HỌC VIỆN CÔNG NGHỆ BƯU CHÍNH VIỄN THÔNG



Nguyễn Thị Quỳnh Anh

ĐỀ ÁN TỐT NGHIỆP THẠC SĨ KỸ THUẬT

(Theo định hướng ứng dụng)

HÀ NỘI - 2025

HỌC VIỆN CÔNG NGHỆ BƯU CHÍNH VIỄN THÔNG

Nguyễn Thị Quỳnh Anh

XÂY DỰNG QUY TRÌNH SINH MÃ KIỂM THỬ CHỨC NĂNG TỰ ĐỘNG SỬ DỤNG LLM VÀ ÁP DỤNG CHO HỆ THỐNG VNEDU

CHUYÊN NGÀNH : HỆ THỐNG THÔNG TIN
MÃ SỐ: 8.48.01.04 (Hệ thống thông tin)

ĐỀ ÁN TỐT NGHIỆP THẠC SĨ KỸ THUẬT

(Theo định hướng ứng dụng)

NGƯỜI HƯỚNG DẪN KHOA HỌC: TS. ĐỖ THỊ BÍCH NGỌC

HÀ NỘI - 2025

MỤC LỤC

MỤC LỤC **3**
LỜI CAM ĐOAN **5**
LỜI CẢM ƠN **6**
DANH MỤC CÁC KÝ HIỆU, CÁC CHỮ VIẾT TẮT **7**
DANH MỤC CÁC BẢNG **8**
DANH MỤC CÁC HÌNH **9**
I. MỞ ĐẦU **1**
II. NỘI DUNG **6**
CHƯƠNG I: KIỂM THỬ CHỨC NĂNG TỰ ĐỘNG VÀ HIỆN TRẠNG CỦA HỆ THỐNG VNEDU **6**
1.1. Kiểm thử chức năng tự động 6
1.2. Giới thiệu về hệ thống VNEDU 8
1.3. Hiện trạng hoạt động kiểm thử tự động của hệ thống VNEDU 9
KẾT LUẬN CHƯƠNG **12**
CHƯƠNG II: LÝ THUYẾT LLM VÀ SỬ DỤNG CÔNG CỤ LLM TRONG KIỂM THỬ CHỨC NĂNG TỰ ĐỘNG **13**
2.1. Giới thiệu về LLM và kỹ thuật Prompt Engineering 13
2.1.1. Large Language Models (LLMs): 13
2.1.2. Prompt Engineering: 14
2.2. Sử dụng công cụ LLM trong sinh mã kiểm thử chức năng tự động 15
2.3. Công nghệ và công cụ sử dụng 18
KẾT LUẬN CHƯƠNG **19**
CHƯƠNG III: XÂY DỰNG QUY TRÌNH SINH MÃ KIỂM THỬ CHỨC NĂNG TỰ ĐỘNG SỬ DỤNG LLM **20**
3.1. Tổng quan các thành phần cần thiết để thực hiện sinh mã chức năng tự động 20
3.2. Thu thập đầu vào dựa trên đánh giá thực trạng 21
3.3. Phân loại đầu vào 22

3.3.1. Nguyên tắc phân loại đầu vào 22
3.4. Xây dựng mẫu prompt 24
3.6. Sinh mã kiểm thử chức năng tự động 27
3.7. Phân tích, chỉnh sửa mã kiểm thử chức năng tự động 29
3.8. Lưu trữ và quản lý mã 30
3.9. Quy trình sinh mã kiểm thử chức năng tự động 31
KẾT LUẬN CHƯƠNG 34
CHƯƠNG IV: THỰC NGHIỆM VÀ ĐÁNH GIÁ TRÊN HỆ THỐNG VNEDU 35
4.1. Thực nghiệm quy trình trên hệ thống VNEDU: 35
4.1.1. Thu thập đầu vào: 35
4.1.2. Phân loại đầu vào: 38
4.1.3. Xây dựng prompt mẫu: 39
4.1.4. Tiền xử lý và liên kết dữ liệu hỗ trợ: 45
4.1.5. Sinh mã kiểm thử chức năng tự động: 46
4.1.6. Phân tích và chỉnh sửa mã: 50
4.1.7. Lưu trữ và quản lý mã: 50
4.2. Phân tích kết quả thực nghiệm: 52
KẾT LUẬN CHƯƠNG 64
KẾT LUẬN VÀ HƯỚNG PHÁT TRIỂN 65
DANH MỤC TÀI LIỆU THAM KHẢO 66

LỜI CAM ĐOAN

Em xin cam đoan đề án là thành quả công trình nghiên cứu độc lập với sự giúp đỡ của giáo viên hướng dẫn. Kết quả nghiên cứu chưa từng được công bố trong bất kỳ công trình nào khác. Tất cả sự giúp đỡ cho việc thực hiện đề án đã được cảm ơn và các thông tin trích dẫn đều có nguồn gốc rõ ràng.

TÁC GIẢ ĐỀ ÁN

Nguyễn Thị Quỳnh Anh

LỜI CẢM ƠN

Trước hết, em xin gửi lời cảm ơn chân thành tới Học viện Công nghệ Bưu chính Viễn thông, Khoa Đào tạo Sau đại học và toàn thể các thầy cô giáo của Học viện đã tạo điều kiện thuận lợi, truyền đạt kiến thức và hỗ trợ em trong suốt quá trình học tập và nghiên cứu chương trình Thạc sĩ.

Em xin bày tỏ lòng biết ơn sâu sắc tới Tiến sĩ Đỗ Thị Bích Ngọc, người đã tận tâm hướng dẫn, dành nhiều thời gian, công sức và đưa ra những định hướng quý báu, giúp em hoàn thiện đề án này một cách tốt nhất.

Em cũng xin chân thành cảm ơn Ban lãnh đạo cùng các anh chị đồng nghiệp tại Trung tâm giải pháp Giáo dục số - Công ty Công nghệ Thông tin VNPT đã luôn hỗ trợ, chia sẻ kinh nghiệm và đóng góp nhiều ý kiến thiết thực trong quá trình em khảo sát và thực nghiệm nghiên cứu.

Cuối cùng, em xin gửi lời tri ân tới gia đình, người thân và bạn bè, những người đã luôn ở bên động viên, khích lệ và tạo chỗ dựa tinh thần vững chắc để em hoàn thành quá trình học tập và hoàn thiện đề án này.

Em xin trân trọng cảm ơn.

Hà Nội, ngày 15 tháng 12 năm 2025
Người viết

Nguyễn Thị Quỳnh Anh

DANH MỤC CÁC KÝ HIỆU, CÁC CHỮ VIẾT TẮT

Từ viết tắt	Giải thích	Tên tiếng anh
VNPT IT	Công ty Công Nghệ thông tin VNPT	VNPT IT
LLM	Mô hình ngôn ngữ lớn	Large language model
HTML	Ngôn ngữ đánh dấu được thiết kế ra để tạo nên các trang web	Hyper Text Markup Language
QC/Tester	Kiểm thử viên	Quality Controler/Tester
XML	Ngôn ngữ đánh dấu mở rộng được thiết kế để lưu trữ và truyền tải dữ liệu một cách có cấu trúc	Extensible Markup Language
Xpath	Một ngôn ngữ truy vấn được thiết kế để điều hướng và truy xuất các thành phần trong tài liệu XML	XML Path Language
DOM	Mô hình Các Đối tượng Tài liệu	Document Object Model

DANH MỤC CÁC BẢNG

Bảng 1.1. Ví dụ Test case Đăng nhập thành công 6
Bảng 2.1. Tổng quan về các công cụ LLM sinh mã kiểm thử chức năng tự động ...17
Bảng 4.1. Số lượng test case thử nghiệm 38
Bảng 4.2. Testcase đại diện theo nhóm chức năng hệ thống VNEDU 39
Bảng 4.3. Ví dụ phân loại Xpath và Keyword test case Thêm nhân sự thành công .39
Bảng 4.4. Prompt mẫu.....40
Bảng 4.5. Sinh mã cho các loại dữ liệu.....46
Bảng 4.6. Các loại thực nghiệm 54

DANH MỤC CÁC HÌNH

Hình 1.1. Ví dụ Test Script Đăng nhập thành công 7
Hình 1.2. Ví dụ gom nhóm keyword Đăng nhập thành công 8
Hình 3.1. Quy trình sinh mã kiểm thử chức năng tự động sử dụng LLM (1).....31
Hình 3.2. Quy trình sinh mã kiểm thử chức năng tự động sử dụng LLM (2).....33
Hình 4.1. Ví dụ tệp HTML đầu vào 38
Hình 4.2. Test case Thêm nhân sự thành công sau xử lý 45
Hình 4.3. Keyword của test case Thêm nhân sự thành công sau xử lý.....46
Hình 4.4. Ví dụ sinh mã cho loại dữ liệu Xpath tĩnh 48
Hình 4.5. Ví dụ sinh Xpath cho test case Thêm nhân sự thành công 48
Hình 4.6. Ví dụ sinh Keyword cho test case Thêm nhân sự thành công 49
Hình 4.7. Ví dụ sinh test case Thêm nhân sự thành công 49
Hình 4.8. Ví dụ phân tích và chỉnh sửa mã 50
Hình 4.9. Xác nhận mã trước khi lưu.....51
Hình 4.10. Xác nhận tệp trước khi lưu.....51
Hình 4.11. Cấu trúc lưu trữ và quản lý đầu vào và đầu ra của mô hình LLM 51
Hình 4.12. Accuracy Xpath Tĩnh 55
Hình 4.13. Test Coverage Xpath tĩnh 55
Hình 4.14. Accuracy Xpath động.....56
Hình 4.15. Test Coverage Xpath động.....56
Hình 4.16. Accuracy Keyword thao tác giao diện 57
Hình 4.17. Test Coverage Keyword thao tác với giao diện.....57
Hình 4.18. Accuracy Keyword kiểm tra giao diện 58
Hình 4.19. Test Coverage Keyword kiểm tra giao diện.....58
Hình 4.20. Accuracy Test case một tính năng 59
Hình 4.21. Accuracy Test case nhiều tính năng.....59
Hình 4.22. Test Coverage Test case một tính năng 59
Hình 4.23. Test Coverage Test case nhiều tính năng.....60

Hình 4.24. Generation time.....61
Hình 4.25. Pass Rate và Execution Time.....62

I. MỞ ĐẦU

1. Tính cấp thiết của đề án.

Trong quá trình phát triển phần mềm, yêu cầu về chất lượng, tốc độ phát triển và khả năng thích ứng với thay đổi của hệ thống phần mềm ngày càng cao. Điều này tạo áp lực lớn cho kiểm thử phần mềm, vốn đóng vai trò then chốt trong việc đảm bảo sản phẩm ổn định, đáng tin cậy và đáp ứng yêu cầu người dùng.

Kiểm thử tự động ra đời để giảm tải việc kiểm thử thủ công và tăng độ chính xác, nhưng các phương pháp kiểm thử tự động thông thường vẫn phụ thuộc nhiều vào việc kiểm thử viên (tester) tự viết mã kiểm thử và dần bộc lộ hạn chế khi quy mô hệ thống lớn và thay đổi liên tục. Việc xây dựng và bảo trì kịch bản kiểm thử tự động đòi hỏi kỹ năng lập trình, tốn nhiều thời gian và nhân lực, đặc biệt trong các mô hình phát triển phần mềm nhanh như Agile/Scrum.

Sự xuất hiện của các mô hình ngôn ngữ lớn LLM (Large Language Model) như GPT, Claude hay Gemini đã mở ra cơ hội ứng dụng LLM sinh mã kiểm thử chức năng tự động từ mô tả ý định kiểm thử (test intent description) [1]. Tuy nhiên việc chỉ áp dụng mô hình LLM sinh mã kiểm thử chức năng tự động mà không có một quy trình phù hợp để quản lý đầu vào và kiểm soát đầu ra có thể dẫn đến nhiều rủi ro:

- Thiếu ngữ cảnh hoặc hiểu sai ngữ cảnh kiểm thử, sinh mã kiểm thử chức năng tự động dựa trên ngữ cảnh hiện tại không hiểu được ngữ cảnh tổng thể, quy trình nghiệp vụ, điều kiện tiền đề dẫn đến dễ sinh mã kiểm thử chức năng sai thứ tự, nhảy bước
- Không có tiêu chí để đánh giá mã kiểm thử chức năng sinh ra: nếu không có chuẩn đánh giá cụ thể, kiểm thử viên (tester) không thể xác định mã có đang đúng yêu cầu không, có bỏ sót bước nào không
- Không kiểm soát được sự phụ thuộc giữa các bước kiểm thử: nhiều mã kiểm thử chức năng thực tế cần có các bước điều kiện trước, nếu không có quy trình hợp lý thì không nhận biết được các phụ thuộc này

Trung tâm giải pháp Giáo dục số của VNPT-IT, đơn vị phát triển hệ thống VNEDU, hiện cũng gặp nhiều thách thức trong việc mở rộng hoạt động kiểm thử tự

động. Việc viết mã kiểm thử cho hệ thống VNEDU tốn nhiều công sức và khó theo kịp tốc độ phát triển liên tục của sản phẩm. Hệ thống VNEDU là nền tảng quy mô lớn phục vụ học sinh, giáo viên, phụ huynh, các cơ sở giáo dục và Sở Giáo dục trên phạm vi toàn quốc, do đó mọi thay đổi từ phía Bộ Giáo dục và Đào tạo đều tác động trực tiếp đến toàn bộ hệ thống. Theo Thông tư 22/2021/TT-BGDĐT, yêu cầu thay đổi hoàn toàn cách ghi nhận kết quả học tập và kết quả rèn luyện: chuyển thuật ngữ từ 'học lực - hạnh kiểm' sang 'kết quả học tập - kết quả rèn luyện', đồng thời chuyển toàn bộ mức xếp loại kết quả học tập, kết quả rèn luyện từ năm mức Giỏi, Khá, Trung bình, Yếu, Kém sang bốn mức Tốt, Khá, Đạt, Chưa đạt buộc hệ thống phải cập nhật lại phân hệ như Nhập điểm, Tính tổng kết, Thống kê báo cáo,... Theo Thông tư 12/2025/TT-BGDĐT yêu cầu thay thế các cụm từ "Ủy ban nhân dân cấp huyện", "Phòng Giáo dục và Đào tạo" bằng "Uỷ ban nhân dân cấp xã", đồng thời chuyển đổi mô hình quản lý từ 64 tỉnh thành sang 34 tỉnh thành, ảnh hưởng trực tiếp đến phân quyền, dữ liệu và luồng nghiệp vụ tại cơ sở quản lý giáo dục cấp xã. Những thay đổi mang tính cấu trúc như vậy khiến hàng loạt màn hình, nghiệp vụ của hệ thống VNEDU phải chỉnh sửa trong thời gian ngắn, làm ảnh hưởng trung bình 20 đến 30% số mã kiểm thử chức năng của hệ thống VNEDU mỗi lần thay đổi thông tư.

Trước nhu cầu ngày càng tăng của hoạt động kiểm thử tự động trong các hệ thống phần mềm có quy mô lớn và thay đổi liên tục, cùng với việc nhận thấy rõ tiềm năng của các mô hình ngôn ngữ lớn LLM trong việc hỗ trợ sinh mã kiểm thử tự động từ mô tả ý định kiểm thử [1], học viên đã lựa chọn đề án **"XÂY DỰNG QUY TRÌNH SINH MÃ KIỂM THỬ CHỨC NĂNG TỰ ĐỘNG SỬ DỤNG LLM VÀ ÁP DỤNG CHO HỆ THỐNG VNEDU"** hướng tới xây dựng một quy trình sử dụng mô hình LLM sinh mã kiểm thử chức năng từ các bước kiểm thử thủ công và tệp HTML chứa cấu trúc giao diện của hệ thống, hỗ trợ kiểm thử viên không phải tự viết mã kiểm thử, quản lý đầu vào đầu ra của hoạt động kiểm thử tự động, đồng thời hỗ trợ mô hình LLM sinh mã kiểm thử chức năng, kiểm soát chất lượng và lưu trữ kết quả đầu ra sau khi mô hình LLM sinh mã kiểm thử chức năng, qua đó hỗ trợ phát triển hoạt động kiểm thử tự động trong các hệ thống có quy mô lớn như VNEDU .

2. Tổng quan nghiên cứu.

Kiểm thử phần mềm là một lĩnh vực quan trọng nhằm đảm bảo chất lượng trước khi triển khai đến người dùng. Hiện nay ngày càng nhiều các dự án phần mềm yêu cầu kiểm thử tự động song song với kiểm thử thủ công. Các công cụ kiểm thử tự động phổ biến như Selenium, Robot Framework hay Katalon Studio yêu cầu kiểm thử viên tự lập trình và viết các kịch bản kiểm thử dưới dạng mã lệnh, từ việc xác định phần tử giao diện đến xây dựng các bước thao tác và kiểm tra kết quả. Tuy nhiên kiểm thử tự động vẫn tồn tại hạn chế ở khâu viết kịch bản kiểm thử, đòi hỏi tester phải có kỹ năng lập trình và tốn nhiều thời gian viết khi hệ thống thay đổi [6][12]. Trong bối cảnh đó, sự phát triển mạnh mẽ của các mô hình LLM đã mở ra khả năng tự sinh mã kiểm thử tự động từ mô tả ý định kiểm thử [1][7][15]. Các nghiên cứu [10][11][14] cho thấy LLM mở ra cơ hội sinh mã kiểm thử chức năng từ ngôn ngữ tự nhiên, nhưng vẫn tồn tại những thách thức về độ chính xác cần được giải quyết để áp dụng trong thực tế. Để ứng dụng LLM sinh mã kiểm thử chức năng tự động trong thực tế cần có một quy trình kiểm soát được các yếu tố như đầu vào, sử dụng kỹ thuật thiết kế tương tác đầu vào (Prompt Engineering) hợp lý [2][3][8][9], kiểm soát chất lượng đầu ra và tổ chức lại mã kiểm thử sinh ra để dễ tái sử dụng [4][5][13]. Đây là hướng đi cho quy trình mới giúp giảm thiểu việc kiểm thử viên phải tự viết mã kiểm thử bằng cách sử dụng mô hình LLM chuyển đổi các bước kiểm thử thủ công sang kịch bản kiểm thử tự động.

Để đảm bảo quy trình sinh mã kiểm thử chức năng tự động bằng mô hình (LLM) đạt được hiệu quả và tính ứng dụng thực tế, cần xem xét các yếu tố sau đây:

- Thu thập các loại dữ liệu đầu vào cần thiết và lưu trữ tập trung, tạo nền tảng thống nhất để phục vụ cho các bước xử lý và sinh mã kiểm thử chức năng ở các phần tiếp theo.
- Phân loại đầu vào là phần cần thiết để phân loại và gom nhóm các bước kiểm thử được viết bằng ngôn ngữ tự nhiên, tạo cơ sở cho kỹ thuật Prompt Engineering.
- Xây dựng mẫu prompt là phần xây dựng các thiết kế hướng dẫn (prompt)

hỗ trợ mô hình LLM hiểu và sinh mã kiểm thử chức năng phù hợp với từng hành động cụ thể bằng cách sử dụng kỹ thuật Prompt Engineering hợp lý.

- Tiền xử lý và liên kết dữ liệu hỗ trợ giúp mô hình LLM hiểu được ngữ cảnh kiểm thử thông qua việc xử lý đầu vào vốn được viết bằng ngôn ngữ tự nhiên, đánh dấu các thành phần trong đầu vào tương ứng với các phần tử giao diện, đồng thời đánh dấu các biến dữ liệu và gom nhóm các bước kiểm thử thủ công lặp lại.
- Sinh mã kiểm thử chức năng tự động, là trọng tâm của quy trình, trong đó mô hình LLM tạo ra mã kiểm thử dựa trên đầu vào ngữ cảnh là prompt và dữ liệu đầu vào đã được xử lý.
- Để đảm bảo mã sinh ra có thể thực thi hiệu quả, bước phân tích và chỉnh sửa mã kiểm thử chức năng là cần thiết nhằm đánh giá, tinh chỉnh và khắc phục các lỗi nghiệp vụ, cú pháp hoặc chưa phù hợp với hệ thống cụ thể.
- Bước lưu trữ và quản lý mã kiểm thử chức năng đóng vai trò đảm bảo khả năng truy xuất, tái sử dụng và tích hợp vào hoạt động kiểm thử tự động của tổ chức.

Mỗi thành phần trong quy trình đều đóng vai trò hỗ trợ lẫn nhau, tạo thành một quy trình tổng thể nhằm tối ưu hiệu quả của việc sinh mã kiểm thử chức năng tự động sử dụng LLM.

3. Mục tiêu nghiên cứu.

Mục tiêu của đề án này là xây dựng một quy trình sinh mã kiểm thử chức năng tự động sử dụng mô hình LLM kết hợp kỹ thuật Prompt Engineering, nhằm chuyển đổi mô tả các bước kiểm thử thủ công thành mã kiểm thử chức năng tự động. Quy trình này hướng đến việc hỗ trợ cách tương tác giữa người kiểm thử và mô hình ngôn ngữ lớn LLM nhằm nâng cao hiệu quả tạo kịch bản kiểm thử tự động và có khả năng ứng dụng trong thực tiễn kiểm thử phần mềm.

4. Đối tượng và phạm vi nghiên cứu.

- Đối tượng nghiên cứu:

Quy trình sinh mã kiểm thử chức năng tự động sử dụng LLM, trong đó tập trung vào việc khai thác khả năng sinh mã của các mô hình LLM và kỹ thuật Prompt Engineering để hỗ trợ kiểm thử chức năng tự động cho giao diện web

- Phạm vi nghiên cứu: Tập hợp các bước kiểm thử chức năng được mô tả bằng ngôn ngữ tự nhiên
- Phương pháp nghiên cứu:
  - Phương pháp nghiên cứu lý thuyết: Thu thập và phân tích các tài liệu, công trình nghiên cứu, sách chuyên khảo và bài báo khoa học liên quan đến kiểm thử tự động, mô hình LLM, và kỹ thuật Prompt Engineering nhằm xây dựng cơ sở lý thuyết cho việc xây dựng quy trình sinh mã kiểm thử chức năng tự động
  - Phương pháp thực nghiệm: Thử nghiệm quy trình sinh mã kiểm thử chức năng tự động sử dụng mô hình LLM trên tập các bước mô tả kiểm thử thủ công, đánh giá kết quả dựa trên các chỉ số như mức độ chính xác của mã sinh ra, khả năng thực thi
  - Phương pháp phân tích định tính: Phân tích khả năng áp dụng của quy trình trong môi trường thực tế, từ đó đánh giá hiệu quả của quy trình sinh mã kiểm thử tự động trong bối cảnh các hệ thống phần mềm có giao diện web

II. NỘI DUNG

CHƯƠNG I: KIỂM THỬ CHỨC NĂNG TỰ ĐỘNG VÀ HIỆN TRẠNG CỦA HỆ THỐNG VNEDU

Chương I trình bày lý thuyết về kiểm thử chức năng tự động, giới thiệu hệ thống VNEDU và hiện trạng hoạt động kiểm thử chức năng tự động của hệ thống VNEDU.

