ĐẠI HỌC QUỐC GIA TP. HỒ CHÍ MINH
TRƯỜNG ĐẠI HỌC CÔNG NGHỆ THÔNG TIN
KHOA MẠNG MÁY TÍNH VÀ TRUYỀN THÔNG

NGUYỄN LÊ THẢO NGỌC - 21521191

BÁO CÁO ĐỒ ÁN CHUYÊN NGÀNH

Cải thiện khả năng sinh phản hồi của Web Honeypot sử dụng mô hình ngôn ngữ

Improving Response Generation of Web Honeypot Leveraging Language Modeling

GIẢNG VIÊN HƯỚNG DẪN
THS. ĐỖ THỊ THU HIỀN

TP. HỒ CHÍ MINH, 2025

ĐẠI HỌC QUỐC GIA TP. HỒ CHÍ MINH
TRƯỜNG ĐẠI HỌC CÔNG NGHỆ THÔNG TIN
KHOA MẠNG MÁY TÍNH VÀ TRUYỀN THÔNG

NGUYỄN LÊ THẢO NGỌC - 21521191

BÁO CÁO ĐỒ ÁN CHUYÊN NGÀNH

Cải thiện khả năng sinh phản hồi của Web Honeypot sử dụng mô hình ngôn ngữ

Improving Response Generation of Web Honeypot Leveraging Language Modeling

GIẢNG VIÊN HƯỚNG DẪN
THS. ĐỖ THỊ THU HIỀN

TP. HỒ CHÍ MINH, 2025

LỜI CẢM ƠN

Lời đầu tiên em xin bày tỏ lòng biết ơn chân thành đến Trường Đại học Công nghệ Thông tin - Đại học Quốc Gia Thành Phố Hồ Chí Minh vì đã tạo điều kiện thuận lợi về cơ sở vật chất để hỗ trợ em trong quá trình học tập, nghiên cứu và cũng như thực hiện đồ án chuyên ngành. Em cũng xin gửi lời tri ân đặc biệt đến các quý thầy cô giáo viên tại trường, những người đã truyền đạt những kiến thức quý báu và kinh nghiệm thực tế trong cho sinh viên trong suốt quá trình giảng dạy. Đặc biệt em rất biết ơn thầy cô đã hướng dẫn tận tình, đóng góp ý kiến và nhắc nhở em trong quá trình nghiên cứu nhằm giúp em làm hoàn thành đồ án chuyên ngành tốt nhất có thể. Ngoài ra, em muốn bày tỏ lòng biết ơn đến gia đình và bạn bè, những người đã động viên và đóng góp ý kiến quý báu trong quá trình nghiên cứu. Cuối cùng, do hạn chế kiến thức, đồ án chuyên ngành của em có thể không tránh khỏi những thiếu sót. Em hy vọng nhận được phản hồi, ý kiến đóng góp và phê bình từ quý thầy cô để đề tài nghiên cứu này trở nên hoàn thiện hơn.

MỤC LỤC

DANH MỤC HÌNH

DANH MỤC BẢNG

DANH MỤC TỪ VIẾT TẮT

AI	Artificial intelligence
RAG	Retrieval-Augmented Generation
LLM	Large language model
HTTP	Hypertext Transfer Protocol
SQLi	SQL injection
XSS	Cross-Site Scripting
CSP	Content Security Policy
IDS	Intrusion Detection System
IPS	Intrusion Prevention System
NSM	Network Security Monitoring
IP	Internet Protocol
LIHP	Low-interaction honeypots
MIHP	Medium-interaction honeypots
HIHP	High-interaction honeypots
ANN	Approximate-nearest neighbor
ZAP	OWASP Zed Attack Proxy

