Exploring the Viability of LLMs: Generating Python Bindings with Open Source and API-Based Models

09.18.2023 - 12.07.2023

Sandeep Yedla
Axle Informatics Inc.
6116 Executive Blvd Suite 400, Rockville, MD 20852

Overview

This project emphasizes exploring the capabilities of Large Language models in automating the generation of binding code. Traditional binding code often contains a substantial amount of boilerplate code, leading to increased development time for developers when writing these codes manually.

The solution powered by Large Language Models (LLMs) addresses this challenge by offering users options for C++ classes. Users can specify the desired class, initiate a pipeline, and subsequently generate the code along with the associated CMakeLists.txt. This approach aims to streamline the code generation process, reducing manual effort and enhancing overall development efficiency.

Problem Statement

The **problem at hand involves the creation of Python wrappers for a C++ API**, with the goal of seamless integration with Python.

The primary challenge lies in creating Python wrappers for a pre-existing C++ API. This involves the development of an interface that allows the C++ functionality to be accessed and utilized seamlessly from within a Python environment. This is often necessary when there is a desire to leverage existing C++ codebases or libraries within a Python-centric project or application.

Additionally, the objective is to extend compatibility beyond Python to include other programming languages such as Java, WASM and Julia.

Solution

A open source Large Language models (Code-LLM) powered solution to automate the generation of bindings code, reducing the manual effort required and minimizing the chances of errors

Helpful Links:

LLM Pipeline - GitHub repository link https://github.com/Sandeepyedla0/GenAI_bindings_pipeline

GitHub Repository link: https://github.com/Sandeepyedla0/Bindings-Data-Collection

Research Articles:

https://ai.meta.com/blog/llama-2-updates-connect-2023/

https://huggingface.co/WizardLM

https://huggingface.co/Phind/Phind-CodeLlama-34B-v2

https://www.preprints.org/manuscript/202307.2142/v2

https://www.together.ai/products#inference

Important files:

GitHub link: https://github.com/Sandeepyedla0/GenAI_bindings_pipeline

clang_parser.py - Consists of project parser script that extracts Class names

extract_class.py - Consists of class definition extraction script

Opensource_codellm.py - Consists of Input prompt generation, NCAT server deployed model function and together.ai API request function

Bindings_pipeline.py - The pipeline that connects all the modules.

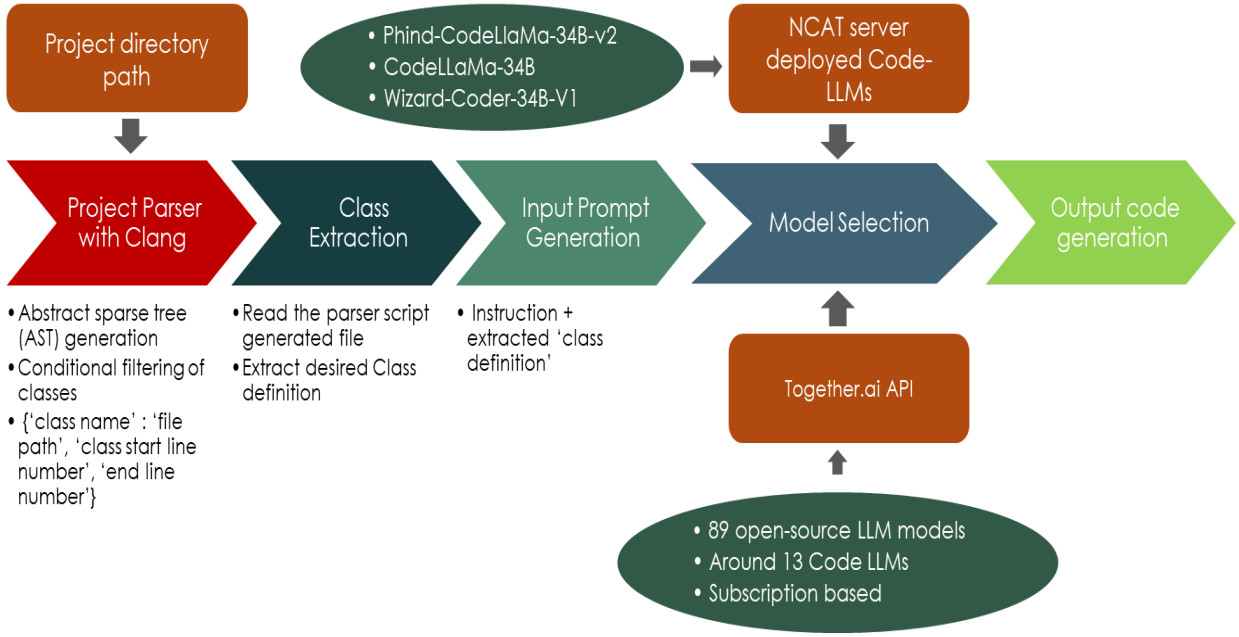
Steps to Setup:

For NCAT deployed models, hardware requirement : LLM Server with enough RAM (96 CPU, 768 Gb RAM)

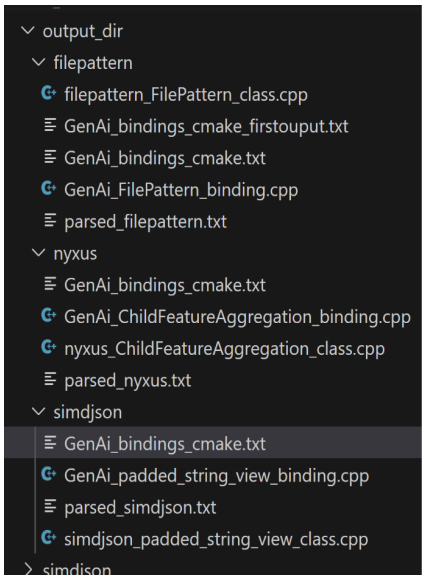
Each pre-trained checkpoint size is around 80 - 90gb, for 3 models, at least 300 gb of memory is required.

1. git clone https://github.com/Sandeepyedla0/GenAI_bindings_pipeline
2. Create a new conda environment: conda create --name your_env_name
3. Activate conda environment
4. Install requirements.txt : pip install requirements.txt
5. Run python3 bindings_pipeline.py --proj_dir external_proj/simdjson --output_dir output_dir.
    a. Pipeline execution order
        i. **Clang_parser.py**
        ii. **extract_class.py**
        iii. Input prompt generation from **Opensource_codellm.py**
        iv. Model function **from Opensource_codellm.py**
            1. **Replace the together.ai API key in** together_api(input_promt) function

Project Pipeline



Project Execution flow



Parsed_nyxus.txt : meta data Ex: Class: TrivialHistogram, Path: nyxus/src/nyx/features/histogram.h, Start Line: 17, End Line: 419