1.1. Kiểm thử chức năng tự động

Kiểm thử chức năng (Functional Testing) là phương pháp kiểm thử phần mềm nhằm đảm bảo các tính năng của hệ thống hoạt động đúng như mô tả trong tài liệu yêu cầu hoặc đặc tả nghiệp vụ.

Kiểm thử tự động (Automated Testing) là phương pháp kiểm thử phần mềm trong đó kiểm thử viên đọc các trường hợp kiểm thử (test case) sau đó dựa trên test case và giao diện của chức năng để viết các kịch bản kiểm thử tự động (script) dưới dạng mã, các mã này được thực thi tự động thông qua các công cụ kiểm thử tự động thay vì thực hiện thủ công bởi con người. Các công cụ kiểm thử tự động thực hiện các thao tác trên giao diện theo kịch bản kiểm thử tự động được lập trình sẵn và ghi nhận kết quả kiểm thử tự động.

Test case là một trường hợp cần kiểm thử, nó bao gồm các bước kiểm thử là các thao tác với giao diện, tiền điều kiện, các giá trị đầu vào, và kết quả mong đợi. Mỗi một thành phần trong test case được mô tả trong một cột thể hiện dưới dạng bảng nhiều cột ví dụ test case Đăng nhập thành công:

Bảng 1.1. Ví dụ Test case Đăng nhập thành công

Tên test case	Tiền điều kiện	Các bước kiểm thử	Dữ liệu đầu vào	Kết quả mong đợi
Đăng nhập thành công	1. Mở trình duyệt vào trang xxx	1. Nhập tài khoản 2. Nhập mật khẩu 3. Click nút Đăng nhập	1. Trang: xxx 2. Giá trị tài khoản: username 2. Giá trị mật khẩu: password	Đăng nhập thành công, không hiển thị nút Đăng nhập trên giao diện

DOM (Document Object Model) là mô hình biểu diễn cấu trúc của một tài liệu HTML hoặc XML dưới dạng cây phân cấp (tree structure). Mỗi thành phần trên trang web như thẻ div, button, input,... được biểu diễn dưới dạng một nút (node) trong cây DOM

Xpath là một cú pháp được sử dụng để tìm vị trí của bất kỳ phần tử nào trên trang web trong cấu trúc DOM dựa trên thuộc tính, nhãn (label) của phần tử,... Ví dụ Xpath trỏ đến tất cả thẻ input có thuộc tính id là txtUsername trong DOM: //input[@id='txtUsername']

Kịch bản kiểm thử (test script) là một tập hợp các mã lệnh được kiểm thử viên viết bằng công cụ kiểm thử tự động để tự động hóa quá trình thực thi một trường hợp kiểm thử (test case). Một test script được tạo ra từ các bước được mô tả trong test case, bao gồm tiền điều kiện, dữ liệu đầu vào, các bước thao tác và kết quả mong đợi. Trong test script các bước thực hiện trong test case sẽ được ánh xạ thành các keyword, dữ liệu đầu vào của test case được truyền vào keyword dưới dạng tham số, các keyword kết hợp với các XPath để xác định phần tử giao diện cần tương tác trên web:

- XPath: dùng để xác định và truy cập các phần tử giao diện trên web (button, textbox, dropdown,...).
- Keyword: dùng để thực hiện các thao tác trên hệ thống như click, nhập hoặc kiểm tra giá trị,... Các công cụ kiểm thử tự động phổ biến hỗ trợ hai dạng keyword: keyword có sẵn (Built-in Keyword) ví dụ như (Open Browser, Click Element,...), Keyword do người dùng tự định nghĩa (Custom Keyword) là các keyword kiểm thử viên tự tạo ra, thường được xây dựng từ nhiều keyword nhỏ hơn
- Test data: là giá trị cần kiểm tra của test case (ví dụ tài khoản, mật khẩu) được truyền vào keyword dưới dạng tham số, mỗi test case có test data khác nhau

```
Dang_nhap_thanh_cong
    [Arguments]    ${site}
    Open Browser    ${site}    ${browser}
    Input Text    //div[normalize-space(.)='Tên tài khoản']/following::input[1]    ${tai_khoan}
    Input Text    //div[normalize-space(.)='Mật khẩu']/following::input[1]    ${mat_khau}
    Click Element    //button[normalize-space(.)='Đăng nhập']
    Element Should Not Be Visible    //button[normalize-space(.)='Đăng nhập']
```

Hình 1.1. Ví dụ Test Script Đăng nhập thành công

Keyword-driven testing: là phương pháp được sử dụng phổ biến trong kiểm

thử giao diện web mục đích tạo các test script từ tập từ khóa (Keyword), các thao tác được kiểm thử viên tự đóng gói thành các keyword có thể tái sử dụng dựa trên việc gom nhóm các hành động kiểm thử dùng chung ở nhiều test case (Custom Keyword).

```
Run | Debug | Run in Interactive Console
TC50_XOA_THU_MUC_HSS_THANH_CONG
    [Documentation]    Steps: Đăng nhập -> Vào chức năng -> Thêm thư mục -> Xóa thư mục
    Dang_nhap_thanh_cong    ${SITE}    ${BROWSER}    ${USERNAME}    ${PASSWORD}
    Sleep    2s

    go_chuc_nang    ${TEN_CHUC_NANG}
    Sleep    2s
```

Hình 1.2. Ví dụ gom nhóm keyword Đăng nhập thành công

Các framework kiểm thử tự động phổ biến gồm Selenium, Robot Framework,… hỗ trợ định vị phần tử giao diện thông qua nhiều cơ chế như XPath, CSS Selector,... đảm bảo có thể truy cập chính xác đối tượng cần thao tác, cung cấp tập hợp các keyword có sẵn như click, nhập,... cho phép kiểm thử viên tự viết, cung cấp báo cáo kết quả, ghi log chi tiết nhằm phục vụ kiểm thử hồi quy.

1.2. Giới thiệu về hệ thống VNEDU

Để đảm bảo tính thực tiễn của đề tài, đề án cần lựa chọn một hệ thống đủ lớn và có độ phức tạp nghiệp vụ phù hợp làm môi trường đánh giá quy trình. Hệ thống VNEDU đáp ứng đầy đủ các tiêu chí này nên việc giới thiệu tổng quan về hệ thống VNEDU là cần thiết trước khi đi vào xây dựng quy trình và thực nghiệm. VNEDU là hệ thống quản lý giáo dục số được xây dựng và phát triển bởi Trung tâm giải pháp Giáo dục số – VNPT IT, mục tiêu số hóa các hoạt động quản lý dạy học trong nhà trường. Hệ thống phục vụ hơn 20 triệu người dùng, bao gồm học sinh, giáo viên, phụ huynh, cán bộ quản lý, ban giám hiệu và các cơ quan quản lý giáo dục trên 34 tỉnh thành với hơn 3 triệu hồ sơ học sinh, 9000 trường học, bao gồm hơn 30 phân hệ nghiệp vụ, hàng trăm tính năng nghiệp vụ và hàng trăm màn hình giao diện. Một số phân hệ tính năng chính ví dụ như:

- Quản lý học sinh: lưu trữ và cập nhật hồ sơ học sinh, theo dõi quá trình học tập, điều chuyển và tiếp nhận học sinh theo năm học
- Quản lý giáo viên và chủ nhiệm: quản lý thông tin giáo viên, phân công chủ nhiệm, phân công giảng dạy

- Sổ học bạ: tổng hợp kết quả học tập, kết quả rèn luyện và thông tin học sinh theo quy định hiện hành, kết xuất các biểu mẫu dùng trong nhà trường
- Quản lý lớp học: danh sách lớp, sĩ số học sinh, cấu hình môn học theo lớp
- Tổng kết và xét lên lớp: tính điểm tổng kết, danh hiệu thi đua, lên lớp, lưu ban theo năm học

Với quy mô lớn, sự đa dạng về nghiệp vụ và yêu cầu cập nhật thường xuyên theo thay đổi của chính sách giáo dục và chuyển đổi mô hình chính quyền, hệ thống VNEDU đòi hỏi việc kiểm thử chức năng tự động phải được thực hiện chính xác và thường xuyên.

1.3. Hiện trạng hoạt động kiểm thử tự động của hệ thống VNEDU

Hệ thống VNEDU hiện đang vận hành với quy mô nghiệp vụ lớn, bao gồm hơn 300 chức năng thuộc nhiều phân hệ tính năng như Quản lý lớp học, Quản lý học sinh, Quản lý nhân sự, Nhập điểm, Tổng kết,... mỗi tính năng lại bao gồm nhiều màn hình giao diện và luồng xử lý nghiệp vụ phức tạp. Hiện hệ thống VNEDU duy trì trên 200 test case nghiệp vụ chính trong đó 150 test case được tự động hóa tương ứng với 150 kịch bản kiểm thử tự động. Hệ thống VNEDU được phát triển theo mô hình Agile với chu kỳ triển khai liên tục, mỗi sprint kéo dài 2 tuần trung bình khoảng 100 yêu cầu. Mỗi lần cập nhật hệ thống đều đòi hỏi các kiểm thử viên vừa phải kiểm thử chức năng và kiểm thử tự động. Tuy hệ thống lớn và có nhiều yêu cầu nhưng hiện nay đội ngũ kiểm thử viên chỉ có 5 người đồng thời thực hiện các nhiệm vụ kiểm thử chức năng và kiểm thử tự động.

Mặc dù hệ thống VNEDU sở hữu số lượng chức năng lớn và liên tục mở rộng nhưng khi xem xét các chức năng trên nhiều phân hệ tính năng khác nhau dựa trên tài liệu đặc tả yêu cầu (software requirement specification - SRS) bao gồm các đặc tả về chức năng và phi chức năng của hệ thống, có thể nhận thấy các tính năng đều xoay quanh một số dạng chức năng như sau:

- Nhóm chức năng năng CRUD (Create - Read - Update - Delete): chiếm khoảng 35% tổng số chức năng ví dụ tạo mới học sinh, cập nhật thông tin lớp học, xóa thông tin giáo viên,…

● Nhóm chức năng tra cứu: chiếm khoảng 25% tổng số chức năng ví dụ như tra cứu học sinh, tra cứu điểm,...

● Nhóm chức năng thống kê báo cáo: chiếm khoảng 15% phục vụ báo cáo kết quả học tập, chuyên cần, xếp loại,…

● Nhóm chức năng cấu hình hệ thống: chiếm khoảng 10% ví dụ như cấu hình môn học, cấu hình nhập điểm,...

● Nhóm chức năng phân quyền và quản lý tài khoản chiếm khoảng 8% ví dụ như tạo tài khoản, cấp quyền, phân quyền theo vai trò,...

● Nhóm tính năng nhập và xuất dữ liệu chiếm khoảng 7% ví dụ như nhập danh sách học sinh từ Excel, xuất báo cáo

Mặc dù hệ thống VNEDU đã triển khai kiểm thử tự động cho nhiều chức năng nhưng hoạt động kiểm thử tự động hiện nay vẫn được thực hiện hoàn toàn dựa vào năng lực lập trình của kiểm thử viên, kiểm thử viên dựa vào test case và giao diện web thực tế của hệ thống để viết từng bước kiểm thử, chưa tận dụng các công nghệ mới như LLM. Một số vấn đề trong hoạt động kiểm thử tự động cần xem xét trước khi áp dụng các công nghệ mới để sinh mã kiểm thử chức năng như sau:

● Vấn đề 1: Đầu vào của hoạt động kiểm thử tự động chưa được thu thập và lưu trữ tập trung: Mỗi khi cần viết script cho một chức năng, kiểm thử viên đọc test case để hiểu nghiệp vụ, xác định các màn hình giao diện sẽ tương tác sau đó mở màn hình tương ứng trên trình duyệt, bật F12 dùng công cụ Kiểm tra (Inspect) để xem cấu trúc giao diện DOM, sau đó tự xác định phần tử giao diện và viết Xpath, keyword rồi ghép lại thành testcase. Hiện tại hai nguồn đầu vào chưa được thu thập và lưu trữ tập trung.

● Vấn đề 2: Test case có cấu trúc tách biệt nhưng test script lại gộp toàn bộ thông tin vào một dạng duy nhất: Test case được thể hiện dưới dạng bảng bao gồm các cột như tiền điều kiện, các bước kiểm thử, dữ liệu đầu vào và kết quả mong đợi.

● Vấn đề 3: Sự đa dạng và phức tạp của Xpath, Keyword, Test case: Ví dụ Xpath có thể được trích xuất dựa vào thuộc tính ổn định, dựa vào nhãn (label) hoặc được trích xuất khi thực hiện một thao tác tiền điều kiện trước đó. Keyword có thể

chứa thao tác tác động trực tiếp đến trạng thái giao diện hoặc keyword chỉ kiểm tra trạng thái giao diện. Test case chỉ thao tác trên màn hình thuộc một tính năng hoặc test case thao tác trên nhiều màn hình thuộc các tính năng khác nhau.

- Vấn đề 4: Việc tạo ra test script (cấu trúc dự án, XPath, keyword, test case) phụ thuộc hoàn toàn vào kinh nghiệm cá nhân của từng kiểm thử viên. Kiểm thử viên phải tự xây dựng cấu trúc dự án, xác định XPath, tạo ra các loại keyword, ghép test case dựa trên xpath và keyword đã xây dựng, mỗi kiểm thử viên có cách xây dựng khác nhau không có chuẩn chung cho từng loại.
- Vấn đề 5: Test case bao gồm các thông tin về phần tử giao diện, keyword, dữ liệu kiểm tra: Trong test case, các thành phần như phần tử giao diện, keyword và dữ liệu kiểm tra thường được ghi chung trong cùng một câu mô tả, không được đánh dấu hay tách biệt dưới bất kỳ hình thức nào. Kiểm thử viên tự xác định liên kết giữa các phần tử giao diện trong test case và xpath bằng cách đọc test case để xác định phần tử giao diện sau đó sẽ xem cấu trúc giao diện DOM để trích xuất xpath tương ứng
- Vấn đề 6: Thao tác lặp lại giữa các testcase không được tách thành keyword dùng chung, các test case chứa các bước giống nhau tạo ra sự dư thừa lớn, khiến các thay đổi nhỏ (ví dụ thay đổi cách chọn giá trị, thay đổi thao tác mở danh sách) phải sửa tại nhiều điểm khác nhau.
- Vấn đề 7: Cơ chế lưu trữ mã kiểm thử chức năng theo cấu trúc: Việc lưu trữ mã kiểm thử chức năng chưa có một cơ chế cấu trúc thống nhất ví dụ xpath lưu trong test case, không tách riêng mã giao diện, mã nghiệp vụ. Điều này khiến quá trình bảo trì trở nên phức tạp: khi giao diện hoặc nghiệp vụ thay đổi, rất khó xác định phần nào bị ảnh hưởng.
- Vấn đề 8: Kết quả kiểm thử tự động sau mỗi lần chạy chưa được hệ thống phân tích hay phân loại một cách tự động. Kiểm thử viên phải tự đọc thủ công và phân tích để xác định nguyên nhân lỗi. Việc này tiêu tốn nhiều thời gian, đòi hỏi kinh nghiệm của từng kiểm thử viên.

Những vấn đề trên cho thấy việc sử dụng mô hình LLM sinh mã kiểm thử chức

năng không chỉ phụ thuộc vào khả năng sinh mã của mô hình mà còn chịu ảnh hưởng bởi dữ liệu đầu vào và đầu ra của mỗi lần sinh mã. Phần tiếp theo sẽ trình bày các lý thuyết liên quan đến LLM, kỹ thuật Prompt Engineering và các công cụ xây dựng dựa trên mô hình LLM hỗ trợ mô hình LLM sinh mã kiểm thử chức năng, quản lý đầu vào và đầu ra.

KẾT LUẬN CHƯƠNG

Chương I đã trình bày lý thuyết về kiểm thử chức năng tự động, giới thiệu hệ thống VNEDU. Đồng thời trình bày hiện trạng hoạt động kiểm thử chức năng tự động trong hệ thống VNEDU, đưa ra các vấn đề cần xem xét trước khi áp dụng mô hình LLM vào sinh mã kiểm thử chức năng cho hệ thống VNEDU.

CHƯƠNG II: LÝ THUYẾT LLM VÀ SỬ DỤNG CÔNG CỤ LLM TRONG KIỂM THỬ CHỨC NĂNG TỰ ĐỘNG

Chương II trình bày lý thuyết về LLM kỹ thuật Prompt Engineering. Đồng thời giới thiệu về các công cụ LLM và lựa chọn các công cụ phù hợp cho bài toán sinh mã kiểm thử chức năng tự động

2.1. Giới thiệu về LLM và kỹ thuật Prompt Engineering

2.1.1. Large Language Models (LLMs):

Large Language Models (LLM) là các mô hình học sâu quy mô lớn được huấn luyện bằng phương pháp tự giám sát trên lượng dữ liệu văn bản rất lớn, từ đó học được các mẫu ngôn ngữ, cấu trúc cú pháp, ngữ nghĩa và khả năng suy luận. Mô hình LLM được huấn luyện bằng nhiệm vụ "dự đoán token tiếp theo" (next-token prediction) để hình thành mô hình ngôn ngữ có khả năng sinh văn bản tự nhiên, lập luận bước đầu (reasoning) và tạo ra nội dung theo yêu cầu người dùng. Nhờ đó, mô hình LLM sau tiền huấn luyện có thể được xem như một mô hình nền (foundation model) có khả năng hiểu, sinh ngôn ngữ và suy luận ở mức tổng quát [6].

LLM và cơ chế In-Context Learning (ICL): Học trong ngữ cảnh (in-context learning) là cơ chế quan trọng cho phép mô hình LLM thích ứng với nhiệm vụ mới chỉ bằng cách thay đổi prompt mà không cần huấn luyện lại mô hình. Mô hình LLM có thể chuyển từ văn bản sang tóm tắt hay sinh mã chỉ bằng cách thay đổi nội dung trong prompt [6]. Nghiên cứu [6] mô tả ba dạng của in-context learning:

- Zero-shot: mô hình được cung cấp mô tả nhiệm vụ mà không có ví dụ mẫu.
- One-shot: mô hình được cung cấp một ví dụ minh hoạ.
- Few-shot: mô hình được cung cấp nhiều ví dụ để học cách suy luận.

Vai trò của ngữ cảnh có cấu trúc đối với LLM: Theo nghiên cứu [14], ngoài nguồn ngữ cảnh là văn bản thuần túy, mô hình LLM có thể kết hợp thêm các dạng dữ liệu có cấu trúc (ví dụ HTML) để mở rộng khả năng suy luận, đặc biệt trong các bài toán đòi hỏi mô hình phải hiểu cấu trúc, quan hệ và bối cảnh phức tạp.

2.1.2. Prompt Engineering:

Prompt Engineering là kỹ thuật thiết kế tương tác đầu vào nhằm đạt được đầu ra chính xác và ổn định từ LLM. Theo nghiên cứu [6], đây là một yếu tố quan trọng quyết định hiệu quả ứng dụng LLM, bởi mô hình chỉ hiểu nhiệm vụ thông qua ngữ cảnh dữ liệu được cung cấp. Một số nguyên tắc thiết kế prompt được đưa ra trong nghiên cứu [6] bao gồm:

- Mô tả nhiệm vụ rõ ràng: LLM suy luận nhiệm vụ từ văn bản trong prompt, yêu cầu mơ hồ dẫn đến đầu ra kém chính xác.
- Hướng dẫn mô hình tư duy: Các chỉ dẫn như "suy nghĩ từng bước" hoặc "phân tích trước khi trả lời" kích hoạt khả năng lập luận, giúp mô hình tạo ra kết quả logic và có cấu trúc.
- Cung cấp thông tin tham chiếu: LLM dựa vào dữ liệu trong prompt để suy luận, ví dụ minh họa hoặc các thông tin tham chiếu giúp mô hình hiểu nhiệm vụ tốt hơn nhờ cơ chế in-context learning.
- Quan tâm đến định dạng đầu ra: yêu cầu đầu ra có định dạng cụ thể giúp mô hình sinh kết quả nhất quán và tuân thủ cấu trúc mong muốn.

Nghiên cứu [6] chỉ ra rằng bằng cách thay đổi nội dung và hình thức của prompt, mô hình LLM có thể được hướng dẫn để thực hiện các tác vụ như:

- Text Classification: LLM có thể được prompt để phân loại một đoạn văn bản vào các nhóm được xác định trước, người dùng mô tả nhiệm vụ và (nếu cần) cung cấp ví dụ trong prompt, và mô hình sẽ nội suy để gán văn bản vào nhóm tương ứng.
- Information Extraction: LLM có khả năng trích xuất thực thể được đặt tên (named entities) và xác định mối quan hệ giữa các thực thể (relation extraction), tức hiểu thực thể nào liên quan hoặc phụ thuộc vào thực thể nào
- Text Generation (Text Completion và Text Transformation): Khi prompt quy định rõ định dạng hoặc cú pháp, mô hình LLM có thể sinh ra nội dung có cấu trúc cụ thể, bao gồm cả việc chuyển đổi từ văn bản tự nhiên sang mã.
- Question Answering: mô hình LLM có thể trả lời câu hỏi dựa trên ngữ cảnh được cung cấp. Prompt thường gồm một đoạn văn bản chứa các lệnh, mô hình LLM

suy luận để tạo câu trả lời chính xác.

2.2. Sử dụng công cụ LLM trong sinh mã kiểm thử chức năng tự động

Trong thực tiễn triển khai, các công cụ dựa trên mô hình LLM thường được phân chia theo cách thức vận hành của mô hình, bao gồm: LLM chạy trên cloud và LLM chạy cục bộ. Việc phân chia theo hình thức triển khai giúp nhận diện rõ hơn cách mỗi loại công cụ tương tác, yêu cầu tích hợp và mức độ kiểm soát mà người dùng có thể đạt được.

- Công cụ LLM chạy trên cloud: Mô hình LLM được vận hành trên hạ tầng của nhà cung cấp dịch vụ và người dùng truy cập thông qua giao diện lập trình ứng dụng (Application Programming Interface - API) hoặc qua các tiện ích tích hợp trong môi trường phát triển tích hợp (Integrated Development Environment - IDE). Hình thức này cho phép người dùng khai thác mô hình mà không cần tự quản lý việc cài đặt, lưu trữ hoặc vận hành mô hình. Các công cụ như GitHub Copilot, Amazon Code Whisperer hoặc Chat GPT sử dụng cơ chế này.
- Công cụ LLM chạy cục bộ: Đây là các mô hình LLM được triển khai trực tiếp trên máy người dùng, thường dựa trên các mô hình LLM mã nguồn mở và được chạy thông qua các công cụ như Ollama, LM Studio,... Hình thức này cho phép người dùng toàn quyền kiểm soát dữ liệu và quá trình vận hành mô hình LLM, đồng thời không phụ thuộc vào kết nối mạng.