TÓM TẮT

Theo Bộ TT&TT, chỉ trong 8 tháng đầu năm 2024, nước ta đã ghi nhận 4029 sự cố tấn công an ninh mạng được xử lý và có tới 90033 điểm yếu và lỗ hổng bảo mật tại các server, hệ thống thông tin của các cơ quan, tổ chức được phát hiện vào tháng 6/2024. Đây là 1 thực trạng đáng báo động ở nước ta, đã tới lúc chúng ta phải dành nhiều thời gian và đầu tư nhiều hơn vào hệ thống phòng thủ, xử lý sự cố bảo mật nhằm hướng tới 1 không gian mạng an toàn, lành mạnh cho tất cả mọi người. Bên cạnh các giải pháp truyền thống như Firewall, IDS/IPS hay SIEM vốn chủ yếu liên quan đến phòng thủ bị động, chỉ phòng thủ khi hệ thống thực bị tấn công, Honeypot ra đời như một giải pháp phòng thủ chủ động bằng cách thu hút các kẻ tấn công nhằm hứng chịu tác động thay cho hệ thống thực sự và đồng thời giám sát hoạt động của chúng. Với nhiều ưu điểm và khả năng tiềm tàng, honeypot đang ngày một được quan tâm và phát triển, tuy nhiên các phiên bản truyền thống đang dần lộ điểm yếu khi đối phó với các kỹ thuật tấn công mới tận dụng AI hoặc bị nhận dạng bởi các kẻ tấn công có kinh nghiệm do đó việc tích hợp llm vào honeypot đang trở thành xu hướng chính nhằm khắc phục các điểm yếu trên . Vậy nên trong đồ án chuyên ngành này, em chọn nghiên cứu đề tài "Cải thiện khả năng sinh phản hồi của Web Honeypot sử dụng mô hình ngôn ngữ". Nhằm cải thiện khả năng sinh phản hồi, em đã kết hợp cả kỹ thuật RAG và Vector search để bổ trợ llm hoạt động tốt hơn, bên cạnh đó em còn tìm hiểu các kỹ thuật tấn công nhằm vào web để giới hạn kiến thức llm.

Trong quá trình nghiên cứu đề tài này, những việc em đã làm gồm thực hiện thu thập các mẫu HTTP Request và Response, của web và cả các URL liên quan tới tấn công, xây dựng mô hình RAG sử dụng vector search trong truy vấn dữ liệu, cải thiện prompt để llm sinh ra các phản hồi thích hợp nhất.

Ngoài ra em cũng cố gắng triển khai với nhiều mô hình AI khác nhau như gpt-4, llama3, gemini. Các kết quả đạt được gồm response do llm sinh ra và các tài liệu tìm được bằng kỹ thuật vector search đều được đánh giá bằng độ tương đồng Cosine và thống kê dưới dạng biểu đồ.

Chương 1. GIỚI THIỆU

1.1. Giới thiệu về đề tài

Với sự phát triển mạnh mẽ của các công nghệ cũng như trí tuệ nhân tạo (AI), các cuộc tấn công mạng không ngừng xuất hiện và ngày càng trở nên tinh vi hơn và gây ra rất nhiều thiệt hại về tài chính, rò rỉ thông tin mật và đe doạ quyền riêng tư của người dùng. Có rất nhiều loại tấn công và mục tiêu tấn công khác nhau nhưng với xu hướng số hoá và chuyển đổi số, tại việt nam, các cuộc tấn công trực tuyến nhằm vào web hay các trình duyệt web đang dần tăng lên so với các loại khác. Một web application có thể bị tấn công vì nhiều lý do gồm lỗi hệ thống do mã hoá không chính xác, lỗi cấu hình, lỗi thiết kế logic, lỗi phần cứng hoặc phần mềm trên web server,…do đó không phải lúc nào các chuyên gia về bảo mật cũng kịp thời phát hiện hoặc dự đoán được rủi ro có thể xảy ra. Vì vậy honeypot ra đời như giải pháp chủ động phòng thủ, thay vì chờ xảy ra tấn công rồi mới tìm cách ngăn chặn, khắc phục hậu quả, honeypot sẽ chủ động dụ dỗ kẻ tấn công hoặc hứng chịu đòn tấn công thay cho hệ thống thực sự nhằm giám sát cách thức tấn công và ghi log các sự kiện diễn ra để các chuyên gia bảo mật có thêm thông tin nhằm cải tiến hệ thống phòng thủ trước khi thực sự có cuộc tấn công diễn ra. Nhưng với sự phát triển mạnh mẽ của AI và các công nghệ hỗ trợ AI, các honeypot truyền thống dần mất ưu thế khi đối mặt với các cuộc tấn công sử dụng AI và dễ dàng bị nhận dạng bởi các kẻ tấn công giàu kinh nghiệm do chúng không thể hoàn toàn mô phỏng hành động của hệ thống mà chúng đang đóng giả. Vì thế các nghiên cứu về việc tích hợp AI vào honeypot, cải tiến honeypot với LLM đang ngày một nhiều bởi hướng nghiên cứu này giúp tạo ra đa dạng dữ liệu giả hơn, khắc phục hạn chế tương tác giữa honeypot và kẻ tấn công nhờ vào khả năng mô phỏng tự nhiên và linh hoạt của LLM; phát triển khả năng tự động hoá và

