ĐẠI HỌC QUỐC GIA TPHCM

TRƯỜNG ĐẠI HỌC KHOA HỌC TỰ NHIÊN

KHOA CÔNG NGHỆ THÔNG TIN

Project proposal + Proof of concept

Đề tài: Tìm hiểu Ollama và thử nghiệm tích hợp LLM bằng C++

Môn học: Phương pháp lập trình hướng đối tượng

Sinh viên thực hiện:

Nguyễn Quốc Nam - 24120098

Giáo viên hướng dẫn:

ThS. Trần Duy Quang

Ngày 5 tháng 12 năm 2025

Mục lục

Danh sách hình vẽ

1 Giới thiệu

1.1 Lý do chọn đề tài

Trong bối cảnh các mô hình ngôn ngữ lớn (large language models - LLM) đang trở nên phổ biến, nhu cầu chạy các mô hình đó trên máy cá nhân ngày càng tăng. Tuy nhiên, đa số LLM yêu cầu kiến thức triển khai phức tạp. Ollama là nền tảng giúp chạy LLM cục bộ đơn giản, trực quan, nhưng hiện chưa có SDK chính thức dành cho C++. Vì vậy, đề tài hướng đến việc tìm hiểu hoạt động của Ollama và xây dựng PoC (Proof of Concept) cho phép ứng dụng C++ tương tác với mô hình LLM.

1.2 Mục tiêu

- Hiểu cách hoạt động của nền tảng Ollama.
- Cài đặt, cấu hình mô hình LLM chạy trên máy.
- Tích hợp Ollama vào chương trình C++ bằng các phương pháp cơ bản.
- Xây dựng PoC chứng minh khả năng gọi mô hình LLM từ C++.

2 Cơ sở lý thuyết

2.1 Ollama là gì

Ollama là viết tắt của (Omni-Layer Learning Language Acquisition Model: Mô hình Thu nhận Ngôn ngữ Học tập Đa lớp), một phương pháp tiếp cận mới về machine learning. Ollama là một nền tảng đột phá, bình đẳng hóa việc tiếp cận các mô hình ngôn ngữ lớn (LLM) bằng cách cho phép người dùng chạy chúng cục trên máy của họ. [**?**]

2.2 LLM là gì

Mô hình ngôn ngữ lớn (LLM) là một loại chương trình trí tuệ nhân tạo (AI) có khả năng nhận dạng và tạo văn bản, cùng nhiều tác vụ khác. LLM được đào tạo trên các lượng dữ liệu văn bản lớn. LLM được xây dựng dựa trên machine learning: cụ thể là một loại mạng neuron được gọi là mô hình biến đổi. [**?**]

2.3 Khả năng tích hợp vào C++

C++ không có thư viện chính thức để kết nối với Ollama, nhưng có thể tích hợp theo hai hướng:

- Gọi lệnh bằng CLI (ollama run).
- Gửi HTTP request đến REST API của Ollama.

3 Phương pháp thực hiện

3.1 Cài đặt

3.1.1 Ollama

Có thể cài đặt Ollama bằng:

- Website của Ollama
- Github của Ollama
- Tải thủ công câu lệnh:

```
curl -fsSL https://ollama.com/download/ollama-linux-amd64.tgz \
| sudo tar zx -C /usr
```

3.1.2 Các thư viện khác

Bên cạnh Ollama, trình tích hợp mô hình vào C++ cần thêm một số thư viện hỗ trợ:

- nlohmann/json (json.hpp): Thư viện header-only giúp phân tích và tạo dữ liệu JSON trong C++. JSON là định dạng trao đổi dữ liệu chính giữa C++ và Ollama thông qua API.

```
curl -L https://raw.githubusercontent.com/nlohmann/json/develop/
single_include/nlohmann/json.hpp -o path/to/json.hpp
```