Các công cụ LLM được phát triển dựa trên những đặc trưng kỹ thuật khác nhau, trong đó có một số đặc trưng phù hợp cho bài toán sinh mã kiểm thử chức năng tự động ví dụ như:

- Khả năng xử lý và diễn giải yêu cầu ngôn ngữ tự nhiên giúp hiểu và phân tích test case dưới dạng mô tả ngôn ngữ tự nhiên
- Khả năng sinh mã có cấu trúc, mã tạo ra tuân thủ đúng cú pháp và định dạng của framework kiểm thử
- Khả năng tiếp nhận và kết hợp nhiều dạng ngữ cảnh khác nhau từ dự án như HTML, test case
- Khả năng tích hợp với môi trường phát triển hoặc công cụ bên ngoài để khai

thác trực tiếp ngữ cảnh dự án.

Khi xem xét các đặc trưng này trong thực tế triển khai, có thể thấy rằng nhóm công cụ LLM chạy trên cloud đáp ứng đầy đủ và ổn định hơn so với các mô hình cục bộ, bởi các dịch vụ như GitHub Copilot, Cursor hay Chat GPT vừa hỗ trợ sinh mã dựa trên ngôn ngữ tự nhiên, vừa bao quát ngữ cảnh dự án thông qua tích hợp IDE/API, đồng thời mô hình LLM được duy trì và phát triển liên tục bởi nhà cung cấp, người dùng chỉ trả phí theo gói dịch vụ hoặc mức sử dụng mà không cần đầu tư vào hạ tầng phần cứng hay chi phí bảo trì mô hình. Do đó, công cụ LLM chạy trên cloud được xem là lựa chọn phù hợp nhất cho bài toán sinh mã kiểm thử chức năng tự động. Một số công cụ LLM chạy trên cloud điển hình:

- Cursor: là một IDE ra mắt năm 2023 và phát triển dựa trên mã nguồn mở của Visual Studio Code. Công cụ bổ sung một lớp xử lý chuyên biệt để kết nối trực tiếp với các mô hình LLM như GPT-4o, GPT-4 Turbo, Claude 3, Llama 3 cho phép phân tích và chỉnh sửa toàn bộ dự án (project) thay vì chỉ hỗ trợ trong phạm vi một tệp. Người dùng có thể tương tác qua dạng chat, chỉnh sửa dòng (inline) hoặc sử dụng các prompt để thực hiện thay đổi đa tệp trong dự án. Cursor được cung cấp theo các gói Free, Pro (20 USD/tháng) và Business (40 USD/người/tháng). Cursor cho phép người dùng tùy chọn sử dụng API của OpenAI hoặc Anthropic trực tiếp trong IDE, do đó có thể phát sinh thêm chi phí token bổ sung nếu sử dụng các mô hình GPT-4o hoặc Claude với tần suất cao

- Chat GPT: ra mắt năm 2022 được sử dụng chủ yếu qua giao diện chat hoặc thông qua API, trong trường hợp dùng API, toàn bộ ngữ cảnh phải được dồn vào một yêu cầu duy nhất trước khi mô hình xử lý. Chat GPT hỗ trợ nhiều mô hình khác nhau như GPT-3.5, GPT-4, GPT-4 Turbo và GPT-4o, cho phép thực hiện các tác vụ từ trả lời câu hỏi, phân tích tài liệu, hỗ trợ lập trình đến sinh mã theo yêu cầu. Chat GPT không tích hợp trực tiếp vào IDE nên không thể tự truy cập cấu trúc dự án trừ khi người dùng cung cấp mã nguồn hoặc tệp liên quan, các dữ liệu sinh ra phải sao chép bằng tay hoặc phải tự xây dựng thêm cơ chế lưu trữ. Về chi phí, Chat GPT được cung cấp theo hình thức thuê bao cố định khi sử dụng giao diện chat (20 USD/tháng cho

bản Plus), trong khi API tính theo token dựa trên lượng dữ liệu vào và ra

- Github Copilot: được GitHub và OpenAI triển khai từ năm 2021 và tích hợp trực tiếp trong các IDE phổ biến như Visual Studio Code, Visual Studio và JetBrains. Công cụ vận hành dựa trên các mô hình LLM hiện đại của OpenAI, từ Codex, GPT-3.5 đến GPT-4 và GPT-4o. Nhờ đó, Copilot có khả năng hiểu ngôn ngữ tự nhiên, phân tích ngữ cảnh mã nguồn và đưa ra gợi ý mã, giải thích logic hoặc đề xuất chỉnh sửa một cách thông minh. Bên cạnh gợi ý inline, Copilot cung cấp Copilot Chat và chế độ agent giúp mô hình xử lý các tác vụ phức tạp, phân tích cơ sở mã nguồn, đọc các tệp liên quan, đề xuất chỉnh sửa tệp và chạy các lệnh và kiểm tra đầu cuối, tự động sửa lỗi trong một vòng lặp cho đến khi tác vụ hoàn thành. Công cụ được phân phối theo các gói Free, Individual (10 USD/tháng), PRO (20 USD/tháng), Business (19 USD/người/tháng) và Enterprise (39 USD/người/tháng), với ưu điểm là chi phí cố định, không phát sinh token và tuân thủ đầy đủ các tiêu chuẩn bảo mật doanh nghiệp của GitHub

Bảng 2.1. Tổng quan về các công cụ LLM sinh mã kiểm thử chức năng tự động

Đặc tính	Chat Gpt	Cursor	Github Copilot
Mô hình LLM sử dụng	GPT-3.5, GPT-4, GPT-4 Turbo, GPT-4o	GPT-4o, GPT-4 Turbo, Claude 3, Gemini, Llama 3, và mô hình Cursor nội bộ	GPT-4o, GPT-4.1, gói PRO cho phép dùng thêm các mô hình nâng cao GPT-5 series, Claude 3.x, Gemini 2.5,...
Tích hợp IDE	Không tích hợp sẵn	IDE riêng, xây dựng dựa trên VS Code	Tích hợp trực tiếp vào IDE phổ biến VS Code, Visual Studio, JetBrains
Khả năng đọc toàn bộ dự án (project)	Chỉ đọc nội dung người dùng cung cấp	Đọc toàn bộ (project)	Một phần (dựa theo ngữ cảnh các tệp truyền vào)
Cơ chế agent	Có	Có	Có
Cơ chế lưu trữ đầu	Người dùng tự sao	Mã sinh ra dưới	Mã sinh ra dưới

ra	chép hoặc xây thêm cơ chế để lưu trữ	dạng xem trước (preview) để người dùng xác nhận hoặc từ chối lưu trữ	dạng xem trước (preview) để người dùng xác nhận hoặc từ chối lưu trữ
Chi phí	API tính token; Chat Plus 20 USD/tháng	Free, 10 USD (Individual), 20 USD (PRO), 19 USD (Business), 39 USD (Enterprise)	Free; Pro 20 USD/tháng; Business 40 USD/tháng và tính thêm phí token khi dùng API của OpenAI/Anthropic

Dựa trên các tiêu chí đã phân tích, GitHub Copilot là công cụ phù hợp nhất để áp dụng vào bài toán sinh mã kiểm thử chức năng tự động. GitHub Copilot sử dụng mô hình GPT-4o/GPT-4.1, một trong những mô hình LLM ổn định và mạnh trong việc xử lý ngôn ngữ tự nhiên, gói PRO cho phép lựa chọn thêm nhiều mô hình như GPT-5 series hay Claude 3.x, mang lại khả năng tùy biến và nâng cao chất lượng sinh mã khi cần thiết. GitHub Copilot có giao diện GitHub Copilot Chat dễ dàng sử dụng và tích hợp cơ chế Agent cho phép xử lý các tác vụ phức tạp. GitHub Copilot có khả năng tích hợp các IDE phổ biến như VS Code, Visual Studio và JetBrains, giúp công cụ khai thác chính xác ngữ cảnh của tệp đang mở mà không cần người dùng cung cấp thủ công như Chat GPT và cũng không buộc phải chuyển sang một IDE mới hoàn toàn như Cursor. Github Copilot cung cấp cơ chế hiển thị mã dưới dạng xem trước (preview) và yêu cầu người dùng xác nhận trước khi chèn vào tệp, chi phí sử dụng Copilot cũng rõ ràng và ổn định theo tháng, không tính theo token như Chat GPT API và không phát sinh chi phí API thêm như khi sử dụng các mô hình nâng cao trong Cursor, đồng thời cho phép tắt các tùy chọn gửi dữ liệu, đảm bảo an toàn cho dữ liệu trong đề án

2.3. Công nghệ và công cụ sử dụng

Trong kiểm thử chức năng tự động, việc xây dựng và vận hành kịch bản kiểm thử đòi hỏi sự phối hợp của nhiều nhóm công cụ khác nhau bao gồm: công cụ LLM

sinh mã hỗ trợ lập trình, môi trường phát triển để tổ chức cấu trúc dự án kiểm thử, và khung kiểm thử (testing framework) để thực thi các kịch bản sinh ra. Dựa trên các phân tích trong chương 2.2 học viên lựa chọn Github Copilot Chat chế độ Agent sử dụng mô hình LLM mặc định GPT4-o đóng vai trò là công cụ LLM sinh mã kiểm thử chức năng.

Có nhiều môi trường phát triển hỗ trợ kiểm thử tự động như JetBrains PyCharm, IntelliJ tuy nhiên đề án lựa chọn Visual Studio Code (VS Code) vì đây là IDE nhẹ, linh hoạt và có mức độ tương thích cao nhất với GitHub Copilot Chat. VS Code cho phép mở và quản lý toàn bộ không gian làm việc (workspace) của dự án, hỗ trợ nhiều định dạng tệp (HTML, txt,...) làm ngữ cảnh cho mô hình LLM khi sinh mã. VS Code có công cụ mở rộng (extension) phong phú tích hợp nhiều framework kiểm thử tự động như Selenium, Robot Framework, khả năng tùy biến cao và hoàn toàn miễn phí giúp VS Code trở thành lựa chọn phù hợp nhất cho môi trường phát triển.

Trong số các framework kiểm thử như Selenium, Katalon Studio, Robot Framework, đề án lựa chọn Robot Framework vì được tích hợp sẵn trong môi trường phát triển VS Code, các lệnh được xây dựng dựa trên hướng keyword-driven testing rõ ràng và dễ đọc. Ngoài ra, Robot Framework là mã nguồn mở, hoàn toàn miễn phí, dễ mở rộng và tích hợp với SeleniumLibrary cung cấp các từ khóa thao tác giao diện web dựa trên XPath. Ngoài ra Robot Framework còn cung cấp báo cáo dưới dạng các tệp trực quan rõ ràng dễ đọc chứa thông tin về kết quả thực thi bao gồm thời gian thực thi từng test case, số lượng test case đạt, số lượng test case lỗi.

KẾT LUẬN CHƯƠNG

Chương II đã giới thiệu tổng quan về mô hình LLM và kỹ thuật Prompt Engineering là hai yếu tố then chốt trong việc đảm bảo mô hình hiểu đúng yêu cầu đầu vào và sinh ra mã kiểm thử chức phù hợp. Prompt mô tả nhiệm vụ và ngữ cảnh giúp mô hình LLM hiểu đúng, chuyển đổi các các bước kiểm thử thủ công thành mã kiểm thử chức năng tự động. Chương đã phân tích việc sử dụng công cụ LLM trong sinh mã kiểm thử chức năng tự động, đánh giá và xác định các công nghệ và công cụ phù hợp với bài toán sinh mã kiểm thử chức năng tự động.

CHƯƠNG III: XÂY DỰNG QUY TRÌNH SINH MÃ KIỂM THỬ CHỨC NĂNG TỰ ĐỘNG SỬ DỤNG LLM

Chương III xây dựng quy trình sử dụng mô hình LLM để chuyển đổi đầu vào gồm các tệp HTML chứa cấu trúc giao diện và test case chứa các bước kiểm thử thủ công thành các mã kiểm thử chức năng.

3.1. Tổng quan các thành phần cần thiết để thực hiện sinh mã chức năng tự động

Để xây dựng một quy trình sinh mã kiểm thử chức năng tự động bằng mô hình LLM, trước hết cần xác định đầy đủ các thành phần tạo nên quy trình xử lý. Các thành phần như thu thập đầu vào, phân loại đầu vào, xây dựng prompt mẫu, tiền xử lý và liên kết dữ liệu, phân tích chỉnh sửa kết quả sinh mã được thực hiện bởi kiểm thử viên, nhằm đảm bảo dữ liệu đầu vào rõ ràng, ngữ cảnh đầy đủ cho mô hình và đảm bảo chất lượng mã do LLM sinh ra. Hai thành phần sinh mã kiểm thử và lưu trữ quản lý mã sinh ra được tạo ra bởi mô hình LLM.

- Thành phần thu thập đầu vào tập hợp mã HTML của giao diện và toàn bộ các bước kiểm thử thủ công trong đó các bước kiểm thử thủ công ban đầu được chia thành 2 tệp, tệp keyword chứa các hành động kiểm thử và đối tượng kiểm thử (mỗi đối tượng kiểm thử được xác định bởi một Xpath trong tệp HTML), tệp test case chứa các keyword. Kết quả thu được là tập thông tin đầu vào đầy đủ, được đưa vào một không gian lưu trữ thống nhất dùng để làm đầu vào cho mô hình LLM sinh mã kiểm thử chức năng.
- Thành phần phân loại đầu vào mục tiêu là nhận diện và gom nhóm các dạng XPath, các loại keyword và các kiểu test case theo mức độ phức tạp, điều kiện xuất hiện và cách thức được hình thành.
- Thành phần tiếp theo xây dựng prompt mẫu dựa trên phân loại dữ liệu và phân tầng cấu trúc dự án: dựa trên dữ liệu đầu vào đã được phân loại theo nhóm đặc trưng (các loại XPath, các dạng keyword và các kiểu test case) và cấu trúc phân tầng của toàn bộ dự án kiểm thử (tầng project, tầng XPath, tầng keyword và tầng test case) triển khai xây dựng bộ prompt mẫu nhằm hỗ trợ mô hình LLM sinh mã.

- Thành phần tiền xử lý và liên kết dữ liệu tập trung vào việc xử lý test case và keyword và đánh dấu các thành phần để chuẩn bị đầu vào cho việc sinh mã kiểm thử chức năng. Mỗi bước kiểm thử được đưa về một nghiệp vụ duy nhất và các phần tử giao diện, hành động và dữ liệu được đánh dấu để mô hình biết phần nào trong test case cần ánh xạ với phần tử HTML hoặc keyword tương ứng.
- Thành phần sinh mã kiểm thử tự động sử dụng ba nguồn thông tin làm ngữ cảnh đầu vào cho mô hình LLM: HTML, các bước kiểm thử thủ công (keyword, test case), bộ prompt mẫu. Mô hình đọc các loại đầu vào và thực thi các lệnh trong prompt tương ứng để sinh ra mã kiểm thử chức năng
- Thành phần phân tích và chỉnh sửa mã sinh ra: Kiểm thử viên kiểm tra mã chưa phù hợp, chưa đáp ứng đúng yêu cầu nghiệp vụ sau đó cung cấp đoạn mã sai và đoạn mã đúng làm đầu vào để mô hình chỉnh sửa.
- Thành phần lưu trữ và quản lý mã sinh ra: Mô hình LLM sinh mã kiểm thử chức năng và tự lưu lại dưới dạng chờ xác nhận dựa theo cấu trúc thư mục (Xpath lưu trong thư mục Repository, Keyword lưu trong thư mục Resource, Test case lưu trong thư mục Testcases) thông qua các hướng dẫn trong prompt theo tầng và prompt theo loại

3.2. Thu thập đầu vào dựa trên đánh giá thực trạng

Dựa trên đánh giá thực trạng về hoạt động kiểm thử tự động của hệ thống VNEDU**.** Kiểm thử viên tiến hành thu thập đầu vào, gồm hai công việc chính nhằm chuẩn bị đầy đủ dữ liệu cho việc sinh mã kiểm thử chức năng.

- Thu thập HTML của từng màn hình giao diện. Kiểm thử viên mở giao diện web cần kiểm thử, sử dụng công cụ Kiểm tra (F12) của trình duyệt để lấy HTML của toàn bộ màn hình giao diện chứa các phần tử cần thao tác. Trước khi lưu trữ, tester loại bỏ các thông tin nhạy cảm như domain, token, user id. HTML sau khi loại bỏ thông tin nhạy cảm được lưu lại thành từng tệp dạng html theo đúng tên màn hình trong thư mục HTML, tạo nguồn dữ liệu thống nhất cho mô hình LLM sinh XPath nhận diện phần tử giao diện.
- Thu thập test case, kiểm thử viên thu thập test case và gộp các phần trong

test case (tiền điều kiện, các bước kiểm thử, dữ liệu kiểm tra và kết quả mong đợi) thành các tệp dạng txt thư mục STEP_TESTCASE chứa bước theo đúng trình tự nghiệp vụ. Ngoài ra, các bước thao tác lặp lại được gom nhóm thành các keyword dùng để tái sử dụng trong nhiều test case, mỗi keyword đại diện cho một cụm hành động được lưu trữ dưới dạng các file txt tại thư mục STEP_KEYWORD. Như vậy từ các test case ban đầu sẽ được chia làm 2 dạng đầu vào (test case chứa các keyword, keyword chứa các bước thao tác và phần tử giao diện)

Thông qua hai hoạt động này, đầu vào của mô hình LLM bao gồm giao diện (HTML) và các bước kiểm thử thủ công được thu thập đầy đủ, làm cơ sở cho các thành phần xử lý tiếp theo. Như vậy thành phần này giải quyết vấn đề 1 (đầu vào phân tán, không được lưu trữ tập trung) và vấn đề 2 (test case có cấu trúc tách biệt nhưng script lại gộp toàn bộ thông tin vào một dạng duy nhất) đã phân tích trong chương 1.3.

3.3. Phân loại đầu vào

Thành phần phân loại nhằm giải quyết vấn đề về sự đa dạng của dữ liệu. Việc phân loại được triển khai theo hai hướng: phân loại theo nhóm và phân loại theo tầng. Hai cách phân loại này bổ sung cho nhau: phân loại theo nhóm giúp mô hình hiểu đặc tính và sự phức tạp của từng loại dữ liệu, trong khi phân loại theo tầng giúp mô hình hiểu mối quan hệ và vai trò của chúng trong toàn bộ dự án kiểm thử

3.3.1. Nguyên tắc phân loại đầu vào

Dữ liệu test case sau khi thu thập ở chương 3.1 chứa thông tin sau: phần tử giao diện (tương ứng với Xpath), keyword, test case. Các thông tin này mang những đặc tính riêng về cấu trúc, cách thức hình thành, phạm vi và mức độ phức tạp khiến chúng không thể được xử lý theo một quy tắc chung duy nhất. Chính sự khác biệt này là cơ sở để kiểm thử viên phân chia dữ liệu thành các nhóm tương ứng.

- Phân loại dựa trên đặc trưng cấu trúc: XPath tĩnh có cấu trúc cố định và không chứa tham số, XPath động có cấu trúc biến đổi theo tham số hoặc ngữ cảnh
- Phân loại dựa trên cách thức hình thành: Các keyword thao tác giao diện (như click, nhập liệu, chọn giá trị) được hình thành từ các hành động mang tính thao tác; trong khi đó các keyword kiểm tra giao diện lại hình thành từ các bước xác nhận

trạng thái và mang tính kiểm tra.

- Phân loại dựa trên phạm vi và mức độ phức tạp: Các test case một tính năng thường có luồng nghiệp vụ nhỏ, số lượng bước không lớn, test case nhiều tính năng bao gồm nhiều nghiệp vụ, liên kết nhiều màn hình.

3.3.2. Phân loại đầu vào

Đầu vào của mô hình LLM được tổ chức theo cơ chế phân loại theo nhóm nhằm tách bạch các đặc trưng của XPath, keyword và test case. Việc phân loại theo nhóm kiểm soát phạm vi của từng loại dữ liệu, nhằm xây dựng prompt tương ứng với từng loại dữ liệu và đánh giá khả năng sinh mã kiểm thử chức năng của mô hình LLM đối với từng dạng đầu vào.

- Đối với nhóm XPath, dữ liệu được chia thành hai nhóm nhỏ phản ánh mức độ phức tạp khác nhau của cấu trúc XPath.

  ○ Nhóm XPath tĩnh: bao gồm các phần tử có thuộc tính cố định, không phụ thuộc vào dữ liệu hay trạng thái giao diện, thường được sinh ra từ các thuộc tính như id, name hoặc class ổn định.

  ○ Nhóm XPath động (còn gọi là XPath chứa tham số), được sử dụng khi một XPath phải thay đổi linh hoạt theo dữ liệu hoặc theo nhiều giá trị khác nhau; dạng này thường có cấu trúc chứa biến, cho phép truyền tham số vào để định vị nhiều phần tử có cùng thuộc tính trong bảng, danh sách hay dropdown.

- Nhóm Keyword được chia thành hai loại tương ứng với bản chất của hành động kiểm thử: nhóm thao tác yêu cầu sinh mã tác động đến trạng thái giao diện, còn nhóm kiểm tra chỉ yêu cầu sinh mã xác minh trạng thái giao diện

  ○ Nhóm keyword tương tác giao diện đại diện cho các bước kiểm thử có thao tác trên giao diện như click, nhập liệu, chọn giá trị, tương tác dropdown hoặc thao tác thay đổi trạng thái giao diện.

  ○ Nhóm keyword kiểm tra giao diện được sử dụng cho các bước xác minh như kiểm tra phần tử hiển thị, kiểm tra giá trị hoặc trạng thái, và không làm thay đổi giao diện.

- Nhóm test case, dữ liệu được phân chia thành hai dạng dựa trên phạm vi

nghiệp vụ.

○ Nhóm test case chứa một tính năng là các kịch bản tập trung vào một chức năng (không tính chức năng cơ bản như đăng nhập đăng xuất, do toàn bộ test case trong hệ thống đều sử dụng chức năng này), có số lượng thao tác ngắn, thao tác trực tiếp với một màn hình hoặc một nghiệp vụ cụ thể.

○ Nhóm test case chứa nhiều tính năng bao gồm các luồng kiểm thử dài, liên quan đến nhiều màn hình, nhiều điểm chuyển trạng thái. Sự phân loại này giúp đánh giá chính xác khả năng của mô hình LLM khi sinh mã cho luồng đơn lẻ so với luồng phức tạp, từ đó phản ánh mức độ áp dụng vào thực tế.

Cách phân loại này sẽ giải quyết vấn đề 3 trong chương 1.3 về sự đa dạng và phức tạp của dữ liệu trong hệ thống. Việc phân loại này nhằm tạo ra một tập dữ liệu đầu vào đã được phân nhóm để xây dựng các prompt mẫu phù hợp cho từng nhóm dữ liệu.