nâng cao hiệu quả phân tích log nhờ vào AI. Với đề tài "Cải thiện khả năng sinh phản hồi của Web Honeypot sử dụng mô hình ngôn ngữ", em đề xuất biện pháp cải thiện khả năng phản hồi của web honeypot bằng cách triển khai honeypot với mô hình kết hợp RAG và LLM nhằm tạo ra honeypot sinh ra các phản hồi phù hợp với ngữ cảnh và tự nhiên hơn, có thể đóng giả web server mà không cần cấu hình phức tạp.

1.2 Mục tiêu, đối tượng và phạm vi nghiên cứu

1.2.1 Mục tiêu nghiên cứu

Đề tài tập trung vào việc cải thiện khả năng sinh phản hồi của Web Honeypot và kết hợp mô hình RAG với LLM để nâng cao hiệu quả phản hồi của honeypot.

1.2.2 Đối tượng nghiên cứu

Các đối tượng nghiên cứu chính gồm:

+ Kỹ thuật vector search

+ Mô hình RAG

+ Mô hình LLM

+ Các thông điệp HTTP tương ứng trong cuộc tấn công

1.2.3 Phạm vi nghiên cứu

Phạm vi nghiên cứu chính gồm:

+ Tập trung vào giả lập các phản hồi ứng với các cuộc tấn công web như Brute-force, Command injection, SQL injection, Blind SQL injection, Cross-Site Scripting (XSS), Content Security Policy (CSP) Bypass

Chương 2. CƠ SỞ LÝ THUYẾT VÀ CÁC NGHIÊN CỨU LIÊN QUAN

2.1. Cơ sở lý thuyết

2.1.1 Honeypot

a) Khái niệm

- Honeypot là một công cụ an ninh mạng được thiết kế để đóng vai trò như một "mồi nhử," nhằm thu hút các cuộc tấn công vào một hệ thống được giả lập. Thay vì bảo vệ trực tiếp hệ thống chính, giá trị của honeypot nằm ở khả năng thu thập thông tin về hành vi, công cụ, và mục tiêu của kẻ tấn công. Ban đầu, ý tưởng honeypot đã xuất hiện từ những năm 1990 trong lĩnh vực bảo mật thông tin, nhưng khái niệm này chỉ được chính thức định nghĩa vào đầu những năm 2000 bởi Lance Spitzner. Honeypots có thể được thiết lập ở nhiều cấp độ khác nhau, từ mô phỏng các dịch vụ cơ bản đến các hệ thống hoàn chỉnh, nhằm phục vụ các mục tiêu nghiên cứu hoặc sản xuất [1].

- Mục tiêu chung của honeypot là đánh lạc hướng kẻ tấn công khỏi mục tiêu thực sự của chúng hoặc thu thập thông tin về kẻ tấn công và các kiểu tấn công, chẳng hạn như tập hợp các host thường là mục tiêu và hay thực hiện yêu cầu/phản hồi. Tuy nhiên, cần lưu ý rằng honeypot không nên được coi là cách triển khai giải quyết một vấn đề nào đó mà là một khái niệm chung.