- yhirose/cpp-httplib (httplib.h): Thư viện HTTP/HTTPS header-only đa nền tảng cho C++11.

```
curl -L https://raw.githubusercontent.com/yhirose/cpp-httplib/master/
httplib.h -o path/to/httplib.h
```

3.1.3 Các mô hình trong Ollama

Các mô hình ngôn ngữ lớn (LLM) trong Ollama được đóng gói dưới dạng tệp GGUF và tối ưu hóa cho việc chạy trực tiếp trên máy cục bộ. Mỗi mô hình được phân phối với nhiều biến thể dung lượng khác nhau có thể được tìm thấy tại Ollama Library
Người dùng cần lưu ý cấu hình máy:

- 8Gb RAM: có thể chạy được mô với 7 tỉ tham số.
- 16Gb RAM: có thể chạy được mô hình với 13 tỉ tham số.
- 32Gb RAM: có thể chạy được mô hình với 33 tỉ tham số.

Model	Parameters	Size	Download
Gemma 3	1B	815MB	`ollama run gemma3:1b`
Gemma 3	4B	3.3GB	`ollama run gemma3`
Gemma 3	12B	8.1GB	`ollama run gemma3:12b`
Gemma 3	27B	17GB	`ollama run gemma3:27b`
QwQ	32B	20GB	`ollama run qwq`
DeepSeek-R1	7B	4.7GB	`ollama run deepseek-r1`
DeepSeek-R1	671B	404GB	`ollama run deepseek-r1:671b`
Llama 4	109B	67GB	`ollama run llama4:scout`
Llama 4	400B	245GB	`ollama run llama4:maverick`
Llama 3.3	70B	43GB	`ollama run llama3.3`
Llama 3.2	3B	2.0GB	`ollama run llama3.2`
Llama 3.2	1B	1.3GB	`ollama run llama3.2:1b`
Llama 3.2 Vision	11B	7.9GB	`ollama run llama3.2-vision`
Llama 3.2 Vision	90B	55GB	`ollama run llama3.2-vision:90b`
Llama 3.1	8B	4.7GB	`ollama run llama3.1`
Llama 3.1	405B	231GB	`ollama run llama3.1:405b`
Phi 4	14B	9.1GB	`ollama run phi4`
Phi 4 Mini	3.8B	2.5GB	`ollama run phi4-mini`
Mistral	7B	4.1GB	`ollama run mistral`

Hình 1: Một số model với số tham số, kích thước, cmd để tải

3.2 Tìm hiểu API và cách gọi mô hình [?]

Ollama cung cấp REST API hoạt động qua port mặc định 11434, giúp các ngôn ngữ lập trình (C++, Python, Go...) gọi mô hình dễ dàng thông qua HTTP request.

3.2.1 Kiến trúc API

- Server chạy local, không yêu cầu internet.
- Mặc định giao tiếp qua http://localhost:11434/api/...
- Dữ liệu trao đổi dạng JSON.

3.2.2 Endpoint quan trọng

- /api/generate – sinh văn bản từ prompt Yêu cầu tối thiểu:

```
{
  "model": "llama3.1",
  "prompt": "Hello Ollama!"
}
```

  Phản hồi (rút gọn)

```
{
  "response": "Hello! How can I help?",
  "done": true
}
```

  Tham số quan trọng:

  - model: tên mô hình đã pull.
  - prompt: nội dung đầu vào.
  - stream: true hoặc false (stream token theo thời gian thực).

- /api/chat – kiểu chatbot (dạng tin nhắn)
- /api/tags – xem danh sách model cài trên máy
- /api/pull – tải mô hình mới

4 Proof of Concept (PoC)

4.1 Mô tả

Mục tiêu của PoC là xác minh rằng:

- Ứng dụng C++ có thể gửi yêu cầu sinh văn bản tới mô hình LLM đã được cài đặt trong Ollama.
- Kết quả trả về có thể được đọc, xử lý và sử dụng tiếp trong chương trình.
- Cả hai phương thức giao tiếp (CLI và REST API) đều hoạt động ổn định, có thể mở rộng cho các ứng lớn hơn.