3.4. Xây dựng mẫu prompt

Việc xây dựng mẫu prompt được phát triển dựa trên các lý thuyết Prompt Engineering đã trình bày ở Chương II bao gồm prompt đa tầng và prompt theo nhóm. Cách thiết kế này chia nhỏ nhiệm vụ thành nhiều lớp xử lý, mỗi lớp giải quyết một phần riêng biệt của bài toán nhằm giảm tải cho mô hình và cải thiện độ chính xác.

● Prompt theo tầng:

○ Prompt Project định nghĩa tổ chức thư mục, nguyên tắc quản lý tài nguyên để các prompt ở các tầng tiếp theo sinh mã theo đúng cấu trúc ví dụ như: thư mục html để lưu trữ đầu vào là các tệp HTML chứa cấu trúc giao diện của từng màn hình, step_keyword lưu trữ đầu vào là mô tả keyword của từng chức năng, step_testcase lưu trữ đầu vào là các test case, repository lưu trữ xpath của từng màn hình, Resource lưu trữ keyword và Testcases để lưu trữ test case,....

○ Prompt Xpath chứa các lệnh yêu cầu mô hình phân tích giao diện web ở dạng DOM và mô tả test case để trích xuất ra xpath theo quy tắc, tách bạch việc sinh mã giao diện và việc sinh mã nghiệp vụ.

○ Prompt Keyword tạo ra keyword với tên gọi khớp với mô tả kiểm thử, danh

sách tham số rõ ràng và các câu lệnh sử dụng chính xác XPath tương ứng.

○ Prompt Testcase sinh ra keyword theo đúng thứ tự nghiệp vụ, truyền tham số thích hợp.

● Prompt theo nhóm:

○ Với nhóm XPath, hệ thống xây dựng 2 loại prompt tương ứng với 2 dạng XPath có đặc trưng xử lý khác nhau: prompt cho XPath tĩnh tập trung vào việc hướng dẫn mô hình LLM tạo ra các biến giao diện dựa trên thuộc tính cố định có trong HTML, prompt cho XPath động (chứa tham số) hướng dẫn mô hình sinh ra các XPath có chứa giá trị động được biểu diễn dưới dạng tham số.

○ Đối với nhóm keyword, hai loại prompt được xây dựng tương ứng với hai dạng hành vi kiểm thử chính. Prompt dành cho keyword tương tác giao diện quy định mô hình sinh ra các bước thao tác như click, nhập liệu hay chọn dữ liệu. Prompt dành cho keyword kiểm tra giao diện yêu cầu mô hình chỉ sinh bước kiểm tra như kiểm tra hiển thị, kiểm tra text hoặc trạng thái phần tử, tuyệt đối không được sinh thao tác làm thay đổi giao diện. Cả hai loại prompt đều yêu cầu kèm theo việc ánh xạ biến XPath từ các tệp Xpath đã sinh ra trước đó

○ Đối với nhóm test case, hai loại prompt được thiết kế theo phạm vi của luồng nghiệp vụ. Prompt cho test case chứa một tính năng hướng mô hình sinh mã theo cấu trúc ít bước, chỉ kết hợp các keyword liên quan đến một chức năng duy nhất. Trong khi đó, prompt cho test case chứa nhiều tính năng yêu cầu mô hình đọc toàn bộ luồng kiểm thử phức tạp, kết hợp nhiều keyword thuộc nhiều tính năng khác nhau, đồng thời đảm bảo được trình tự các bước.

Việc kết hợp hai nhóm prompt: theo loại dữ liệu và theo tầng giúp mô hình hiểu không chỉ bản chất của từng loại đầu vào, mà còn hiểu rõ vị trí, vai trò và mối quan hệ của chúng trong tổng thể dự án kiểm thử tự động. Thành phần này sẽ giải quyết vấn đề 4, khi trước đây việc xây dựng XPath, keyword và test case phụ thuộc hoàn toàn vào kinh nghiệm và phong cách cá nhân của kiểm thử viên, không có chuẩn chung cho từng loại. Các loại đầu vào như XPath tĩnh, Xpath động, Keyword thao tác với giao diện, Keyword kiểm tra giao diện, Test case chứa một tính năng, Test case

chứa nhiều tính năng được xây dựng dựa trên prompt chung cho từng loại. Còn prompt phân tầng, mô hình LLM cũng xác định được mã sinh ra cần đưa vào vị trí: biến giao diện vào tầng XPath, keyword vào tầng Keyword, test case vào tầng test case, tách riêng mã giao diện và mã nghiệp vụ

3.5. Tiền xử lý và liên kết dữ liệu hỗ trợ

Do keyword và test case đều xuất phát từ cùng một nguồn là các bước kiểm thử thủ công, bước tiền xử lý được chia thành hai hướng: tiền xử lý cho keyword và tiền xử lý cho testcase. Mục tiêu của bước này là tiền xử lý dữ liệu và thiết lập quy tắc biểu diễn thống nhất để mô hình LLM có thể hiểu, ánh xạ và sinh mã kiểm thử chức năng

- Tiền xử lý Keyword: Mỗi keyword được viết theo một cấu trúc bán chuẩn, keyword bán chuẩn là dạng mô tả hành động kiểm thử được chuẩn hóa một phần từ bước kiểm thử thủ công. Mỗi keyword bán chuẩn chỉ biểu diễn một hành động duy nhất, trong đó các mô tả tương ứng với phần tử giao diện được đánh dấu trong dấu [], các mô tả tương ứng với các dữ liệu thay đổi được đánh dấu trong () nhằm giúp mô hình LLM dễ suy luận và liên kết dữ liệu với giao diện web. Các ký hiệu như dấu ngoặc vuông, ngoặc đơn giữ vai trò như lớp meta-information (các thông tin bổ sung hoặc mô tả về dữ liệu chính, ngữ cảnh, cấu trúc, hoặc cách mà dữ liệu đó được tạo ra và sử dụng) để mô hình nhận biết đâu là phần tử giao diện cần xác định Xpath, đâu là dữ liệu cần tham số hóa

  - Dấu ngoặc vuông [] được sử dụng để biểu diễn phần tử giao diện theo tên gần nghĩa với phần tử trong HTML, chẳng hạn như dropdown, textbox, button hoặc một ô bảng. Việc dùng ký hiệu này giúp mô hình LLM xác định bất cứ cụm nào nằm trong ngoặc vuông đều được hiểu là phần tử giao diện và phải được đối chiếu với HTML để lấy XPath.

  - Dấu ngoặc tròn () được dùng để đánh dấu các giá trị thay đổi theo dữ liệu như tên, ngày sinh, các giá trị trong bộ lọc. Các tham số này là dữ liệu đầu vào của keyword, đồng thời cũng là thành phần chứa trong XPath động. Việc tách dữ liệu động vào ngoặc tròn giúp mô hình phân biệt được giữa hành động và dữ liệu, đồng

thời đảm bảo keyword có thể tái sử dụng ở nhiều kịch bản khác nhau mà không bị cố định theo dữ liệu của một test case cụ thể.

- Tiền xử lý Test case: Test case thủ công đóng vai trò mô tả luồng nghiệp vụ tổng thể, còn keyword là đơn vị thao tác nhỏ hơn, thể hiện các hành động cụ thể mà hệ thống cần thực hiện.
  - Các testcase thủ công được phân tách thành từng bước riêng biệt, mỗi bước chỉ đại diện cho một hành động duy nhất.
  - Để giúp mô hình nhận biết được đâu là keyword, các cụm hành động trong testcase được đặt trong dấu nháy đơn ‘ ’. Dấu này đóng vai trò định danh, giúp mô hình LLM hiểu rằng đây là một lệnh hoặc hành động cụ thể cần được ánh xạ tới keyword. Ví dụ, bước “Đăng nhập thành công” sẽ ánh xạ với keyword Dang_nhap_thanh_cong.

Thành phần này sẽ giải quyết vấn đề 5 (mô tả test case mơ hồ), vấn đề 6 (không phân biệt được phần tử giao diện, keyword và dữ liệu kiểm tra) và vấn đề 7 (các thao tác lặp lại không được tách thành keyword dùng chung). Kết quả là tập test case đã được xử lý và đánh dấu để có thể liên kết, làm đầu vào cho LLM sinh mã ở các thành phần tiếp theo.

3.6. Sinh mã kiểm thử chức năng tự động

Sau khi hoàn thành các bước phân loại đầu vào, xây dựng mẫu prompt, và tiền xử lý liên kết dữ liệu hỗ trợ chuyển sang sinh mã kiểm thử chức năng tự động. Giai đoạn mô hình LLM được sử dụng để chuyển đổi toàn bộ ngữ cảnh đầu vào bao gồm test case đã được thu thập, phân loại, xử lý, các tệp HTML và prompt thành mã kiểm thử chức năng có thể thực thi. Cơ chế sinh mã kiểm thử chức năng hoạt động dựa trên hệ thống prompt theo tầng và prompt theo loại, việc sinh mã không diễn ra đơn lẻ mà là chuỗi bước có liên kết, đầu ra của bước trước trở thành đầu vào cho bước sau.

- Tầng Project: mô hình LLM không sinh mã kiểm thử chức năng cụ thể, mà sinh ra cấu trúc dự án kiểm thử tự động. Đầu vào cho mô hình LLM là prompt project mô tả cấu trúc dự án, bao gồm các thư mục Repository, Keyword, Resource,

Testcase,... Đầu ra của mô hình LLM là khung dự án kiểm thử gồm các thư mục và tệp cần thiết của một dự án theo nguyên tắc ví dụ như thư mục Repository lưu trữ Xpath

- Tầng XPath sử dụng đầu vào là các tệp HTML và các tệp mô tả keyword bán chuẩn để sinh ra Xpath cho các phần tử nằm trong dấu [] , trong đó mô hình sử dụng các prompt chuyên biệt theo từng nhóm XPath: Prompt XPath tĩnh, Prompt XPath động. Đầu ra tầng này là tập XPath được lưu trữ dưới dạng các biến được đặt tên dựa theo tên phần tử nằm trong dấu [], mỗi biến thể hiện cho một phần tử giao diện duy nhất, phục vụ trực tiếp cho việc ánh xạ Xpath ở tầng Keyword
- Tầng Keyword được thực hiện dựa trên đầu vào là prompt keyword theo loại (keyword thao tác giao diện hoặc keyword kiểm tra giao diện), kết hợp với tệp mô tả keyword bán chuẩn và các Xpath được sinh ra ở tầng Xpath . Ở tầng này, mô hình ánh xạ các phần tử được đánh dấu trong ngoặc vuông với biến XPath được sinh ra ở tầng Xpath, tham số hóa các giá trị trong ngoặc tròn và sinh keyword hoàn chỉnh.
- Tầng Prompt Testcase, mô hình sử dụng mô tả testcase đã được tiền xử lý (các keyword đã được đặt trong dấu nháy đơn) để sinh ra kịch bản kiểm thử hoàn chỉnh. Tầng này thực hiện việc gọi các keyword đã sinh ở tầng Keyword và kết hợp lại theo đúng trình tự các bước trong mô tả test case. Tùy thuộc vào phạm vi nghiệp vụ của test case, mô hình sử dụng đầu vào là prompt testcase cho một tính năng (đối với các luồng đơn, ít bước) hoặc prompt testcase cho nhiều tính năng (đối với các luồng dài, có nhiều màn hình kết hợp). Đầu ra của tầng này là test script hoàn chỉnh tương ứng với một trường hợp kiểm thử.

Thông qua hệ thống prompt theo tầng và theo loại, mô hình LLM không chỉ sinh mã ở từng tầng mà còn hiểu mối quan hệ giữa các tầng: XPath - Keyword - Test case, đầu ra của tầng trước là đầu vào của tầng sau, đảm bảo chuỗi hành động thống nhất và có khả năng thực thi. Mô hình LLM sinh mã kiểm thử và các biến đại diện cho dữ liệu kiểm thử, không thực hiện sinh dữ liệu kiểm thử hoàn chỉnh. Nguyên nhân là dữ liệu kiểm thử có tính thay đổi cao, việc gán giá trị cụ thể cho các biến dữ liệu vẫn được thực hiện bởi kiểm thử viên nhằm bảo đảm tính chính xác và phù hợp

với nghiệp vụ thực tế. Đây là thành phần cốt lõi của quy trình, dù không trực tiếp giải quyết vấn đề nào trong Chương 1.3, nhưng là nơi toàn bộ dữ liệu đầu vào được chuyển thành mã kiểm thử chức năng.

3.7. Phân tích, chỉnh sửa mã kiểm thử chức năng tự động

Mặc dù mô hình LLM đã có khả năng sinh mã kiểm thử chức năng tự động thông qua hệ thống prompt theo tầng và theo loại nhưng để đảm bảo tính khả thi khi triển khai thực tế, toàn bộ mã sinh ra cần được phân tích và chỉnh sửa thủ công bởi tester trước khi được đưa vào chạy chính thức. Bước này đóng vai trò là giai đoạn hậu xử lý trong quy trình, có nhiệm vụ kiểm định, điều chỉnh và hoàn thiện kết quả sinh mã, nhằm bảo đảm rằng mã kiểm thử tự động không chỉ đúng cú pháp mà còn phù hợp với hành vi nghiệp vụ thực tế của hệ thống

Người kiểm thử rà soát lại mã được tạo tự động bằng cách chạy thử test script để đảm bảo sự phù hợp với giao diện thực tế và luồng nghiệp vụ. Với file XPath, tester kiểm tra xem các XPath có trỏ đúng phần tử giao diện, có đủ tính ổn định khi giao diện thay đổi hay không. Với file keyword, tester kiểm tra logic thao tác xem đã đúng chức năng mô tả, thứ tự hành động có phản ánh đúng luồng nghiệp vụ. Với test case, tester kiểm tra sự đầy đủ của các bước và khả năng thực thi liên tục. Trong quá trình thực thi test script nếu xuất hiện lỗi runtime như XPath không trỏ đúng phần tử, keyword thiếu tham số, tester sẽ sử dụng trực tiếp giao diện chat của GitHub Copilot để hướng dẫn mô hình LLM sửa sai. Kiểm thử viên cung cấp cho mô hình cả dòng lỗi thực tế, nguyên nhân lỗi, và phiên bản đúng của đoạn mã cần thay thế. Nhờ cơ chế này, mô hình có thể cập nhật phần mã bị sai trong các tệp, đồng thời học được quy tắc sửa lỗi đúng từ cặp dữ liệu "lỗi sai – mã đúng". Điều này giúp giảm thiểu việc lặp lại lỗi ở các lần sinh mã tiếp theo và nâng cao độ chính xác của mô hình trong cùng một dự án kiểm thử.

Mặc dù thành phần này không giải quyết trực tiếp một vấn đề nào trong Chương 1.3, đây vẫn là thành phần bắt buộc vì mô hình LLM có thể sinh ra những đoạn mã chưa hoàn toàn khớp với giao diện hoặc nghiệp vụ, và cần sự kiểm tra cuối cùng của con người để bảo đảm tính chính xác trước khi chuyển sang bước lưu trữ

3.8. Lưu trữ và quản lý mã

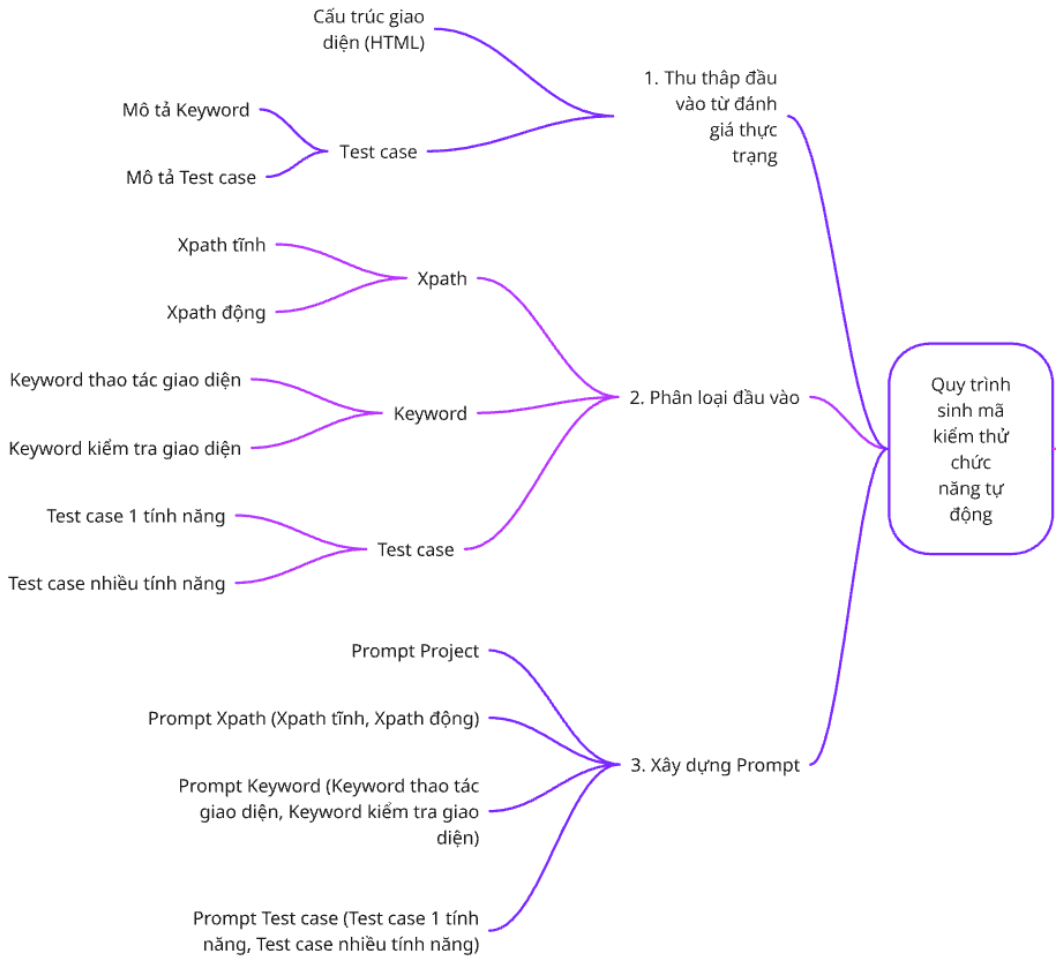
Các mã kiểm thử chức năng do mô hình LLM sinh ra không được ghi trực tiếp vào dự án, mà trước tiên được lưu dưới dạng thay đổi tạm thời (temporary changes). Kiểm thử viên sẽ xem lại nội dung, xác nhận sự chính xác của từng phần rồi mới chấp nhận lưu vào đúng thư mục. Việc lưu trữ không diễn ra tùy ý mà được mô hình LLM sinh ra dựa theo cấu trúc dự án sinh ra trong tầng Prompt Project (Repository, Resource, Testcases) và yêu cầu trong từng prompt theo loại ở các tầng tiếp theo. Ví dụ, thư mục Repository lưu trữ các file XPath được sinh từ tầng XPath, trong đó mỗi prompt XPath theo loại luôn kèm theo yêu cầu như: "Tạo một file XPATH_<TênFile>.robot tại thư mục Repository với <TênFile> trùng tên file HTML". Điều này giúp mô hình sinh ra tên tệp, định dạng tệp và lưu trữ tệp sinh ra ở đâu, tách biệt rõ ràng giữa phần tử giao diện (XPath), hành động kiểm thử (keyword) và luồng nghiệp vụ (test case) nhằm xử lý việc bảo trì và mở rộng.

- Repository: chứa các tệp XPath, được lưu trữ theo từng màn hình
- Resource: chứa các tệp keyword được lưu trữ theo từng tính năng
- Testcases: chứa các test case, mỗi test case đại diện cho một trường hợp nghiệp vụ cần kiểm thử

Ngoại trừ 3 thư mục Repository, Resource, Testcases còn chứa các thư mục phục vụ các chức năng khác như: html để lưu trữ đầu vào là các tệp HTML chứa cấu trúc giao diện của từng màn hình, step_keyword lưu trữ đầu vào là các keyword bán chuẩn của từng chức năng, step_testcase lưu trữ đầu vào là các test case, Results chứa kết quả thực thi của các script, run_test.bat là tệp dùng để chạy tất cả các script trong thư mục Testcases

Thành phần này sẽ giải quyết vấn đề 7 trong Chương 1.3 về việc thiếu cơ chế lưu trữ có cấu trúc. Nhờ cấu trúc thư mục và việc lưu - xác nhận của kiểm thử viên, mã sinh ra được quản lý một cách có cấu trúc, có khả năng tái sử dụng cho lần sau.

3.9. Quy trình sinh mã kiểm thử chức năng tự động



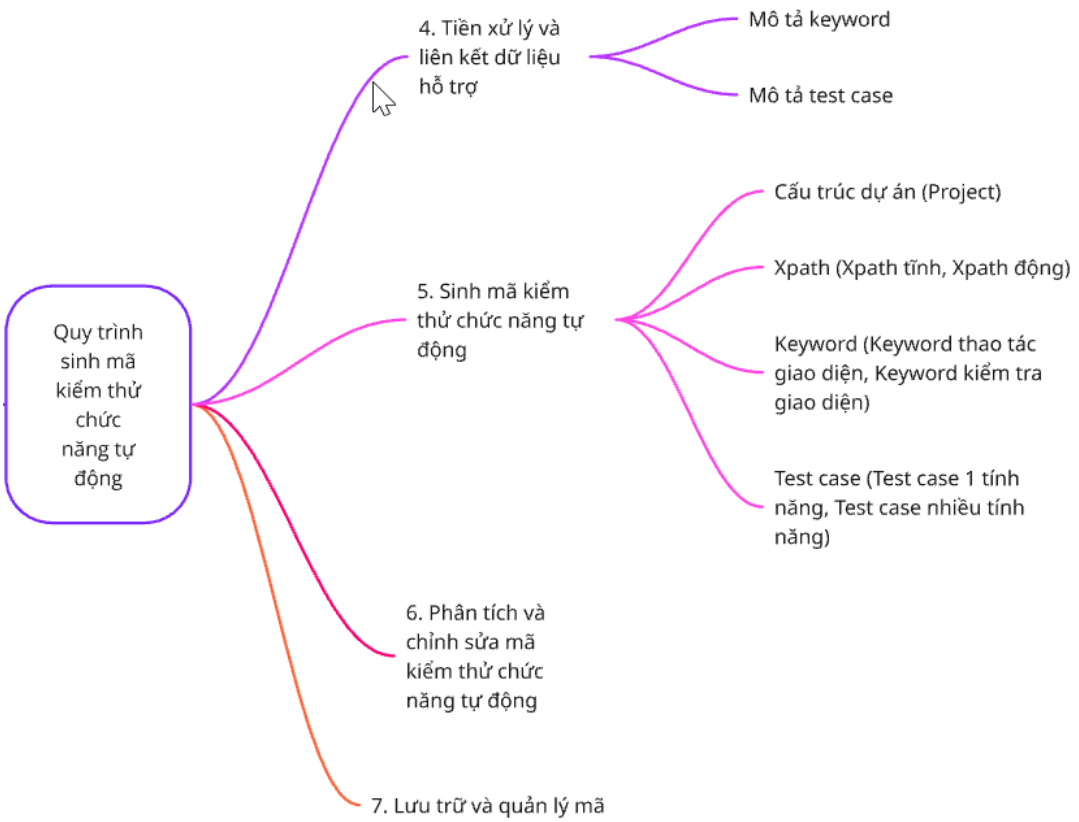
Hình 3.1. Quy trình sinh mã kiểm thử chức năng tự động sử dụng LLM (1)

Quy trình sinh mã kiểm thử chức năng tự động sử dụng LLM được triển khai theo bảy thành phần liên tiếp từ thu thập đầu vào, phân loại đầu vào, xây dựng prompt, sinh mã, tiền xử lý và liên kết dữ liệu hỗ trợ, phân tích và chỉnh sửa mã đến quản lý và lưu trữ kết quả.