- Bất kể honeypot được triển khai ở đâu và như thế nào (ví dụ: bộ định tuyến, tập lệnh mô phỏng các dịch vụ cụ thể, máy ảo hoặc hệ thống vật lý tiêu chuẩn), honeypot đều có đóng góp bằng cách nó được đặt vào hoàn cảnh dễ bị tấn công.

- Sự khác nhau giữa honeypot với các khái niệm/công nghệ bảo mật khác như Firewall, IDS, NSM, ...
+ Firewall (Tường lửa): Firewall hoạt động chủ yếu để ngăn chặn các cuộc xâm nhập bằng cách lọc lưu lượng mạng dựa trên các quy tắc đã định trước. Ngược lại, honeypot không ngăn chặn trực tiếp mà được thiết kế để hấp dẫn

và quan sát các cuộc tấn công, ghi lại các hành động của kẻ tấn công mà không ảnh hưởng đến lưu lượng của hệ thống chính.

+ Intrusion Detection System (IDS): IDS là hệ thống phát hiện các dấu hiệu tấn công bằng cách phân tích các gói dữ liệu và so sánh với các mẫu chữ ký tấn công đã biết. IDS thường đưa ra cảnh báo khi phát hiện các mối đe dọa nhưng có thể bị quá tải với cảnh báo sai. Trong khi đó, honeypot giảm thiểu cảnh báo sai vì mọi kết nối đều là bất thường, cho phép thu thập thông tin chính xác hơn.

+ Intrusion Prevention System (IPS): IPS là hệ thống kết hợp giữa firewall và IDS, có khả năng phát hiện và ngăn chặn các mối đe dọa. Dù vậy, IPS cũng chỉ ngăn chặn các cuộc tấn công đã biết, còn honeypot có thể ghi nhận các cuộc tấn công mới hoặc chưa từng được phát hiện trước đó, bao gồm cả lỗ hổng "zero-day".

+ Network Security Monitoring (NSM) và Log-Monitoring: Các công cụ NSM và log-monitoring giúp theo dõi các nhật ký hệ thống và lưu lượng mạng nhằm phát hiện hoạt động đáng ngờ. Khác với honeypot, NSM và log-monitoring cần xử lý lưu lượng và hoạt động sản xuất thực tế, điều này dễ gây nhiễu. Honeypot thì chỉ tập trung vào các hành vi bất thường, giúp việc phân tích dữ liệu trở nên đơn giản và chính xác hơn.

b) Phân loại

Có khá nhiều cách phân loại honeypot tuỳ theo mục đích và nhu cầu mà chúng ta quan tâm. Một số cách phân loại phổ biến [1] là:

- Phân loại dựa trên lĩnh vực hoạt động và mục đích của honeypot, có hai nhóm chính:

+ Production honeypots: Loại phổ biến nhất, production honeypots là loại honeypot được sử dụng để thu thập thông tin liên quan đến an ninh mạng trong mạng sản xuất của doanh nghiệp hoặc tổ chức. Sau khi triển khai, honeypot sản xuất sẽ chờ một cuộc tấn công. Nếu một cuộc tấn công xảy ra, nó có thể thu thập dữ liệu như địa chỉ Giao thức Internet (IP) gốc, tần suất và khối lượng lưu lượng, phụ kiện thư mục và nhiều hơn nữa. Production honeypots được ưa chuộng trong các doanh nghiệp vì chúng dễ sử dụng trong khi tiết lộ thông tin cần thiết về các mối đe dọa và lỗ hổng mạng mà mạng của họ đang phải đối mặt. Tuy nhiên, production honeypots thường không tiết lộ nhiều thông tin như các nghiên cứu khác.