Trước khi chạy chương trình phải kiểm tra xem ollama có đang chạy bằng lệnh:

```
ollama serve
```

Kiểm tra các mô hình đã tải bằng lệnh:

```
ollama list
```

4.2 Gọi Ollama CLI

Phương pháp này dùng command-line của Ollama:

```
ollama run llama3.1 "Hello"
```

Trong C++, có thể dùng lệnh std::system trong <cstdlib>

```
void cli::autoAsk(const string& model) {
    string prompt;
    do {
        cout << RED_COLOR << "You: " << BLUE_COLOR;
        getline(cin, prompt);

        if (prompt == "exit") break;
        cout << GREEN_COLOR << "Ollama: \n" << BLUE_COLOR << flush;
        string cmd = "ollama run " + model + " \"" + cli::escapeQuotes(prompt) +
    "\"";
        system(cmd.c_str());
    }
    while (true);
}
```

4.3 Gọi Ollama REST API

Phương pháp này giao tiếp bằng REST API:

```
generate(const string& model, const string& prompt) {
    httplib::Client cli("localhost", 11434);

    json body = {
        {"model", model},
        {"prompt", prompt},
        {"stream", false}
    };

    auto res = cli.Post("/api/generate", body.dump(), "application/json");

    Response out;
    if (!res) {
        out.response = "[Error] HTTP request failed. Is Ollama running?";
        return out;
    }
    if (res->status != 200) {
        out.response = "[Error] HTTP " + to_string(res->status) + ": " + res->
   body;
        return out;
    }

    json resp_json = json::parse(res->body);
    out.response   = resp_json.value("response", "");
    return out;
}
```

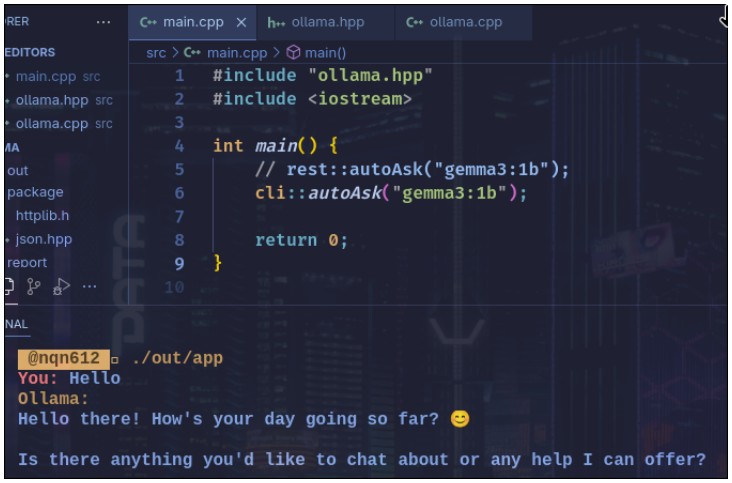
5 Kết luận

5.1 Kết quả đạt được

Cả hai phương pháp đều đáp ứng tốt yêu cầu thử nghiệm mô hình LLM trên C++ như:

- Gửi câu hỏi từ C++ đến mô hình LLM và nhận được câu trả lời chính xác.

- Kích hoạt mô hình cục bộ mà không phụ thuộc vào kết nối Internet.
- Tạo được module autoAsk cho phép người dùng nhập câu hỏi liên tục theo dạng hội thoại đơn giản.



Hình 2: Test module autoAsk()

5.2 Hướng phát triển

Chương trình có thể được mở rộng theo nhiều hướng nhằm nâng cao tính ứng dụng và độ hoàn thiện. Một số hướng phát triển khả thi bao gồm:

- Tích hợp REST API hoàn chỉnh
- Giao diện người dùng (GUI)
- Triển khai vào ứng dụng thực tế

6 Tài liệu tham khảo