Thành phần đầu tiên tập trung vào việc thu thập đầu vào cho mô hình LLM, bao gồm các tệp HTML của giao diện web và tập test case thủ công. HTML được trích xuất trực tiếp từ DOM của từng màn hình và lưu trữ trong một thư mục, test case được gom nhóm thành hai dạng: các bước keyword lặp lại (keyword) và luồng nghiệp vụ đầy đủ (test case). Việc thu thập đầy đủ và lưu trữ tập trung dữ liệu đầu vào giúp xây dựng một nền tảng dữ liệu hoàn chỉnh và nhất quán cho các bước tiếp theo.

Tiếp theo, toàn bộ dữ liệu thu thập được phân loại thành các nhóm đặc trưng nhằm giảm mức độ phức tạp và tạo điều kiện cho mô hình LLM xử lý hiệu quả hơn. Trong giai đoạn này, XPath được phân loại thành XPath tĩnh và XPath động, keyword được chia thành keyword tương tác giao diện và keyword kiểm tra giao diện, còn test case được tách thành nhóm một tính năng và nhiều tính năng. Việc phân loại này giúp mô hình nhận diện đúng bản chất của từng loại dữ liệu và tạo nền tảng cho việc thiết kế prompt phù hợp.

Dựa trên kết quả phân loại, hệ thống prompt mẫu được xây dựng theo hai hướng: prompt theo tầng và prompt theo nhóm. Prompt theo tầng định nghĩa cơ chế sinh mã theo các lớp Project - XPath - Keyword - Test Case, còn prompt theo nhóm mô tả cụ thể cách sinh mã cho từng loại dữ liệu như XPath tĩnh, XPath động, keyword thao tác, keyword kiểm tra, test case một hoặc nhiều tính năng. Thành phần này đóng vai trò then chốt trong việc hướng dẫn mô hình LLM sinh mã theo đúng quy tắc, định dạng và cấu trúc dự án.



Hình 3.2. Quy trình sinh mã kiểm thử chức năng tự động sử dụng LLM (2)

Thành phần tiền xử lý và liên kết dữ liệu hỗ trợ xử lý mô tả test case và keyword, tách rõ ràng hành động, dữ liệu và phần tử giao diện, đồng thời đánh dấu chúng bằng các ký hiệu chuẩn nhằm giúp mô hình LLM hiểu chính xác từng thành phần, ánh xạ giữa các bước kiểm thử với XPath, keyword và HTML tương ứng. Hoạt động tiền xử lý giúp chuyển đổi mô tả ngôn ngữ tự nhiên thành dữ liệu có cấu trúc, giảm mơ hồ và tăng độ chính xác trong quá trình sinh mã

Dựa trên HTML, test case, keyword đã xử lý và bộ prompt mẫu, mô hình LLM thực hiện sinh mã kiểm thử theo từng tầng, bao gồm sinh cấu trúc dự án, sinh XPath tĩnh, Xpath động, sinh keyword thao tác giao diện, keyword kiểm tra giao diện, test case 1 tính năng và test case nhiều tính năng. Mã được sinh theo cấu trúc của dự án Robot Framework, đảm bảo tính tách biệt giữa mã giao diện, mã thao tác và mã luồng

nghiệp vụ.

Sau khi mô hình sinh mã kiểm thử chức năng, kiểm thử viên sẽ kiểm tra và so sánh mã với nghiệp vụ thực tế nhằm phát hiện các lỗi ví dụ như sai tham số, keyword không phù hợp, thiếu bước hoặc thừa bước. Các lỗi này được cung cấp ngược lại cho mô hình dưới dạng ví dụ đúng - sai để mô hình chỉnh sửa lại theo cơ chế few-shot. Nhờ đó, mã kiểm thử chức năng được hoàn thiện dần và đạt độ chính xác cần thiết để thực thi

Thành phần cuối cùng đảm bảo mã sinh ra được lưu trữ đúng chuẩn cấu trúc dự án, với XPath được lưu trong thư mục Repository, keyword trong Resource và test script trong TestCases. Mã được sinh dưới dạng xem trước (preview) để kiểm thử viên xác nhận trước khi lưu nhằm tránh ghi đè ngoài ý muốn. Việc tổ chức mã theo cấu trúc thống nhất giúp dễ dàng truy xuất, bảo trì và tái sử dụng trong các lần sinh mã tiếp theo, hoàn thiện toàn bộ quy trình sinh mã kiểm thử tự động

KẾT LUẬN CHƯƠNG

Chương III học viên đã trình bày quy trình sinh mã kiểm thử chức năng tự động sử dụng LLM, bắt đầu từ thu thập và phân loại đầu vào, xây dựng prompt, tiền xử lý và liên kết dữ liệu hỗ trợ, sử dụng mô hình LLM để sinh mã kiểm thử chức năng, phân tích chỉnh sửa mã, và cuối cùng là lưu trữ, quản lý mã sinh ra

CHƯƠNG IV: THỰC NGHIỆM VÀ ĐÁNH GIÁ TRÊN HỆ THỐNG VNEDU

Chương IV trình bày thực nghiệm và đánh giá quy trình sinh mã kiểm thử chức năng tự động sử dụng mô hình LLM trên hệ thống VNEDU sau một thời gian triển khai.

4.1. Thực nghiệm quy trình trên hệ thống VNEDU:

4.1.1. Thu thập đầu vào:

Gồm 2 loại đầu vào là các bước kiểm thử thủ công và các file HTML chứa cấu trúc DOM giao diện của hệ thống VNEDU

Dữ liệu các bước kiểm thử thủ công được trích xuất từ bộ test case đang sử dụng tại Trung tâm giải pháp Giáo dục số – VNPT IT. Dựa trên các nhóm chức năng phân tích ở chương 1.3 bộ test case được chia thành các nhóm test case có đặc trưng riêng

- Nhóm test case CRUD: Đây là nhóm chiếm tỷ lệ lớn nhất đặc trưng của nhóm test case này là có nhiều bước nhập liệu, chọn giá trị từ danh sách, nhấn nút xác nhận và kiểm tra phản hồi của hệ thống sau mỗi thao tác, là cơ sở để xây dựng bộ từ khóa thao tác giao diện phong phú (nhập, chọn, click, lưu, hủy bỏ) và tập hợp XPath đa dạng từ các trường dữ liệu đến các nút chức năng.
- Nhóm test case tra cứu: Đặc trưng của test case tra cứu là thao tác chủ yếu trên bảng dữ liệu không cố định, bao gồm nhập điều kiện lọc, chọn điều kiện tìm kiếm, chọn dòng, xem chi tiết. Trong góc độ sinh mã kiểm thử chức năng tự động, nhóm này chứa nhiều XPath động, đặc biệt là phần tử trong bảng có chứa các hàng không cố định, phần tử xuất hiện sau thao tác lọc.
- Nhóm test case thống kê báo cáo: Đặc trưng của nhóm này là giao diện thường có cấu trúc bảng cố định, với các cột và hàng được định nghĩa sẵn theo từng loại báo cáo. Vì cấu trúc cột và hàng là cố định, test case thuộc nhóm này ưu tiên các thao tác kiểm tra như kiểm tra đúng tiêu đề cột, đúng số dòng dữ liệu, đúng giá trị trong từng ô.
- Nhóm test case cấu hình hệ thống: Đặc trưng là số bước thao tác ngắn, ít

phức tạp, nhưng yêu cầu tính chính xác cao vì sai cấu hình có thể gây lỗi dây chuyền. Trong kiểm thử tự động, nhóm này mang lại bộ dữ liệu ổn định để kiểm tra khả năng sinh mã với các phần tử giao diện tĩnh, các dropdown cố định, và các thao tác xác nhận thay đổi.

● Nhóm test case phân quyền và quản lý tài khoản: Đặc trưng cần kiểm tra chính xác hành vi được phép/không được phép, hiển thị hay không hiển thị các nút, menu và chức năng đòi hỏi mô hình phải sinh được XPath chính xác tại các vị trí có điều kiện hiển thị khác nhau và sinh được các bước kiểm tra phù hợp với trạng thái giao diện.

● Nhóm test case nhập và xuất dữ liệu: Đặc trưng của nhóm này là thao tác gắn liền với xử lý tệp, người dùng cần chọn tệp đầu vào, kiểm tra định dạng, xác nhận nội dung tệp xuất ra khớp với dữ liệu hiển thị trên hệ thống đòi hỏi xử lý tệp ở mức hệ thống (đường dẫn tuyệt đối của tệp đầu vào, kiểm tra nội dung file Excel hoặc PDF của tệp xuất ra,..).

Việc lựa chọn test case chức năng cho thực nghiệm là bước quan trọng nhằm đảm bảo mô hình LLM được đánh giá trên một tập dữ liệu đại diện đầy đủ cho đặc trưng nghiệp vụ của hệ thống VNEDU. Vì vậy, bộ test case cần phản ánh đầy đủ đặc trưng về nghiệp vụ, phần tử giao diện (được thể hiện qua XPath), thao tác (được thể hiện qua keyword) để đảm bảo mô hình LLM được đánh giá toàn diện dựa trên một số nguyên tắc như sau:

● Nguyên tắc 1: Bao phủ các nhóm tính năng của hệ thống: bao phủ nhóm chức năng có tần suất sử dụng cao, ảnh hưởng trực tiếp đến hoạt động nghiệp vụ hằng ngày như CRUD

● Nguyên tắc 2: Bao phủ các nhóm keyword: Test case được chọn phải thể hiện đầy đủ các dạng thao tác giao diện phổ biến như nhập liệu, chọn giá trị trong dropdown, click nút chức năng,... Các bước kiểm tra phải bao gồm cả kiểm tra hiển thị, kiểm tra thông báo, kiểm tra quyền truy cập theo vai trò. Điều này đảm bảo mô hình LLM được đánh giá trên cả hai nhóm keyword quan trọng: keyword thao tác và keyword kiểm tra.

● Nguyên tắc 3: Bao phủ các nhóm xpath: Bao phủ các dạng phần tử như trường nhập liệu, combobox, nút chức năng, bảng dữ liệu cố định,... để mô hình LLM có thể được đánh giá khả năng sinh cả XPath tĩnh và XPath động.

● Nguyên tắc 4: Bao phủ cả test case một tính năng và test case nhiều tính năng nhằm đánh giá khả năng mô hình LLM sinh mã kiểm thử chức năng từ những thao tác đơn giản trong phạm vi một tính năng đến những luồng nghiệp vụ liên thông và phức tạp trong thực tế vận hành của hệ thống VNEDU

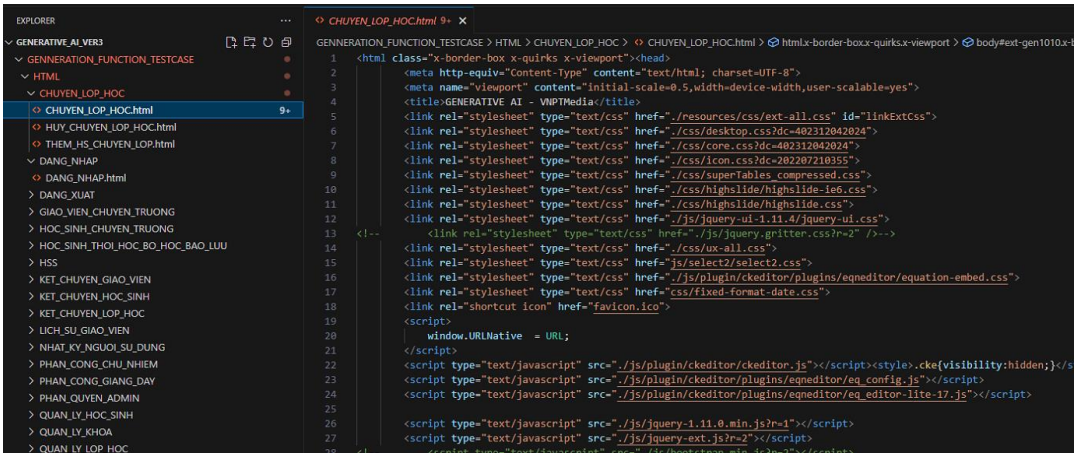
● Nguyên tắc 5: Lựa chọn test case cần ưu tiên những phân hệ chức năng có tần suất sử dụng cao, ảnh hưởng lớn đến quá trình học tập, quản lý của nhà trường hoặc có độ nhạy cao với dữ liệu (học sinh, nhân sự, điểm số,...).

Từ phân loại và nguyên tắc lựa chọn test case trên, học viên áp dụng quy trình sinh mã kiểm thử chức năng tự động cho toàn bộ 150 test case trong đó lựa chọn 75 test case đại diện từ năm nhóm (CRUD, tra cứu, thống kê báo cáo, cấu hình hệ thống và phân quyền người dùng) để mô tả, phân tích và đánh giá chi tiết trong đề án. 75 test case đại diện chia ra làm 2 nhóm test case 36 test case một tính năng và 39 test case nhiều tính năng, trong đó mỗi test case chứa từ 5 - 50 Xpath bao gồm cả Xpath tĩnh và Xpath động, 4 -35 keyword bao gồm cả keyword thao tác giao diện và keyword kiểm tra giao diện. Riêng nhóm nhập/xuất dữ liệu không được đưa vào phạm vi thực nghiệm vì phần lớn thao tác của nhóm này phụ thuộc vào xử lý tệp ở cấp hệ điều hành, không phải dạng test case điển hình mà mô hình LLM cần sinh mã dựa trên giao diện và mô tả các bước kiểm thử.

Bảng 4.1. Số lượng test case thử nghiệm

Nhóm chức năng	Số lượng test case	Tỷ lệ
CRUD	15	20%
Tra cứu	15	20%
Thống kê Báo cáo	15	20%
Cấu hình hệ thống	15	20%
Phân quyền và quản lý người dùng	15	20%

Ngoài dữ liệu các bước kiểm thử thủ công được trích xuất từ test case, đề án còn sử dụng thêm tập dữ liệu HTML chứa giao diện của từng màn hình trong hệ thống được thể hiện dưới dạng cây DOM. Các tệp HTML này được kiểm thử viên thu thập trực tiếp từ giao diện hệ thống VNEDU bằng cách truy cập vào từng màn hình và sử dụng công cụ Kiểm tra (F12) của trình duyệt để lấy dữ liệu của toàn bộ cây DOM.



Hình 4.1. Ví dụ tệp HTML đầu vào

4.1.2. Phân loại đầu vào:

Từ tổng số 75 test case đại diện đã được lựa chọn theo các nguyên tắc, đề án tiếp tục chọn ra 5 test case. Năm test case này được lựa chọn nguyên tắc được phân tích trong chương 4.1.1.

Bảng 4.2. Testcase đại diện theo nhóm chức năng hệ thống VNEDU

Mã	**Tên test case**	**Loại test case**
01	Thêm nhân sự thành công	Test case một tính năng
02	Tra cứu hồ sơ học sinh	Test case nhiều tính năng
03	Thống kê tỷ lệ học sinh nam, nữ	Test case một tính năng
04	Cấu hình sổ điểm	Test case nhiều tính năng
05	Phân công giảng dạy cấp THCS (Trung học cơ sở)	Test case nhiều tính năng

Từ bộ 5 test case đại diện đã được lựa chọn, kiểm thử viên tiến hành phân tích chi tiết thông tin đầu vào bằng cách phân loại dữ liệu của từng bước kiểm thử theo thông tin về XPath (bao gồm XPath tĩnh, XPath động), thông tin Keyword (bao gồm keyword thao tác giao diện và keyword kiểm tra giao diện), ngoài ra các thao tác lặp lại ở các test case cùng được phân tích thành các keyword dùng chung.

Bảng 4.3. Ví dụ phân loại Xpath và Keyword test case Thêm nhân sự thành công

Mã	**Tên test case**	**Thông tin Xpath**	**Thông tin Keyword**
01	Thêm nhân sự thành công	**- Xpath tĩnh** + button Thêm mới + textbox Họ và tên: + textbox Ngày sinh: + button Lưu + button Yes + button OK + ô chứa placeholder Tìm kiếm theo tên, tài khoản của nhân sự… **- Xpath động** + ô cột thứ 2 chứa giá trị Họ và tên	**- Keyword dùng chung:** 1: Đăng nhập thành công 2: Vào chức năng Quản lý nhân sự **- Keyword thao tác giao diện** 3. Click button Thêm mới 4. Nhập giá trị Họ và tên vào ô Họ và tên: 5. Nhập giá trị Ngày sinh vào ô Ngày sinh: 6. Click button Lưu 7. Click button Yes 8. Click OK **- Keyword kiểm tra giao diện** 9. Kiểm tra ô cột thứ 2 chứa giá trị Họ và tên

4.1.3. Xây dựng prompt mẫu:

Dựa trên cơ chế prompt theo tầng và theo loại, đề án đã tiến hành xây dựng bộ prompt mẫu gồm 7 mẫu nhằm hỗ trợ mô hình LLM xử lý tuần tự các loại dữ liệu đầu vào và sinh ra mã kiểm thử chức năng theo từng loại dữ liệu.

Bảng 4.4. Prompt mẫu

Loại Prompt	**Nội dung Prompt**
Prompt Project	Tạo một project với tên là <TÊN Project> sử dụng RobotFramework Quy tắc chung : Các file trong project đều được tạo với đuôi .robot và được encode UTF8 Các thư viện cần import trong các file robot: Library SeleniumLibrary Library ExcelLibrary Library String Library DateTime Library Process Các thành phần của project: 1. 1 file bat được viết để chạy hết các testcase trong folder Testcase và ghi kết quả chạy vào 1 file 2. 1 Folder Testcases để lưu các testcase. VD: Testcases_01.robot có 3 phần: Phần 1 là phần Settings chứa các thư viện và resource, Phần 2 Variable khai báo các biến, Phần 3 Testcase chứa các testcase lưu dưới dạng keyword ví dụ testcase Testcases_01. Các phần ngăn cách nhau bởi cấu trúc chuẩn của RobotFramework về khai báo thư viện Settings, khai báo biến Variable 3. 1 Folder Resource để lưu các hàm, thư viện được tự viết cho các chức năng. VD: Resource.robot có 3 phần: Phần 1 là phần Settings chứa các thư viện và resource, Phần 2 Variable khai báo các biến, Phần 3 Keywords chứa các Keyword lưu dưới dạng keyword ví dụ keyword Keyword_01.robot. Các phần ngăn cách nhau bởi cấu trúc chuẩn của RobotFramework về khai báo thư viện Settings, khai báo biến Variable, và khai báo keyword dưới dạng Keywords

	4. 1 Folder Repository để lưu các Variable Xpath Locator VD: XPATH_SAMPLE.robot có 2 phần: Phần 1 là phần Settings chứa các thư viện và resource, Phần 2 Variable khai báo các biến. Các phần ngăn cách nhau bởi cấu trúc chuẩn của RobotFramework về khai báo thư viện Settings, khai báo biến Variable 5. 1 thư mục HTML được lưu trữ html của trang web. VD HTML_template.html 6. 1 thư mục STEP_TESTCASE để lưu trữ các bước của testcase manual. VD Testcase_manual.txt 7. 1 thư mục STEP_KEYWORD để lưu trữ các bước của Keyword. VD Keyword_sample.txt 8. 1 thư mục results để lưu trữ kết quả chạy các testcases
Prompt Xpath tĩnh	Đọc file .html thư mực HTML và file .txt thư mục STEP_KEYWORD. Tạo một file XPATH_<Tên File>.robot mới tại thư mục Repository có <Tên File> giống với tên file .html Với mỗi phần tử trong [] trong file .txt, xác định đúng element trong HTML và sinh một XPath theo quy tắc: - Sinh Xpath cho các phần tử thuộc các dòng không chứa () - Dựa vào thẻ label, button, a, span, div thể hiện các element dạng label hoặc text, input, textarea thể hiện cho các element dạng nhập; select thể hiện cho các element dạng dropdown, ul li thể hiện cho element dạng list, tr thể hiện cho các element dạng hàng, th thể hiện cho các element dạng cột. - Dựa vào anchor ổn định như text, label hoặc dùng /.. để đi lên cha gần nhất. - Không được dùng XPath tuyệt đối (/html/body/...). - XPath tĩnh không được chứa tham số dạng ${}

	Với mỗi Xpath tạo biến với định dạng ${[ten_the_html]_[noi_dung_text_khong_dau_cach_bang_gach_duoi]}. Ghi toàn bộ các biến XPath vừa tạo vào file XPATH_<TEN_FILE>.robot.
Prompt Xpath động	Đọc file .html thư mực HTML và file .txt thư mục STEP_KEYWORD. Tạo một file XPATH_<Tên File>.robot mới tại thư mục Repository có <Tên File> giống với tên file .html Với mỗi phần tử trong [] trong file .txt, xác định đúng element trong HTML và sinh một XPath theo quy tắc: - Sinh Xpath cho các phần tử thuộc các dòng chứa () - Dựa vào thẻ label, button, a, span, div thể hiện các element dạng label hoặc text, input, textarea thể hiện cho các element dạng nhập; select thể hiện cho các element dạng dropdown, ul li thể hiện cho element dạng list, tr thể hiện cho các element dạng hàng, th thể hiện cho các element dạng cột. - Dựa vào anchor ổn định như text, label hoặc dùng /.. để đi lên cha gần nhất. - Không được dùng XPath tuyệt đối /html/body/... - Dấu () xác định tham số của phần tử [] ví dụ [cột chứa giá trị (a)] được thể hiện dưới dạng //th[contains(normalize-space(.),'${a}')] Với mỗi Xpath tạo biến với định dạng ${[ten_the_html]_[noi_dung_text_khong_dau_cach_bang_gach_duoi]}. Ghi toàn bộ các biến XPath vừa tạo vào file XPATH_<TEN_FILE>.robot.