+ Research honeypots: là loại honeypot được thiết kế để thu thập thông tin về phương pháp và chiến thuật của tin tặc. Chúng chứa dữ liệu giả mạo có giá trị và ghi lại thông tin về các cuộc tấn công và lỗ hổng. Khác với production honeypots, thường được sử dụng trong mạng doanh nghiệp, research honeypots chủ yếu được triển khai bởi chính phủ và tổ chức nghiên cứu trên nhiều mạng và địa điểm khác nhau. research honeypots cũng phức tạp hơn production honeypots. Do đó, chúng đòi hỏi nhiều công việc hơn để triển khai. Tuy nhiên, do sự phức tạp của chúng, research honeypots cung cấp thêm thông tin về các cuộc tấn công và lỗ hổng.

- Phân loại dựa trên các đặc điểm tương tác:

+ Low-interaction honeypots (LIHP): Mô phỏng một số dịch vụ cơ bản (như SSH, FTP), nhưng không cho phép truy cập vào hệ điều hành. Ưu điểm chính của chúng là dễ triển khai và bảo trì, đồng thời chúng là những công cụ thống kê tuyệt vời. Tuy nhiên, chúng bị hạn chế khi phát hiện các kiểu tấn công mới.

+ Medium-interaction honeypots (MIHP): Có khả năng tương tác cao hơn LIHP và mô phỏng các dịch vụ phức tạp hơn. MIHP có thể tạo ra các phản hồi hợp lý để đánh lừa và thu hút các cuộc tấn công tiếp theo. Tuy nhiên, chúng vẫn không cung cấp quyền truy cập đầy đủ vào hệ thống.

+ High-interaction honeypots (HIHP): Đây là các honeypot phức tạp nhất, cho phép kẻ tấn công tương tác với môi trường hệ điều hành thực tế, giúp thu thập dữ liệu chi tiết về các cuộc tấn công. HIHP yêu cầu triển khai và giám sát phức tạp, vì khả năng bị xâm phạm cao hơn và dữ liệu thu thập được thường phải phân tích bằng tay.

- Phân loại dựa trên hướng tương tác:

+ Server honeypots: Đợi kẻ tấn công khởi tạo kết nối và thu thập thông tin từ các yêu cầu của kẻ tấn công.

+ Client honeypots: Chủ động tìm kiếm các thực thể độc hại và tương tác với chúng. Ví dụ, client honeypots có thể kiểm tra tính an toàn của các trang web bằng cách yêu cầu trang và kiểm tra hoạt động bất thường.

- Phân loại dựa trên tính vật lý:

+ Physical honeypots: Được triển khai trên máy vật lý thực trong hệ thống mạng. Physical honeypots có thể đạt được độ xác thực cao hơn nhưng ít được sử dụng hơn do chi phí liên quan.

+ Virtual honeypots: Được triển khai dưới dạng máy ảo, cho phép một máy chủ mô phỏng nhiều honeypot cùng lúc, giúp tiết kiệm chi phí phần cứng và tăng tính linh hoạt.

c) Ưu và nhược điểm của Honeypot

- **Ưu điểm**

Thu thập dữ liệu có giá trị: Honeypot thu thập dữ liệu không bị nhiễu từ các hoạt động tương tác với nó và thường có giá trị cao. Điều này làm cho các tập dữ liệu có kích thước nhỏ hơn và việc phân tích dữ liệu trở nên ít phức tạp hơn.

Độc lập với khối lượng công việc: Honeypot chỉ cần xử lý lưu lượng hướng đến chúng (incoming traffic) hoặc bắt nguồn từ chúng (outcoming traffic). Điều này có nghĩa là chúng độc lập với khối lượng công việc mà các hệ thống thực tế (hệ thống mà honeypot đang đóng giả) gặp phải.

Phát hiện khai thác Zero Day: Honeypot thu thập mọi thứ được sử dụng để chống lại chúng do đó các chiến lược tấn công chưa biết tới và zero-day-exploits sẽ được phát hiện.

Giảm kết quả dương tính giả và âm tính giả: Client Honeypot xác minh các cuộc tấn công bằng cách phát hiện các thay đổi trạng thái hệ thống. Việc này dẫn đến giảm kết quả dương tính giả và âm tính giả trong việc nhận định 1 cuộc tấn công.

