"\",\n \"void test_set_three() {\",\n \" std::cout << \\\"test_set_three\\\" << std::endl;\",\n \"}\"\n]\n },\n {\n \"Column Number\": 1,\n \"Error Description\": \"expected '}'\",\n \"File Path\": \"test-code/test-set-two.cpp\",\n \"Line Number\": 7,\n \"Source Code Causing Error\": [\n \"\",\n \" std::cout << \\\"test_set_two\\\" << std::endl; \",\n \"\"\n]\n },\n {\n \"Column Number\": 20,\n \"Error Description\": \"to match this '{'\",\n \"File Path\": \"test-code/test-set-two.cpp\",\n \"Line Number\": 4,\n \"Source Code Causing Error\": [\n \"#include <iostream> \",\n \"\",\n \"void test_set_two(){\",\n \"\",\n \" std::cout << \\\"test_set_two\\\" << std::endl; \"\n]\n }\n]\n```\nI made the following changes:\n\n* In `test-code/test-set-one.cpp`, I removed the extra curly braces and added a semicolon after the line that includes

```
the file.\n* In `test-code/test-set-three.cpp`, I fixed the preprocessing directive by removing the extra closing parenthesis.\n* In both files, I replaced the use of `std` with the correct identifier `std::`.\n* In `test-code/test-set-two.cpp`, I added a semicolon after the line that includes the file and fixed the mismatched curly braces by adding an opening brace on line 4."
      }
   ],
   "created": 1701298651,
   "id": "foobarbaz",
   "model": "Mistral Instruct",
   "object": "text_completion",
   "usage": {
      "completion_tokens": 837,
      "prompt_tokens": 765,
      "total_tokens": 1602
   }
}
```


The text contains the AI response with fixed code in the “text” section. The created field contains the unique ID for the session, the model type, the object, and the usage broken down by tokens.