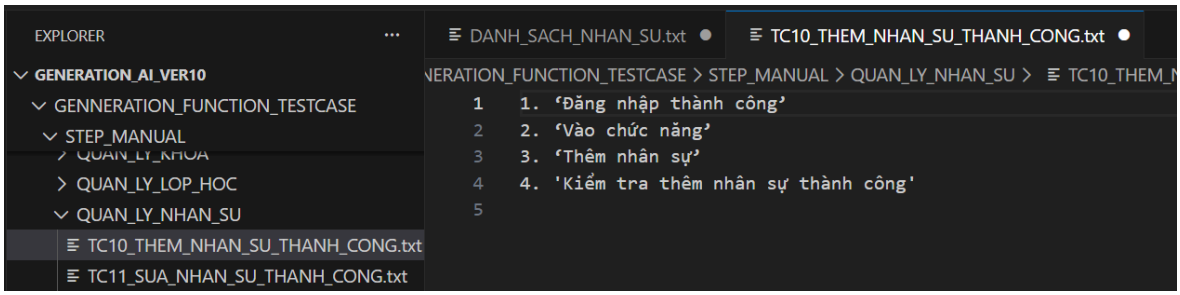
Prompt Keyword thao tác với giao diện	Đọc file .txt trong folder STEP_KEYWORD Đọc các file XPATH_<TEN_FILE>.robot trong folder Repository Tạo file <TÊN FILE Keyword>.robot folder Resource. Khai báo Resource để sử dụng các biến và hàm trong các file Repository ví dụ Resource XPATH_<TEN_FILE>.robot. Trong file <TÊN FILE Keyword>.robot folder Resource sinh keyword từ keyword bán chuẩn được ghi trong file <TÊN FILE Keyword>.txt trong đó gọi đến các file <TÊN REPOSITORY> để lấy các xpath là giá trị của biến phù hợp nhất với các nội dung trong [] của từng bước theo quy tắc: - Chỉ sinh keyword cho các bước KHÔNG chứa từ khóa kiểm tra như “kiểm tra”, “verify”, “so sánh”, “assert”, “hiển thị” hoặc các dạng mô tả kết quả mong đợi. - Nếu phần tử [] có chứa tham số trong dấu (), tham số này là giá trị động của chính phần tử được mô tả trong [] và phải được đưa vào câu lệnh Robot Framework dưới dạng argument. - Sử dụng các keyword có sẵn như Click Element, Input Text, Select From List, Press Keys… tương ứng với loại thao tác của từng bước - Mỗi keyword bán chuẩn kết thúc bởi dấu.
Prompt Keyword kiểm tra giao diện	Đọc file .txt trong folder STEP_KEYWORD Đọc các file XPATH_<TEN_FILE>.robot trong folder Repository Tạo file <TÊN FILE Keyword>.robot folder Resource. Khai báo Resource để sử dụng các biến và hàm trong các file Repository ví dụ Resource XPATH_<TEN_FILE>.robot. Trong file <TÊN FILE Keyword>.robot folder Resource sinh keyword từ keyword bán chuẩn được ghi trong file <TÊN FILE Keyword>.txt trong đó gọi đến các file <TÊN REPOSITORY> để lấy các xpath là giá trị của biến phù hợp nhất với các nội dung trong []

	của từng bước theo quy tắc: - Chỉ sinh keyword cho các bước CHỨA từ khóa kiểm tra như “kiểm tra”, “verify”, “so sánh”, “assert”, “hiển thị” hoặc các dạng mô tả kết quả mong đợi. - Nếu bước có chứa tham số trong dấu (), tham số này là giá trị động của chính phần tử được mô tả trong [] và phải được đưa vào câu lệnh Robot Framework dưới dạng argument. - Sử dụng các keyword có sẵn như Click Element, Input Text, Select From List, Press Keys… tương ứng với loại thao tác của từng bước - Mỗi keyword bán chuẩn kết thúc bởi dấu.
Prompt test case một tính năng	Đọc file .txt trong folder STEP_MANUAL Đọc các keyword trong file .robot folder Resource Khai báo Resource để sử dụng các biến và hàm trong file .robot folder Resource. Nếu đã khai báo rồi thì không cần khai báo lại Khai báo đầy đủ các biến để sử dụng các biến và hàm trong file Resource.robot. Tạo file robot <TÊN FILE TESTCASE>.robot để thực hiện các bước được ghi trong file <TÊN FILE TESTCASE>.txt trong đó gọi đến file .robot folder Resource để lấy các keyword phù hợp nhất với các nội dung trong " theo quy tắc: - Chỉ dùng các lệnh/bước có trong file testcase. Không sinh biến tạm, keyword mới
Prompt test case nhiều tính năng	Đọc file .txt trong folder STEP_MANUAL Đọc các keyword trong file .robot folder Resource Khai báo Resource để sử dụng các biến và hàm trong file .robot folder Resource. Nếu đã khai báo rồi thì không cần khai báo lại Tạo file robot <TÊN FILE TESTCASE>.robot để thực hiện các bước

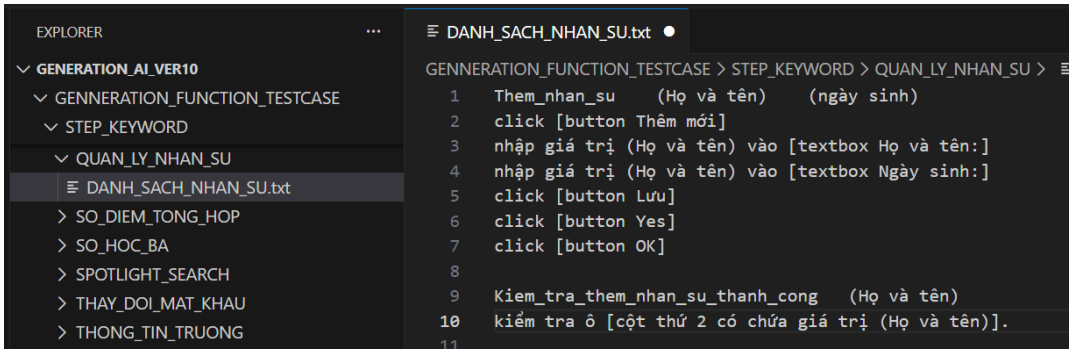
	được ghi trong file <TÊN FILE TESTCASE>.txt trong đó gọi đến file .robot folder Resource để lấy các keyword phù hợp nhất với các nội dung trong " theo quy tắc: - Chỉ dùng các lệnh/bước có trong file testcase. Không sinh biến tạm, keyword mới - Khi đang gọi keyword thuộc file .robot A và chuyển sang gọi keyword thuộc file .robot B tự động chèn thêm một bước chờ 3 giây trước khi gọi keyword tiếp theo

4.1.4. Tiền xử lý và liên kết dữ liệu hỗ trợ:

Sau khi tiền xử lý test case dựa trên các quy tắc đã phân tích trong chương 3.4, đầu vào cho mô hình LLM sinh mà kiểm thử chức năng là test case được chia nhỏ thành hai phần, phần một các các keyword dùng chung được sắp xếp theo thứ tự luồng nghiệp vụ và được lưu trong thư mục STEP_TESTCASE, thành phần hai chứa các keyword dưới dạng bán chuẩn là các bước kiểm thử được gom nhóm lại để dùng trong nhiều test case và được lưu trong thư mục STEP_KEYWORD



Hình 4.2. Test case Thêm nhân sự thành công sau xử lý



Hình 4.3. Keyword của test case Thêm nhân sự thành công sau xử lý

4.1.5. Sinh mã kiểm thử chức năng tự động:

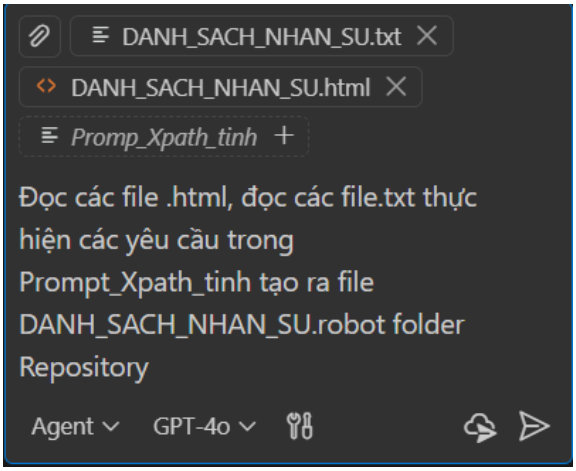
Trong thực nghiệm, quy trình sinh mã kiểm thử chức năng tự động được triển khai bằng mô hình LLM GPT-4o, mô hình được tích hợp trong công cụ GitHub Copilot Chat. GitHub Copilot Chat được sử dụng trực tiếp trong môi trường phát triển Visual Studio Code của dự án Robot Framework, cho phép mô hình LLM truy cập các tệp ngữ cảnh (context files) và các prompt và thực hiện sinh mã dựa trên cấu trúc thực tế của dự án. Thực nghiệm triển khai sinh cấu trúc dự án và 6 loại dữ liệu, sử dụng các prompt và các tệp ngữ cảnh làm đầu vào cho mô hình LLM sinh mã kiểm thử chức năng tự động cho kết quả tương ứng với 6 loại dữ liệu được trình bày trong bảng sau:

Bảng 4.5. Sinh mã cho các loại dữ liệu

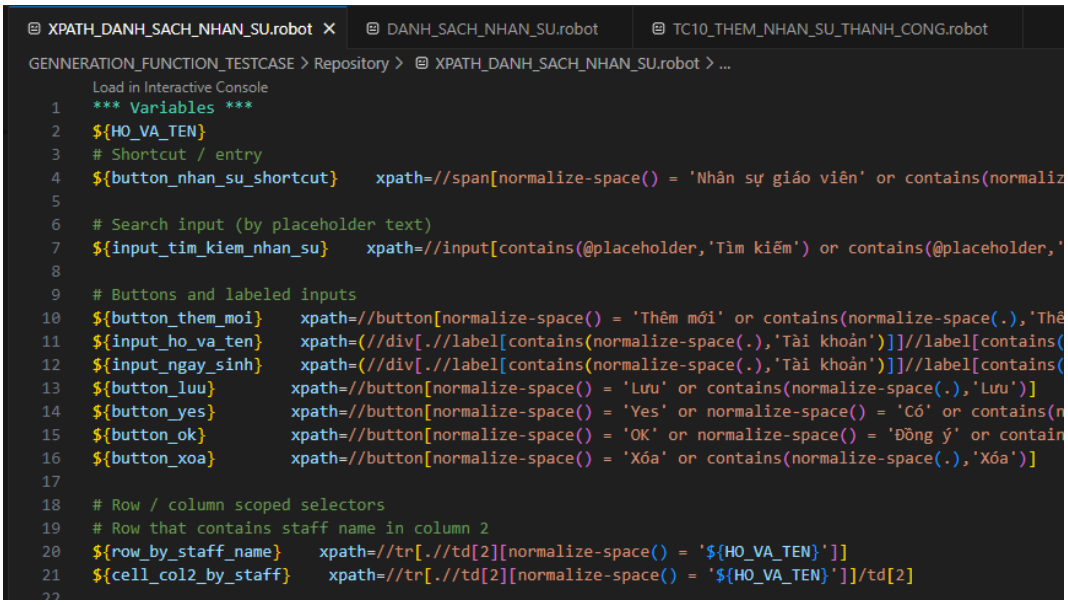
Loại dữ liệu cần sinh mã	**Prompt sử dụng**	**Tệp ngữ cảnh đi kèm**	**Kết quả sinh mã**
Project	Prompt Project		Cấu trúc dự án gồm: - Thư mục STEP_TESTCASE - Thư mục STEP_KEYWORD - Thư mục HTML - Thư mục Repository - Thư mục Resource - Thư mục Testcases - File run_test.bat

			- Thư mục results
XPath tĩnh	Prompt Xpath tĩnh	- Các file .html thư mục HTML - Các file .txt thư mục STEP_KEYWORD	Xpath tĩnh tự động sinh ra và lưu vào tệp .robot thư mục Repository dưới dạng chờ xác nhận
Xpath động	Prompt Xpath động	- Các file .html thư mục HTML - Các file .txt thư mục STEP_KEYWORD	Xpath động tự động sinh ra và lưu vào tệp .robot thư mục Repository dưới dạng chờ xác nhận
Keyword thao tác giao diện	Prompt Keyword thao tác giao diện	- Các tệp .robot thư mục Repository - Các file .txt thư mục STEP_KEYWORD	Keyword thao tác giao diện tự động sinh ra và lưu vào tệp. robot thư mục Resource dưới dạng chờ xác nhận
Keyword kiểm tra giao diện	Prompt Keyword kiểm tra giao diện	- Các tệp .robot thư mục Repository - Các file .txt thư mục STEP_KEYWORD	Keyword thao tác giao diện tự động sinh ra và lưu vào tệp. robot thư mục Resource dưới dạng chờ xác nhận
Test case một tính năng	Prompt Test case một tính năng	- Các tệp .robot thư mục Repository - Các tệp .robot thư mục Resource	Test script tương ứng với test case một tính năng tự động sinh ra và lưu vào tệp .robot thư mục Testcases dưới dạng chờ xác nhận
Test case nhiều tính năng	Prompt Test case nhiều tính năng	- Các tệp .robot thư mục Repository - Các tệp .robot thư mục Resource	Test script tương ứng với test case nhiều tính năng tự động sinh ra và lưu vào tệp .robot thư mục Testcases dưới dạng chờ xác nhận

Ví dụ sử dụng mô hình LLM của Github Copilot Chat trong Visual Studio để sinh mã cho loại dữ liệu XPath tĩnh. Ngữ cảnh đầu vào của mô hình LLM sẽ được truyền vào thông qua giao diện chat bao gồm các tệp .html chứa cấu trúc giao diện, các tệp .txt chứa các keyword bán chuẩn, sau đó mô hình LLM thực hiện các yêu cầu và quy tắc trong Prompt Xpath tĩnh để tạo đầu ra là tập biến XPath tĩnh lưu trong các tệp .robot thư mục Repository



Hình 4.4. Ví dụ sinh mã cho loại dữ liệu Xpath tĩnh



Hình 4.5. Ví dụ sinh Xpath cho test case Thêm nhân sự thành công

```
Load in Interactive Console
Them_nhan_su
    [Arguments]    ${ten_nhan_su}    ${ngay_sinh}
    Click Element    ${button_them_moi}
    Input Text    ${input_ho_va_ten}    ${ten_nhan_su}
    Input Text    ${input_ngay_sinh}    ${ngay_sinh}
    Click Element    ${button_luu}
    Click Element    ${button_yes}
    Click Element    ${button_ok}
Load in Interactive Console
Kiem_tra_them_nhan_su_thanh_cong
    [Arguments]    ${ten_nhan_su}
    # Verify the staff appears in column 2
    Wait Until Element Is Visible    ${cell_col2_by_staff}    10s
```

Hình 4.6. Ví dụ sinh Keyword cho test case Thêm nhân sự thành công

```
Load in Interactive Console
*** Variables ***
${SITE}
${BROWSER}
${TEN_NHAN_SU}
${NGAY_SINH}
${USERNAME}
${PASSWORD}

*** Test Cases ***
Run | Debug | Run in Interactive Console
TC_Them_Nhan_Su_Thanh_Cong
    [Documentation]    Thực hiện các bước trong STEP_MANUAL/QUAN_LY_NHAN_SU/TC10_
    # 1) Đăng nhập thành công (use shared login keyword)
    Dang_nhap_thanh_cong    ${SITE}    ${BROWSER}    ${USERNAME}    ${PASSWORD}

    # 2) Vào chức năng Quản lý nhân sự
    Vao_chuc_nang_quan_ly_nhan_su

    # 3) Thêm mới nhân sự
    Them_nhan_su    ${TEN_NHAN_SU}    ${NGAY_SINH}

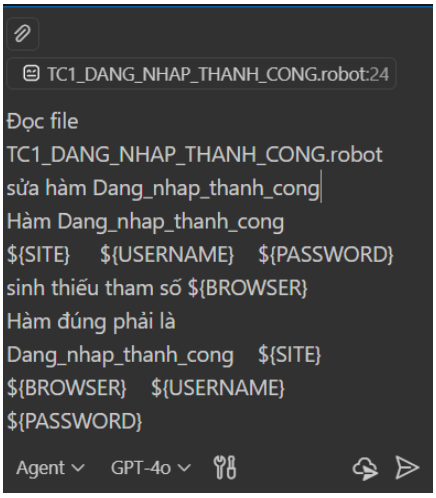
    # 4) Kiểm tra thêm nhân sự thành công
    Kiem_tra_them_nhan_su_thanh_cong    ${TEN_NHAN_SU}
```

Hình 4.7. Ví dụ sinh test case Thêm nhân sự thành công

Đối với dữ liệu kiểm thử, hệ thống không tự động sinh dữ liệu cụ thể mà chỉ sinh các biến khai báo, bởi vì giá trị dữ liệu phụ thuộc vào từng bối cảnh nghiệp vụ, từng môi trường triển khai và từng lần kiểm thử. Việc để kiểm thử viên tự nhập dữ liệu giúp đảm bảo dữ liệu phù hợp thực tế, tránh rủi ro sai lệch nghiệp vụ, đồng thời tăng khả năng tái sử dụng của test case cho nhiều phiên bản khác nhau.

4.1.6. Phân tích và chỉnh sửa mã:

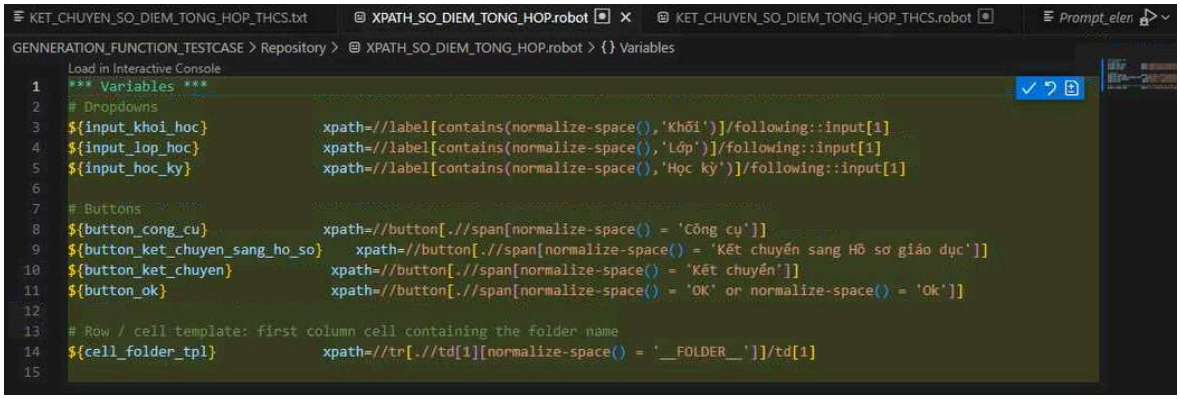
Người kiểm thử rà soát lại mã được tạo tự động bằng cách chạy thử test script để đảm bảo sự phù hợp với giao diện thực tế và luồng nghiệp vụ. Với file XPath, tester kiểm tra xem các XPath có trỏ đúng phần tử giao diện, có đủ tính ổn định khi giao diện thay đổi hay không. Với file keyword, tester kiểm tra logic thao tác xem đã đúng chức năng mô tả, thứ tự hành động có phản ánh đúng luồng nghiệp vụ. Với test case, tester kiểm tra sự đầy đủ của các bước và khả năng thực thi liên tục. Trong quá trình thực thi test script nếu xuất hiện lỗi runtime như XPath không trỏ đúng phần tử, keyword thiếu tham số, tester sẽ sử dụng trực tiếp giao diện chat của GitHub Copilot để hướng dẫn mô hình LLM sửa sai. Kiểm thử viên cung cấp cho mô hình cả dòng lỗi thực tế, nguyên nhân lỗi, và phiên bản đúng của đoạn mã cần thay thế. Nhờ cơ chế này, mô hình có thể cập nhật phần mã bị sai trong các tệp, đồng thời học được quy tắc sửa lỗi đúng từ cặp dữ liệu "lỗi sai – mã đúng". Điều này giúp giảm thiểu việc lặp lại lỗi ở các lần sinh mã tiếp theo và nâng cao độ chính xác của mô hình trong cùng một dự án kiểm thử.



Hình 4.8. Ví dụ phân tích và chỉnh sửa mã

4.1.7. Lưu trữ và quản lý mã:

Sau mô hình LLM sinh mã kiểm thử chức năng, các đoạn mã không lưu trực tiếp vào các tệp mà lưu dưới dạng chờ xác nhận, kiểm thử viên sẽ vào phân tích và chỉnh sửa mã sau đó mới lưu lại các mã.

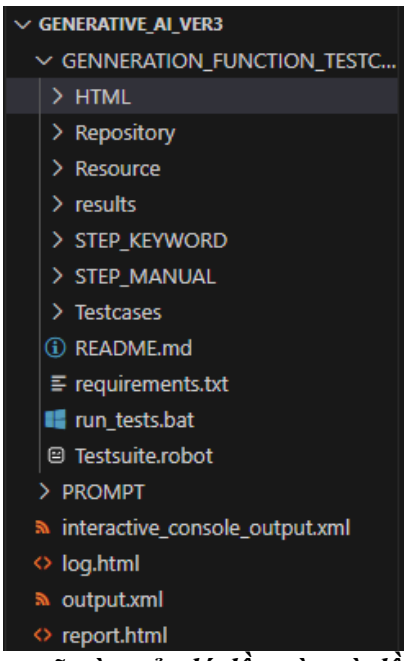


Hình 4.9. Xác nhận mã trước khi lưu



Hình 4.10. Xác nhận tệp trước khi lưu

Toàn bộ đầu vào và đầu ra của mô hình LLM được lưu trữ và quản lý có hệ thống nhằm đảm bảo khả năng truy xuất, bảo trì, tái sử dụng và đánh giá kết quả thực nghiệm dựa trên cấu trúc thư mục đã được mô hình LLM sinh ra ở tầng project



Hình 4.11. Cấu trúc lưu trữ và quản lý đầu vào và đầu ra của mô hình LLM

4.2. Phân tích kết quả thực nghiệm:

Thực nghiệm được triển khai trong khoảng thời gian từ tháng 09/2025 đến tháng 11/2025 trên môi trường thật của hệ thống VNEDU, sử dụng Visual Studio Code tích hợp GitHub Copilot Chat sử dụng mô hình LLM (Chat GPT 4o) và Robot Framework để sinh mã kiểm thử chức năng tự động cho 150 test case của hệ thống VNEDU trong đó lấy đại diện 75 test case từ năm nhóm tính năng chính (CRUD, tra cứu, thống kê báo cáo, cấu hình hệ thống và phân quyền người dùng) phân tích và đánh giá thực nghiệm.