Linh hoạt: Honeypot là một khái niệm rất linh hoạt như có thể thấy qua số lượng lớn các phần mềm honeypot khác nhau như SSH Honeypot, Web Honeypot, SMTP Honeypot,….Các honeypot có thể được thiết kế tuỳ chỉnh để phù hợp nhất cho các nhiệm vụ cụ thể hoặc đảm nhận nhiều nhiệm vụ cùng lúc.

- **Nhược điểm**

Tầm nhìn hạn chế: Server Honeypot đều có một vấn đề chung là chúng vô giá trị nếu không ai tấn công chúng. Miễn là kẻ tấn công không gửi bất kỳ gói nào

đến honeypot, thì honeypot sẽ không biết về bất kỳ hoạt động tấn công nào có thể diễn ra trên hệ thống mà nó đang đóng giả.

Bị lấy dấu vân tay: Các honeypot có mức độ tương tác thấp chỉ mô phỏng cơ bản các các dịch vụ mạng, giao thức TCP và IP. Điều này dẫn tới các dịch vụ của chúng có thể hoạt động khác so với các dịch vụ thực tế, do đó kẻ tấn công có thể dễ dàng lấy dấu vân tay honeypot và phát hiện ra chúng.

Rủi ro cho môi trường: Nếu honeypot bị khai thác, chúng có thể tạo ra các rủi ro cho môi trường người dùng thực tế hay bị lợi dụng để tấn công vào hệ thống thật của tổ chức, doanh nghiệp. Mức độ tương tác của honeypot càng cao thì rủi ro sinh ra khi chúng bị khai thác thành công càng lớn.

2.1.2 Web Honeypot

Web-based honeypot là một hệ thống mồi nhử bắt chước một ứng dụng web hoặc máy chủ web, nhằm mục đích dụ dỗ và thu hút những kẻ tấn công mạng. Mỗi honeypot đều được thiết kế với các lỗ hổng xác định nhằm khiến chúng dễ bị tấn công cũng như dụ dỗ kẻ tấn công nhắm mục tiêu vào bản thân nó thay vì hệ thống thực sự. Khi một honeypot thành công thu hút và bị tấn công, honeypot sẽ ghi log toàn bộ hoạt động diễn ra giữa nó và kẻ tấn công, cố gắng thu thập nhiều thông tin nhất có thể mà không làm lộ thông tin thực nhằm giúp các nhà nghiên cứu bảo mật có thể thu thập thông tin về các kỹ thuật tấn công và công cụ mà kẻ tấn công đã dùng. Từ đó tìm ra được biện pháp bảo vệ các hệ thống thực và ngăn chặn các cuộc tấn công tốt hơn.

Với Web-based honeypot, chúng thường được thiết kế với các lỗ hổng tấn công như SQL injection, Cross-Site Scripting (XSS), Remote File Inclusion, Directory Traversal, Sensitive Data Exposure, Command injection,… và được đặt ở những nơi mà attacker có thể phát hiện và thực hiện được việc scanning nhưng phải tách biệt với hệ thống thật sự để kẻ tấn công không phát ra việc họ đang bị lừa (tấn công sai mục tiêu).

Một khi thành công đánh lừa kẻ tấn công, Web-based honeypot sẽ giúp chúng ta phát hiện được lỗ hổng cần khắc phục trên hệ thống web thực, phát hiện các kỹ thuật tấn công mới nhằm vào web, cũng như kéo dài thời gian cho chúng ta kịp thời triển khai các biện pháp phòng thủ nhằm bảo vệ hệ thống