GenAi_ChildFeatureAggregation_binding.cpp : LLM generated binding file

nyxus_ChildFeatureAggregation_class.cpp : Extracted class file

GenAi_bindings_cmake.txt : CMakeLists.txt content

Generated Sample outputs:

```
class PatternObject; // forward declaration
class FILEPATTERN_EXPORT FilePattern {

    public:

        FilePattern(const std::string& path, const std::string& filePattern="", const std::string
        ~FilePattern();
        std::vector<Tuple> getMatchingByMap (Map& variables);
        std::vector<Tuple> getFiles();
        std::vector<Tuple> getMatching(const std::vector<std::tuple<std::string, std::vector<Type
        std::map<std::string, std::map<Types, int>> getOccurrences(const std::vector<std::tuple<s
        std::map<std::string, std::map<Types, int>> getOccurrencesByMap(Map& mapping);
        std::map<std::string, std::set<Types>> getUniqueValues(const std::vector<std::string>& ma
        std::string outputName(std::vector<Tuple>& vec);
        std::vector<std::string> getVariables();
        std::vector<std::pair<std::vector<std::pair<std::string, Types>> , std::vector<Tuple>>> g
        void group(std::vector<std::string>& groups);
        void setGroup(std::string& groups);
        void setGroup(const std::vector<std::string>& groups);
        void next();
        void nextGroup();
        int currentBlockLength();
        void getNewNaming(std::string& pattern, bool suppressWarnings);
        std::vector<Tuple> getSlice(std::vector<Types>& key);
        std::vector<Tuple> getSliceByIdx(int key);
        static std::string inferPattern(const std::string& path, std::string& variables, const st
        static std::string inferPattern(std::vector<std::string>& vec, std::string& variables);
        std::vector<Tuple> getMatchingBlock();
        Tuple getItem(int key);
```

```
#include <pybind11/pybind11.h>
#include <pybind11/stl.h>
#include "FilePattern.h"

namespace py = pybind11;

PYBIND11_MODULE(FilePatternModule, m) {
    py::class_<FilePattern>(m, "FilePattern")
        .def(py::init<const std::string &, const std::string &, const std::string &, bool, bool>())
        .def("getMatchingByMap", &FilePattern::getMatchingByMap)
        .def("getFiles", &FilePattern::getFiles)
        .def("getMatching", &FilePattern::getMatching)
        .def("getOccurrences", &FilePattern::getOccurrences)
        .def("getOccurrencesByMap", &FilePattern::getOccurrencesByMap)
        .def("getUniqueValues", &FilePattern::getUniqueValues)
        .def("outputName", &FilePattern::outputName)
        .def("getVariables", &FilePattern::getVariables)
        .def("groupBy", &FilePattern::groupBy)
        .def("group", &FilePattern::group)
        .def("setGroup", &FilePattern::setGroup)
        .def("setGroup", &FilePattern::setGroup)
        .def("next", &FilePattern::next)
        .def("nextGroup", &FilePattern::nextGroup)
        .def("currentBlockLength", &FilePattern::currentBlockLength)
        .def("getNewNaming", &FilePattern::getNewNaming)
        .def("getSlice", &FilePattern::getSlice)
        .def("getSliceByIdx", &FilePattern::getSliceByIdx)
        .def("inferPattern", &FilePattern::inferPattern)
        .def("getMatchingBlock", &FilePattern::getMatchingBlock)
        .def("getItem", &FilePattern::getItem)
        .def("getItemList", &FilePattern::getItemList)
        .def("begin", &FilePattern::begin)
        .def("end", &FilePattern::end)
        .def("getSize", &FilePattern::getSize)
        .def("getGroupedSize", &FilePattern::getGroupedSize)
        .def("length", &FilePattern::length)
        .def("getGroupedSliceByIdx", &FilePattern::getGroupedSliceByIdx)
        .def("getPattern", &FilePattern::getPattern)
        .def("setPattern", &FilePattern::setPattern)
        .def("getPath", &FilePattern::getPath)
        .def("getPatternObject", &FilePattern::getPatternObject, py::return_value_policy::reference);
}
```

```cmake
cmake_minimum_required(VERSION 3.14)
project(YourProjectName)

set(CMAKE_CXX_STANDARD 17)

# Set the policy for CMP0148
cmake_policy(SET CMP0148 OLD)

# Find Python and pybind11
find_package(PythonInterp REQUIRED)
find_package(PythonLibs REQUIRED)
find_package(pybind11 REQUIRED)

# Find your project (replace 'YourProjectName' with your actual project name)
find_package(YourProjectName QUIET)

# Create the Python module
pybind11_add_module(backend
    src/path/to/your/bindings_code.cpp
)

# Link your project to the Python module
target_link_libraries(backend PRIVATE YourProjectName::YourProjectName)
```

Data Preparation:

GitHub Repository Link: https://github.com/Sandeepyedla0/Bindings-Data-Collection

In the context of future fine-tuning of the model, a GitHub crawler script has been developed for collecting data. The script utilizes a GraphQL query to make an API request to the GitHub API, retrieving repositories that utilize the Pybind11 module.

To keep track of metadata, a CSV file is maintained. The data collection process has been successful, starting with a total count of 23,676 repositories and 4,583 forks.

These scripts are designed to optimize execution time by avoiding the cloning of already visited repositories. This is achieved through the use of a `Visited_Repositories.txt` file, enhancing the efficiency of the script.

To generate the dataset:

Steps to Setup:

1. Run - bash collect_data.sh
2. Collect_bash.sh calls gh_crawler.py - Replace your Github Access token
3. Meta_data.csv : meta_data of repository
4. Test_output - Will have the cloned repository repository information

5. Nonpybind_visited_repositories.txt - Already checked repos which does not have Pybind11_Module() - Saves unnecessary cloning of non-relevant projects

repo_owner_name	repo_name	repository_url	star_count	fork_count	pushedat	binding_file_path
google	mediapipe	https://github.com/google/mediapipe	23676	4583	2023-10-23T15:0	test_output/medi
Tencent	MMKV	https://github.com/Tencent/MMKV	16254	1761	2023-10-10T08:2	test_output/MMK
davisking	dlib	https://github.com/davisking/dlib	12323	2802	2023-10-11T23:3	test_output/dlib/t
microsoft	onnxruntime	https://github.com/microsoft/onnxruntime	10707	2218	2023-10-23T14:0	test_output/onnx
openai	triton	https://github.com/openai/triton	8512	883	2023-10-23T14:5	test_output/triton
google	re2	https://github.com/google/re2	8248	1057	2023-10-20T14:3	test_output/re2/p
finos	perspective	https://github.com/finos/perspective	6827	835	2023-10-21T03:5	test_output/persp
NVIDIA	DALI	https://github.com/NVIDIA/DALI	4635	571	2023-10-23T14:2	test_output/DALI/
spotify	pedalboard	https://github.com/spotify/pedalboard	4501	204	2023-10-17T14:5	test_output/peda