Kết quả của quy trình sinh mã kiểm thử chức năng tự động sử dụng LLM là mã kiểm thử được sử dụng trực tiếp vào hoạt động kiểm thử chức năng tự động. Do đó để đảm bảo quy trình đề xuất mang lại hiệu quả thực tiễn, mã kiểm thử cần đáp ứng một số yêu cầu sau:

- Yêu cầu về độ chính xác (Accuracy): Mã kiểm thử sinh ra cần thực hiện đúng hành động được mô tả trong testcase, tương tác đúng phần tử giao diện tương ứng trong HTML, thứ tự các bước kiểm thử đúng với luồng nghiệp vụ
- Yêu cầu về độ bao phủ (Test Coverage): mã kiểm thử chứa đầy đủ các đối tượng giao diện, không bỏ sót các bước nghiệp vụ
- Yêu cầu về thời gian sinh mã (Generation Time): Mã kiểm thử sinh ra cần được tạo ra trong khoảng thời gian phù hợp để có thể áp dụng trong mô hình phát triển phần mềm nhanh như Agile/Scrum, đặc biệt khi cần tạo mới hoặc cập nhật nhanh bộ kiểm thử chức năng tự động hồi quy sau mỗi thay đổi.
- Yêu cầu về khả năng thực thi (Execution Time): Mã kiểm thử sinh ra cần có khả năng thực thi mà không phát sinh lỗi cú pháp khiến kịch bản không thể chạy
- Yêu cầu về thời gian thực thi (Pass Rate): Mã kiểm thử sinh ra cần có thời gian chạy hợp lý, tránh dư thừa thao tác lặp lại, hạn chế chờ không cần thiết

Trong đó, Accuracy và Test Coverage được kiểm thử viên trực tiếp đối chiếu giữa bước kiểm thử thủ công, giao diện thực tế và mã kiểm thử chức năng do mô hình sinh ra, sau đó tự đánh giá và ghi nhận vào file Excel. Hai chỉ số phụ thuộc vào mức độ hiểu đúng nghiệp vụ và khả năng chuyển hóa bước kiểm thử thủ công thành mã tự

động, vì vậy phương pháp đánh giá thủ công hiện vẫn là lựa chọn phù hợp và chính xác nhất.

$$Accuracy_{Xpath} = \frac{Số\ Xpath\ đúng}{Tổng\ số\ Xpath\ sinh\ ra} \times\ 100\%$$

$$Accuracy_{Keyword} = \frac{Số\ bước\ đúng}{Tổng\ số\ bước\ sinh\ ra} \times\ 100\%$$

$$Accuracy_{Testcase} = \frac{Số\ keyword\ đúng}{Tổng\ số\ keyword\ sinh\ ra} \times\ 100\%$$

$$Testcoverage_{Xpath} = \frac{Số\ Xpath\ sinh\ ra}{Tổng\ số\ Xpath\ phải\ sinh\ ra} \times\ 100\%$$

$$Testcoverage_{Keyword} = \frac{Số\ bước\ sinh\ ra}{Tổng\ số\ bước\ phải\ sinh\ ra} \times\ 100\%$$

$$Testcoverage_{Testcase} = \frac{Số\ keyword\ trong\ testcase\ sinh\ ra}{Tổng\ số\ keyword\ trong\ testcase\ phải\ sinh\ ra} \times\ 100\%$$

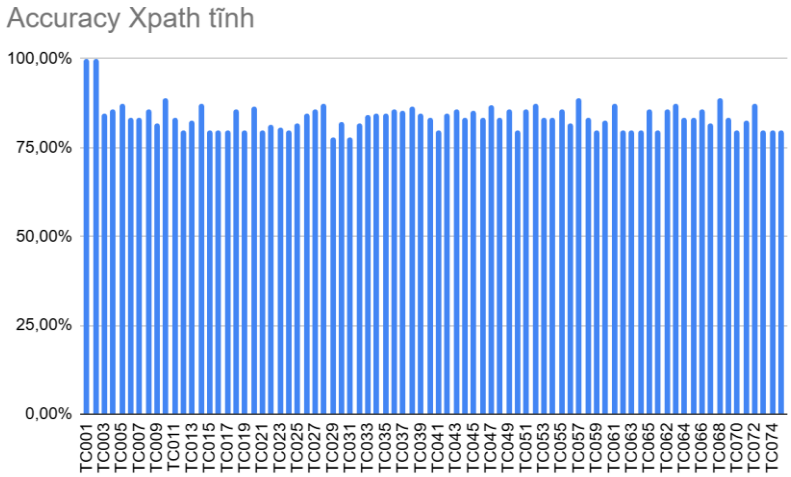
Chỉ số Generation Time được đo bằng công cụ mở rộng (extension) trong Visual Studio Code do kiểm thử viên tự viết, ghi nhận thời gian từ lúc kiểm thử viên bắt đầu nhập yêu cầu vào gửi yêu cầu cho mô hình LLM thông qua giao diện của Github Copilot Chat cho đến thời điểm một tệp trong dự án được tạo mới hoặc bị thay đổi bởi mô hình LLM. Hai chỉ số còn lại, Execution Time và Pass Rate, được lấy trực tiếp từ báo cáo kiểm thử của Robot Framework ghi nhận thời gian thực thi của từng test case và số lượng test case chạy thành công.

Quy trình sinh mã kiểm thử chức năng tự động sử dụng LLM được đánh giá ở 6 loại thực nghiệm tương ứng với 6 loại loại dữ liệu cần sinh mã đã trình bày trong

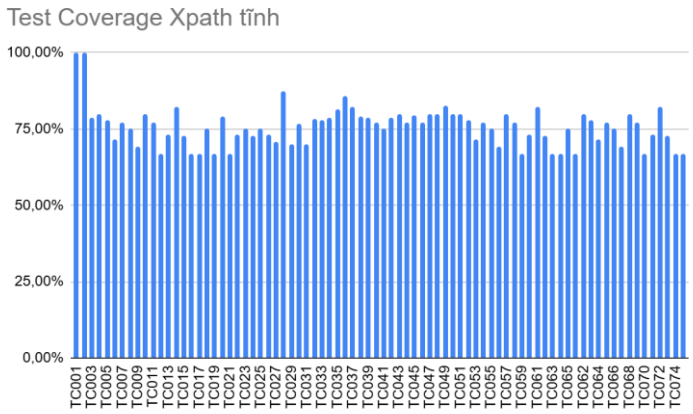
chương 4.1.5. Riêng loại sinh cấu trúc dự án không được đưa vào phạm vi đánh giá vì đây là phần có tính chất thiết lập ban đầu, chỉ được mô hình tạo một lần để khởi tạo cấu trúc thư mục của dự án Robot Framework:

Bảng 4.6. Các loại thực nghiệm

Loại thực nghiệm	**Đặc điểm thực nhiệm**
XPath tĩnh	Sinh XPath cố định từ cấu trúc DOM. Mô hình LLM xác định phần tử cần tạo XPath thông qua ký hiệu [] trong file STEP_KEYWORD. XPath được tạo hoàn toàn cố định, không dùng tham số
XPath động	Sinh XPath có tham số hóa bằng () cho các phần tử có giá trị thay đổi. Vẫn nhận diện phần tử từ [] nhưng mô hình LLM sinh XPath có chứa tham số động trong ()
Keyword thao tác giao diện	Keyword được sinh dựa trên nội dung các bước kiểm thử thủ công để xác định hành động cần thực hiện (click, nhập, chọn…). Mỗi keyword sẽ ánh xạ các phần tử trong [] sang biến XPath tương ứng trong Repository
Keyword kiểm tra giao diện	Sinh keyword kiểm tra trạng thái hoặc giá trị hiển thị trên giao diện (verify text, verify element visible…). Tập trung vào việc đối chiếu giao diện với kết quả mong đợi. Các phần tử trong [] được ánh xạ sang biến XPath tương ứng trong Repository
Test case tính chức năng	Sinh test case cho một tính năng. Sinh mã dựa trên các keyword trong một file tương ứng với một tính năng. Gồm các bước chạy tuần tự và không chèn bước chờ.
Test case nhiều tính năng	Sinh test case cho nhiều tính năng. Sinh mã dựa trên các keyword trong nhiều file tương ứng với nhiều tính năng. Gồm các bước chạy tuần tự và không chèn bước chờ, tự động thêm bước chờ 3s để ổn định giao diện

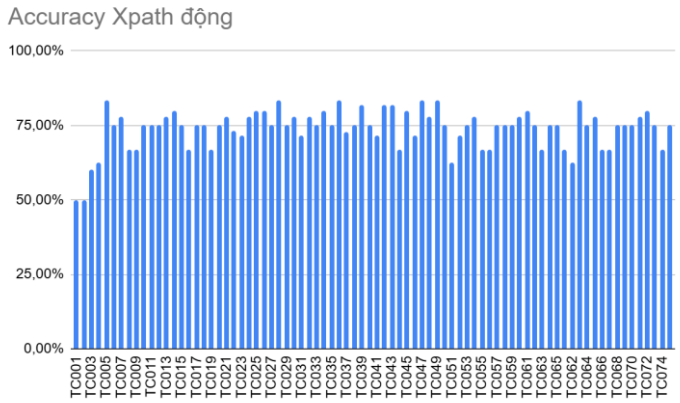


Hình 4.12. Accuracy Xpath Tĩnh

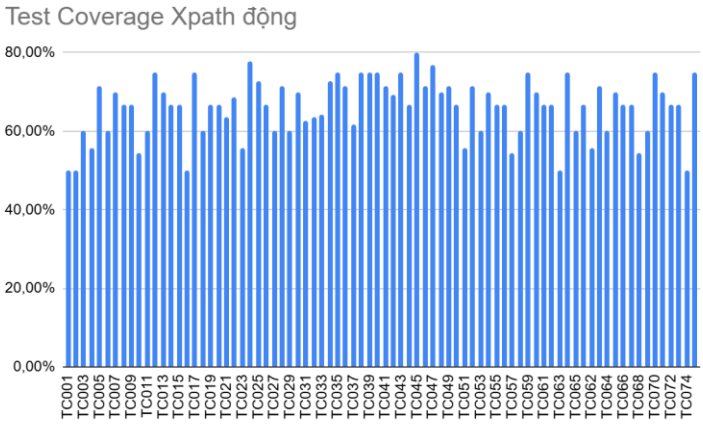


Hình 4.13. Test Coverage Xpath tĩnh

Kết quả thực nghiệm cho thấy việc sinh XPath tĩnh đạt mức Accuracy và Test Coverage tương đối cao. Accuracy của XPath tĩnh chủ yếu nằm trong khoảng 75–90%, một số test case như đạt 100%. Test Coverage của XPath tĩnh dao động ở mức 60–85%. Việc không sinh được mã hoặc sinh sai mã chủ yếu xuất phát từ các đặc thù của cấu trúc giao diện hệ thống VNEDU. Nhiều phần tử có các nhãn giống nhau nhưng lại nằm ở hai khung (panel) khác nhau, khiến mô hình sinh ra sinh XPath đúng cú pháp nhưng không trỏ đến một phần tử duy nhất trên giao diện. Các phần tử không chứa text hiển thị (như icon, hình ảnh) hoặc nằm trong cấu trúc DOM chứa nhiều thẻ lồng nhau khiến mô hình khó xác định phần tử cha thích hợp khiến mô hình không sinh được Xpath.



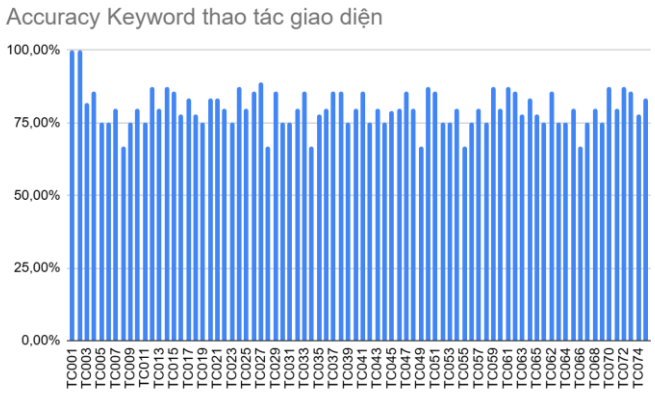
Hình 4.14. Accuracy Xpath động



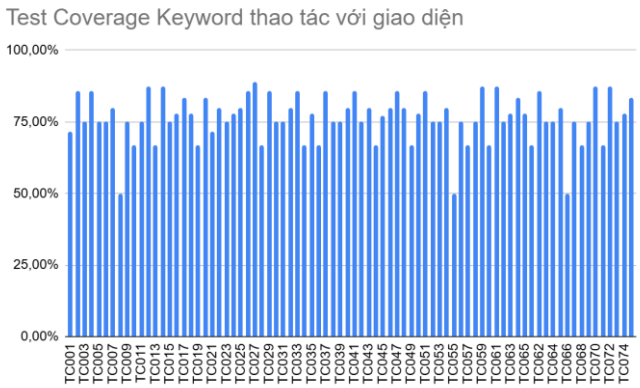
Hình 4.15. Test Coverage Xpath động

Kết quả thực nghiệm cho thấy việc sinh XPath động có độ chính xác và mức bao phủ thấp hơn đáng kể so với XPath tĩnh. Accuracy của XPath động chủ yếu dao động trong khoảng 55–75%, một số test case đạt gần 80%. Test Coverage nằm trong khoảng 50–70%, hiếm trường hợp vượt quá 75%. Nguyên nhân chủ yếu xuất phát từ những đặc thù sau của XPath động. Thứ nhất, các phần tử động thường xuất hiện trong bảng, danh sách nơi vị trí hàng cột thay đổi liên tục tùy theo dữ liệu thực tế. Khi đó, mô hình dễ sinh XPath dựa trên chỉ số vị trí cố định như cột 1 (tr[1]) và dẫn đến sai lệch khi thêm hoặc xóa hàng. Thứ hai, phần giá trị cần tham số hóa nằm trong dấu () đôi khi không được mô hình nhận diện chính xác hoặc được gắn sai vị trí trong XPath, khiến XPath đúng cú pháp nhưng không khớp phần tử thực tế. Bên cạnh đó, một số phần tử chỉ xuất hiện sau các thao tác cụ thể (như chọn bộ lọc, đổi tab) nên

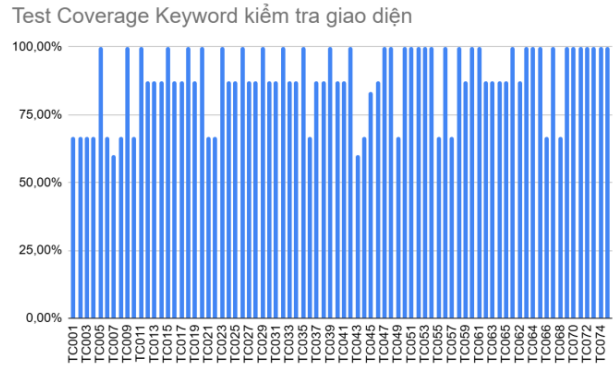
HTML được thu thập không chứa đầy đủ các phần tử này tại thời điểm kiểm thử viên thu thập HTML. Khi mô hình LLM không nhìn thấy các phần tử đó trong dữ liệu đầu vào, test coverage giảm vì không thể sinh XPath cho các phần tử chưa hiển thị trong các tệp HTML. Đây là một nguyên nhân đặc thù chỉ xảy ra với XPath là phần tử phụ thuộc mạnh vào trạng thái giao diện.



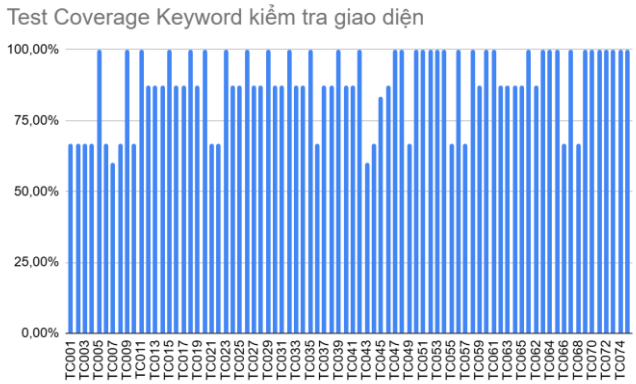
Hình 4.16. Accuracy Keyword thao tác giao diện



Hình 4.17. Test Coverage Keyword thao tác với giao diện

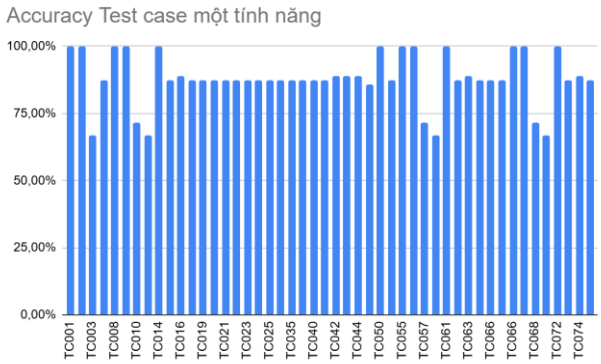


Hình 4.18. Accuracy Keyword kiểm tra giao diện

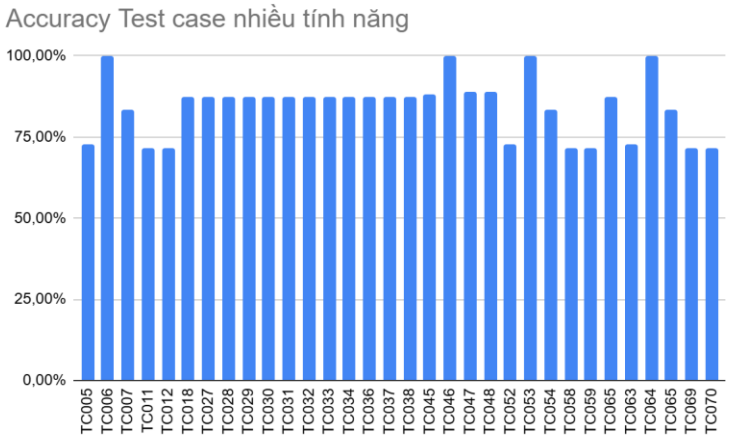


Hình 4.19. Test Coverage Keyword kiểm tra giao diện

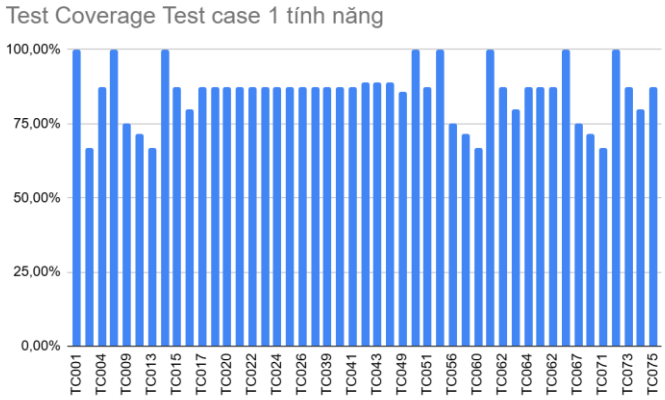
Kết quả thực nghiệm cho thấy Accuracy dao động 75–100% đối với keyword thao tác giao diện và 85–100% đối với keyword kiểm tra giao diện, các keyword sinh sai chủ yếu do việc cả hai loại keyword chứa Xpath động có tham số nhưng mô hình LLM sinh thiếu tham số. Bên cạnh đó, Accuracy giảm còn xuất phát từ việc mô hình sinh sai loại thao tác, ví dụ thay vì sinh Click Element lại sinh các lệnh không phù hợp như Run Keyword And Ignore Error khiến hành động không đúng ngữ nghĩa. Ngoài ra, Test Coverage dao động 70 - 85% với keyword thao tác và 80 - 100% với keyword kiểm tra, chỉ số Test Coverage bị kéo xuống do mô hình sinh thừa keyword không xuất hiện trong mô tả test case như các hành động chờ phần tử xuất hiện sau đó mới click vào phần tử, làm lệch số lượng bước so với các bước kiểm thử thủ công khiến Test Coverage giảm do mô hình thêm hành động khác.