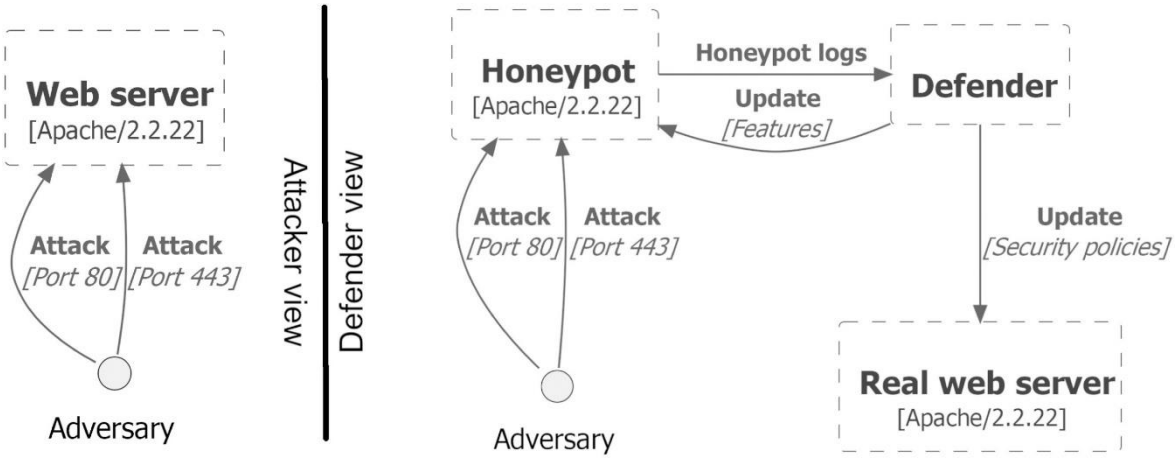
thực và truy vết kẻ tấn công. Hình ? cho thấy cách honeypot hoạt động trong trường hợp bị tấn công [2]:

+ Dưới góc nhìn của attacker:

Attacker tưởng honeypot là web server (mục tiêu tấn công) nên thực hiện các cuộc tấn công như SQLi, XSS,…để xâm phạm web server nhằm lấy các thông tin mà họ nghĩ là thực (trong khi đây là các thông tin giả, không có thực và được honeypot cung cấp).

+ Dưới góc nhìn của defender

Sau khi bên Defender nhận được log thông báo phát hiện tấn công của honeypot, họ sẽ thêm các đặc điểm cho honeypot để chúng hành động chân thực hơn để duy trì việc đánh lừa attacker, đồng thời sử dụng các thông tin chi tiết thu thập được từ các cuộc tấn công mà honeypot đã ghi lại nhằm củng cố các biện pháp phòng thủ, bảo vệ hệ thống thực sự.