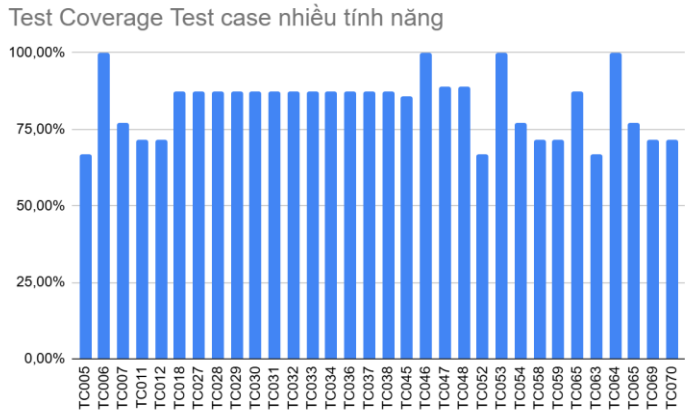
Hình 4.20. Accuracy Test case một tính năng



Hình 4.21. Accuracy Test case nhiều tính năng



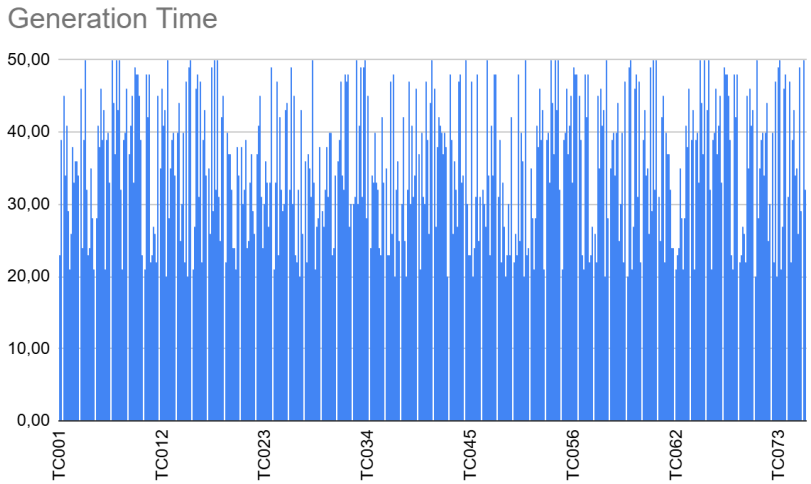
Hình 4.22. Test Coverage Test case một tính năng



Hình 4.23. Test Coverage Test case nhiều tính năng

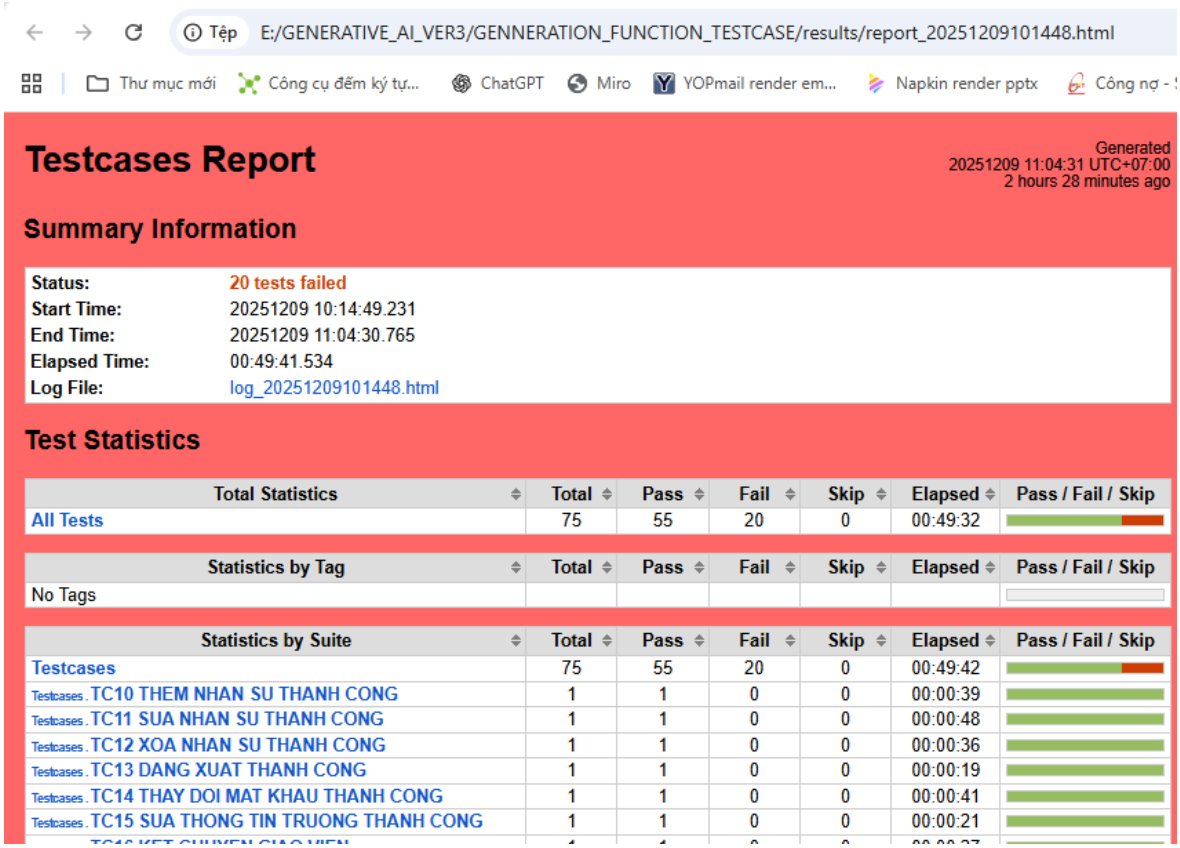
Kết quả thực nghiệm cho thấy nhóm test case một tính năng có Accuracy và Test Coverage tương đối cao (80–100%), nhóm nhiều tính năng chỉ còn khoảng 70–80%. Các sai lệch này chủ yếu xuất phát từ một số nguyên nhân. Thứ nhất, nhiều test case sử dụng keyword chứa Xpath động nhưng mô hình sinh thiếu tham số, dẫn đến test case gọi đúng tên keyword nhưng không truyền đủ dữ liệu đầu vào. Với các test case dài chứa nhiều keyword như các test case về nghiệp vụ nhập điểm tổng kết phải thao tác qua nhiều bước từ tạo lớp, tạo học sinh, cấu hình môn học, nhập điểm theo từng môn từng học kỳ, mô hình đôi khi sinh thiếu một vài bước con do độ dài mô tả và khối thông tin cần xử lý vượt quá khả năng ghi nhớ của cửa sổ ngữ cảnh. Mặc dù GitHub đã công bố rằng Copilot Chat khi sử dụng GPT-4o hỗ trợ cửa sổ ngữ cảnh lên tới 64k token, mức dung lượng này vẫn có thể không đủ đối với những test case có nhiều bước sử dụng nhiều keyword thuộc nhiều luồng nghiệp vụ khác nhau. Khi lượng thông tin đầu vào vượt ngưỡng, mô hình có xu hướng rút gọn, bỏ sót bước hoặc không sinh đầy đủ chuỗi thao tác, dẫn đến việc sinh thiếu một số bước so với các bước kiểm thử thủ công. Thứ ba, ở nhóm test case nhiều tính năng, cùng một test case thường gọi đến nhiều keyword có tên giống nhau nhưng thuộc về 2 luồng nghiệp vụ khác nhau (ví dụ cùng là “Chọn khối” nhưng là Chọn khối khi Tạo lớp và Chọn khối khi Xem sổ học bạ), mô hình dễ ánh xạ nhầm sang keyword có tên giống nhưng thuộc nghiệp vụ khác khiến nhiều bước trong test case nhiều tính năng bị sai mặc dù cấu

trúc tổng thể của test case vẫn đúng.



Hình 4.24. Generation time

Biểu đồ Generation Time cho thấy thời gian mô hình sinh mã kiểm thử chức năng giữa các lần thực thi dao động trong khoảng từ 10s đến 50s, nguyên nhân của sự dao động do những yêu cầu ngắn, ít bước và ít thông tin (ví dụ Xpath tĩnh, Keyword kiểm tra giao diện, Test case một tính năng) thì mô hình LLM xử lý nhanh hơn, trong khi các yêu cầu dài, chứa nhiều bước hoặc mô tả phức tạp (ví dụ Xpath động, Keyword thao tác giao diện, Test case nhiều tính năng) khiến mô hình phải phân tích nhiều hơn nên thời gian sinh mã tăng lên. Bên cạnh đó, thời gian phản hồi không chỉ phụ thuộc vào nội dung đầu vào mà còn bị chi phối bởi cách thức vận hành của mô hình LLM trên nền tảng cloud. Do mô hình phải phục vụ đồng thời lượng lớn người dùng, những thời điểm hệ thống quá tải có thể làm tăng thời gian sinh mã ngay cả khi yêu cầu đầu vào không thay đổi. Bên cạnh đó, thời gian phản hồi cũng chịu tác động từ chất lượng kết nối mạng của người dùng trong quá trình trao đổi dữ liệu với máy chủ, dẫn đến việc thời gian sinh mã đôi khi chậm hơn hoặc không ổn định.



Hình 4.25. Pass Rate và Execution Time

Biểu đồ thể hiện tỷ lệ Pass Rate với 73% tương ứng với 55/75 test case chạy thành công, thời gian thực thi 75 test case trung bình là gần 50s. Khi phân tích các trường hợp lỗi, nguyên nhân xuất phát từ việc mô tả các bước kiểm thử thủ công không bao gồm các bước chờ hoặc điều kiện để tải xong giao diện. Trong môi trường web, nhiều thao tác như mở chức năng, hiển thị bảng, hoặc cập nhật trường nhập liệu cần có thời gian để hoàn tất, trong khi mô hình chỉ dựa trên mô tả các bước kiểm thử thủ công nên không thể tự suy ra chính xác các điểm cần chờ, thời gian chờ là bao lâu. Hậu quả là mã kiểm thử chức năng sinh ra thường thực thi nhanh hơn tốc độ phản hồi của giao diện, dẫn đến lỗi như không tìm thấy phần tử, thao tác sai thời điểm hoặc kiểm tra giao diện khi dữ liệu chưa được hiển thị đầy đủ. Một hạn chế khác phát sinh trong quá trình thực thi kiểm thử tự động là một số kịch bản được chạy trên các thiết bị có độ phân giải thấp hoặc cửa sổ trình duyệt chưa được mở toàn màn hình, dẫn đến việc giao diện không hiển thị đầy đủ các phần tử cần tương tác, các keyword như Click Element, Input Text hoặc Element Should Be Visible có thể bị lỗi do không tìm

được đối tượng, cần phải bổ sung bước Maximize Browser Window ở đầu mỗi test case nhằm đảm bảo giao diện được hiển thị đầy đủ và đồng nhất trước khi thực hiện các thao tác.

Trong hoạt động kiểm thử chức năng tự động thông thường, mã kiểm thử được kiểm thử viên tự xây dựng dựa trên việc phân tích chi tiết test case và giao diện hệ thống. Các thao tác kiểm thử, phần tử giao diện và thứ tự nghiệp vụ được kiểm soát chặt chẽ thông qua kinh nghiệm của người viết mã, thực nghiệm trên 75 test script được viết bởi 5 kiểm thử viên cho kết quả về độ chính xác là 90-100%. Khi áp dụng quy trình sinh mã kiểm thử chức năng tự động sử dụng LLM cho kết quả về độ chính xác là 70-100%, kiểm thử viên không cần tự viết mã kiểm thử mà chuyển sang vai trò kiểm tra mã kiểm thử do mô hình LLM sinh ra và hướng dẫn mô hình điều chỉnh lại mã kiểm thử sinh ra khi mã kiểm thử sinh ra có lỗi. Với quy trình kiểm thử tự động thông thường, thời gian trung bình để xây dựng hoàn chỉnh một test case chức năng dao động từ 30 đến 45 phút, bao gồm viết XPath, keyword và test case, đồng thời yêu cầu kiểm thử viên có kỹ năng lập trình tương đối cao. Điều này gây khó khăn khi số lượng test case lớn hoặc khi hệ thống thường xuyên thay đổi theo chính sách và nghiệp vụ mới. Trong khi đó, với quy trình sinh mã kiểm thử chức năng tự động sử dụng LLM, thời gian sinh mã và hoàn thiện một test case giảm xuống còn khoảng 10 đến 30 phút, bao gồm cả thời gian kiểm thử viên kiểm tra và tinh chỉnh mã sinh ra. Như vậy, quy trình đề xuất cho phép kiểm thử viên tập trung vào việc đánh giá nghiệp vụ và xác nhận kết quả thay vì viết mã chi tiết, qua đó giảm đáng kể công sức và sự phụ thuộc vào nhân lực có chuyên môn cao, đồng thời tăng khả năng mở rộng bộ kiểm thử trong thời gian ngắn.

Qua thực nghiệm quy trình sinh mã kiểm thử chức năng tự động sử dụng LLM trên 150 test case hệ thống VNEDU, quy trình đã được ứng dụng trực tiếp vào hoạt động xây dựng và bảo trì bộ kiểm thử chức năng tự động của hệ thống VNEDU. Trong hoạt động kiểm thử chức năng tự động thông thường, kiểm thử viên thường phải thực hiện thủ công các công việc như phân tích giao diện để xác định XPath, viết keyword thao tác/kiểm tra và ghép nối thành kịch bản kiểm thử theo luồng nghiệp vụ,

đồng thời khi hệ thống thay đổi, việc cập nhật test script thường đòi hỏi rà soát và chỉnh sửa trên nhiều tệp. Quy trình sinh mã kiểm thử chức năng tự động sử dụng LLM hỗ trợ sinh mã theo cấu trúc, giảm thao tác viết mã thủ công. Nhờ việc tách bạch giữa giao diện (XPath), hành động (keyword) và nghiệp vụ (test case), khi hệ thống VNEDU thay đổi giao diện hoặc điều chỉnh nghiệp vụ, kiểm thử viên có thể cập nhật nhanh ở tầng phù hợp (XPath/keyword) hoặc tạo test case mà không phải viết lại toàn bộ kịch bản, qua đó thể hiện rõ tính ứng dụng của quy trình trong bối cảnh vận hành thực tế của hệ thống VNEDU.

Sau khi tiến hành thực nghiệm và đánh giá kết quả, đối chiếu lại các vấn đề đã nêu trong Chương 1.3 để xem mức độ mà quy trình có thể xử lý. Kết quả cho thấy bảy vấn đề đầu tiên bao gồm việc thu thập và tổ chức đầu vào chưa tập trung, kiểm thử viên phải tự viết Xpath, Keyword, test case, sự khác biệt giữa cấu trúc test case và test script, tính đa dạng của XPath - keyword - test case, sự phụ thuộc vào kinh nghiệm cá nhân, chưa có cơ chế đánh dấu liên kết dữ liệu trong các bước mô tả kiểm thử, sự trùng lặp thao tác giữa các test case, việc kiểm thử viên phải tự lưu trữ mã kiểm thử đều được giải quyết đáng kể thông qua quy trình. Vấn đề thứ tám liên quan đến việc phân tích kết quả kiểm thử sau mỗi lần chạy vẫn chưa thể giải quyết do bản chất của kết quả kiểm thử phụ thuộc vào nhiều yếu tố ngoài phạm vi sinh mã như trạng thái môi trường, dữ liệu thời điểm chạy, độ ổn định của giao diện và ngữ cảnh lỗi nên mô hình LLM khó đưa ra đánh giá chính xác chỉ dựa trên kết quả kiểm thử.

KẾT LUẬN CHƯƠNG

Chương IV đã trình bày thực nghiệm quy trình sinh mã kiểm thử tự động sử dụng mô hình LLM cho hệ thống VNEDU và đánh giá kết quả thực nghiệm trong một khoảng thời gian.

KẾT LUẬN VÀ HƯỚNG PHÁT TRIỂN

Chương này sẽ trình bày kết quả thu được sau quá trình sử dụng và hướng phát triển quy trình sinh mã kiểm thử tự động sử dụng LLM trong tương lai

Kết quả khi áp dụng quy trình sinh mã kiểm thử chức năng tự động sử dụng LLM so với kiểm thử tự động thông thường: kiểm thử viên không phải tự viết mã kiểm thử, mô hình LLM dựa trên prompt mẫu, các bước kiểm thử thủ công và các tệp HTML chứa cấu trúc giao diện tự động sinh mã kiểm thử chức năng bao gồm 6 loại dữ liệu (Xpath tĩnh, XPath động, Keyword thao tác giao diện, Keyword kiểm tra giao diện, Test case một tính năng, Test case nhiều tính năng) và cấu trúc dự án kiểm thử tự động.

Kết quả khi áp dụng quy trình sinh mã kiểm thử tự động sử dụng LLM trong mô hình phát triển phần mềm Agile/Scrum: Khi đưa quy trình vào bối cảnh phát triển phần mềm theo mô hình Agile/Scrum, kiểm thử viên có thể sử dụng mô hình LLM sinh mã kiểm thử chức năng tự động ngay khi có mô tả các bước kiểm thử và giao diện HTML mới, rút ngắn thời gian xây dựng kịch bản kiểm thử tự động trong mỗi Sprint

Hướng phát triển tương lai:

- Tự động kiểm tra lại (validate) mã kiểm thử sau khi sinh: Một hướng phát triển thực tế của quy trình là bổ sung prompt tự động phân tích và đề xuất sửa lỗi cho mã kiểm thử chức năng mà mô hình LLM đã sinh ra. Trong tương lai, có thể yêu cầu mô hình LLM phân tích lại đoạn mã kiểm thử chức năng đã sinh ra theo các tiêu chí cụ thể như tính hợp lệ của cú pháp, sự tồn tại của các biến, cấu trúc tham số của keyword hoặc những lỗi thường gặp trong test case.
- Tự động sinh Test Data: Mở rộng khả năng sinh mã kiểm thử chức năng sang phạm vi tự động sinh dữ liệu kiểm tra (test data). Hiện tại quy trình chỉ sinh biến khai báo dữ liệu kiểm tra (test data). Trong tương lai, mô hình có thể được sử dụng để tự động đề xuất giá trị phù hợp cho từng trường dữ liệu dựa trên quy luật dữ liệu phổ biến hoặc label của phần tử.

DANH MỤC TÀI LIỆU THAM KHẢO

Tiếng Anh:

[1] Chang Yi, Chen Hechang, Chen Xiaolong, Guo Siyuan, Han Tao, Liu Huiwu, Wang Jun, Xie Yuming, Zhang Liang, (2025), *Optimizing Case-Based Reasoning System for Functional Test Script Generation with Large Language Models*, [Online], Available: https://doi.org/10.48550/arXiv.2503.20576

[2] Patil Avinash, Jadon Aryan, (2025), *Advancing Reasoning in Large Language Models: Promising Methods and Approaches*, [Online], Available: https://doi.org/10.48550/arXiv.2502.03671

[3] Chen Banghao, Zhang Zhaofeng, Langrené Nicolas, Zhu Shengxin, (2025), *Unleashing the potential of prompt engineering in large language models*, [Online], Available: https://doi.org/10.48550/arXiv.2310.14735

[4] Kathiresan Gopinath, (2024), *Automated Test Case Generation with AI: A Novel Framework for Improving Software Quality and Coverage*, [Online], Available: https://doi.org/10.30574/wjarr.2024.23.2.2463

[5] Narendar Kumar Ale, (2024), *A Generative AI Framework for Enhancing Software Test Automation: Design, Implementation, and Validation*, [Online], Available: https://doi.org/10.21275/SR24604032016

[6] Tong Xiao, Jingbo Zhu, (2025), *Foundations of Large Language Models*, [Online], Available: https://doi.org/10.48550/arXiv.2501.09223

[7] Robert Feldt, Sungmin Kang, Juyeon Yoon, Shin Yoo, (2023), *Towards Autonomous Testing Agents via Conversational Large Language Models*, [Online], Available: https://doi.org/10.48550/arXiv.2306.05152

[7] Junjie Wang, Yuchao Huang, Chunyang Chen, Zhe Liu, Song Wang, Qing Wang, (2023), *Software Testing with Large Language Models: Survey, Landscape, and Vision*, [Online], Available: https://doi.org/10.48550/arXiv.2307.07221

[8] Zhenpeng Chen, Chong Wang, Weisong Sun, Guang Yang, Xuanzhe Liu, Jie M. Zhang, Yang Liu, (2025), *Promptware Engineering: Software Engineering for LLM Prompt Development*, [Online], Available:

https://doi.org/10.48550/arXiv.2503.02400

[9] Sixiang Ye, Zeyu Sun, Guoqing Wang, Liwei Guo, Qingyuan Liang, Zheng Li, Yong Liu, (2025), *Prompt Alchemist: Enhancing Code Generation with Adaptive Prompt Optimization*, [Online], Available: https://doi.org/10.48550/arXiv.2503.11085

[10] Nam Huynh, Beiyu Lin, (2025), *Large Language Models for Code Generation: A Comprehensive Survey of Challenges, Techniques, Evaluation, and Applications*, [Online], Available: https://doi.org/10.48550/arXiv.2503.01245

[11] Juyong Jiang, Fan Wang, Jiasi Shen, Sungju Kim, Sunghun Kim (2024), *A Survey on Large Language Models for Code Generation*, [Online], Available: https://doi.org/10.48550/arXiv.2406.00515

[12] Leotta Maurizio, Ricca Filippo, Yousaf Hafiz Zeeshan, Garcia Boni, (2024), *AI-Generated Test Scripts for Web E2E Testing with ChatGPT and Copilot: A Preliminary Study*, [Online], Available: https://doi.org/10.1145/3661167.3661192

[13] Ningzhi Tang, (2024), *Towards effective validation and integration of LLM-generated code*, [Online], Available: https://doi.org/10.1109/VL/HCC60511.2024.00051

[14] Li T., et al. (2024), *Large Language Models for Automated Web-Form-Test Generation: An Empirical Study*, [Online], Available: https://doi.org/10.1145/3735553.

[15] Shengcheng Yu, Chunrong Fang, Yuchen Ling, Chentian Wu, Zhenyu Chen (2023), *LLM for Test Script Generation and Migration: Challenges, Capabilities, and Opportunities*, [Online], Available: https://doi.org/10.48550/arXiv.2309.13574.

Tiếng Việt:

[16] Hệ thống quản lý giáo dục Việt Nam VNEDU https://vnedu.vn/

BẢN CAM ĐOAN

Tôi cam đoan đã thực hiện việc kiểm tra mức độ tương đồng nội dung đề án qua phần mềm Turnitin một cách trung thực và đạt kết quả mức độ tương đồng …% toàn bộ nội dung đề án. Bản đề án kiểm tra qua phần mềm là bản cứng đề án đã nộp để bảo vệ trước hội đồng. Nếu sai tôi xin chịu các hình thức kỷ luật theo quy định hiện hành của Học viện.

……., ngày …. tháng ….. năm ……

HỌC VIÊN CAO HỌC/NCS

(Ký và ghi rõ họ tên)

17% Tính tương đồng nói chung

Tổng cộng của tất cả các kết quả trùng khớp, bao gồm cả các nguồn trùng lặp, cho mỗi c...

Đã lọc khỏi Báo cáo

- Mục lục tham khảo
- Văn bản được trích dẫn

Nguồn hàng đầu

14% Nguồn Internet
9% Ấn bản
5% Bài tập được nộp (bài của học sinh)

BỘ THÔNG TIN VÀ TRUYỀN THÔNG
HỌC VIỆN CÔNG NGHỆ
BƯU CHÍNH VIỄN THÔNG

CỘNG HOÀ XÃ HỘI CHỦ NGHĨA VIỆT NAM
Độc lập - Tự do - Hạnh phúc

BÁO CÁO GIẢI TRÌNH
SỬA CHỮA, HOÀN THIỆN ĐỀ ÁN TỐT NGHIỆP THẠC SĨ

Họ và tên học viên: Nguyễn Thị Quỳnh Anh

Chuyên ngành: Hệ thống thông tin

Khóa: 2024 đợt 1

Tên đề tài: Xây dựng quy trình sinh mã kiểm thử chức năng tự động sử dụng LLM và áp dụng cho hệ thống VNEDU

Người hướng dẫn khoa học: T.S Đỗ Thị Bích Ngọc

Ngày bảo vệ: 24/01/2026

Các nội dung học viên đã sửa chữa, bổ sung trong đề án tốt nghiệp theo ý kiến đóng góp của Hội đồng chấm đề án tốt nghiệp:

TT	Ý kiến hội đồng	Sửa chữa của học viên
1	Cần làm rõ các yêu cầu của mã kiểm thử và đánh giá các yêu cầu đó: so sánh độ chính xác, tốc độ so với quy trình cũ	Đã bổ sung yêu cầu của mã kiểm thử trang 48, trang 52. Đã bổ sung đánh giá các yêu cầu đó: so sánh độ chính xác, tốc độ so với quy trình cũ trang 63
2	Cần làm rõ ứng dụng của phương pháp đề xuất vào hệ thống VNEDU	Đã bổ sung ứng dụng của phương pháp đề xuất vào hệ thống VNEDU trang 63+64
3	Cần bổ sung tài liệu tham khảo hệ thống VNEDU	Đã bổ sung tài liệu tham khảo hệ thống VNEDU mục [16] trang 64

Hà Nội, ngày……. tháng …. năm 20…

Ký xác nhận của

CHỦ TỊCH HỘI ĐỒNG CHẤM ĐỀ ÁN	THƯ KÝ HỘI ĐỒNG	NGƯỜI HƯỚNG DẪN KHOA HỌC	HỌC VIÊN
PGS.TS. Lê Hữu Lập	TS. Nguyễn Xuân Đức	TS. Đỗ Thị Bích Ngọc	Nguyễn Thị Quỳnh Anh