Hình 1: Góc nhìn của Attacker và Defender

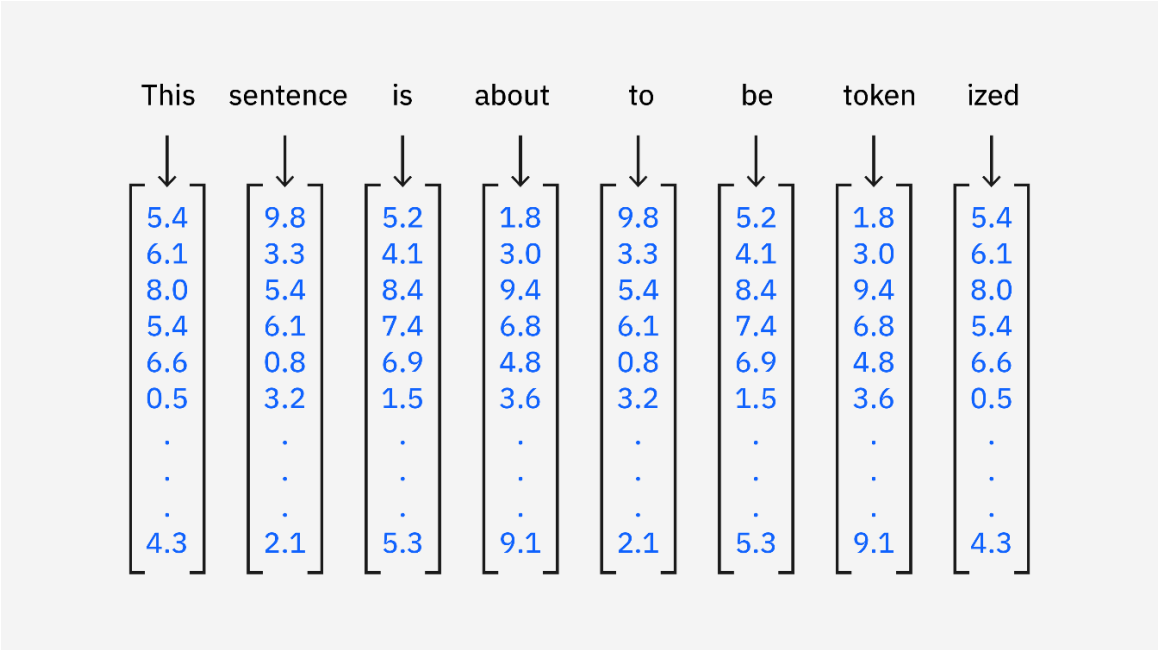
2.1.3 Vector Search

a) Khái niệm

Vector Search là phương pháp tìm kiếm dữ liệu có liên quan tới câu truy vấn đầu vào dựa trên độ tương tự giữa các vector embeddings thay vì chỉ dựa trên tương tự về mặt từ ngữ, các vector càng gần nhau thì chúng càng có ý nghĩa

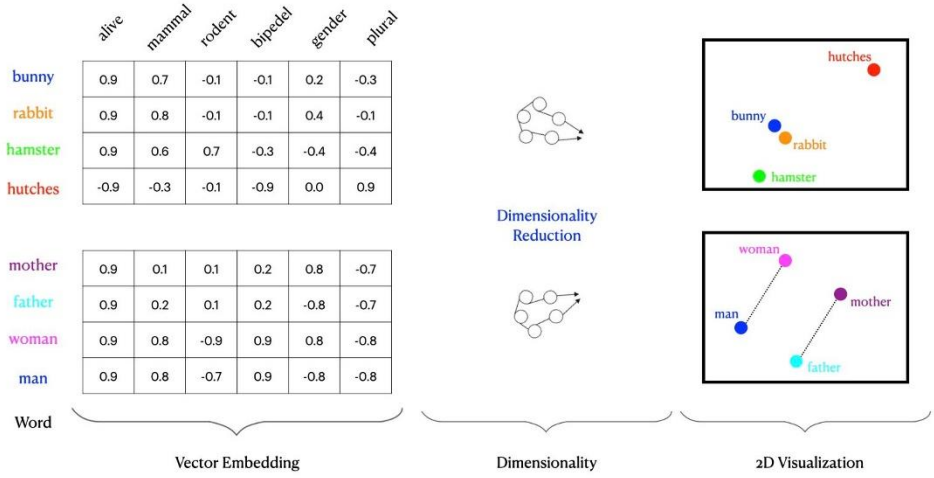
giống nhau. Nhờ tận dụng ưu điểm có thể biểu diễn đặc trưng ngữ nghĩa của embedding vector, vector search có thể khắc phục nhiều nhược điểm của các công cụ tìm kiếm truyền thống. Ví dụ: nếu bạn tìm kiếm cụm từ "Hoa có màu đỏ", tìm kiếm toàn văn bản chỉ trả về dữ liệu có chứa rõ ràng các từ khóa {Hoa, có, màu đỏ}. Tuy nhiên, tìm kiếm ngữ nghĩa có thể trả về dữ liệu có ý nghĩa tương tự, chẳng hạn như “hoa hồng”, “hoa dâm bụt”, “hoa anh đào”, “hoa phượng”,…hoặc các nội dung liên quan như bài viết mô tả cách trồng hoa hồng đẹp hơn, ý nghĩa của các loài hoa màu đỏ,…

Sự khác biệt giữa cách tìm kiếm truyền thống (như Full Text Search) và tìm kiếm bằng vectơ ngữ nghĩa (Semantic Vector Search) là do các phương pháp truyền thống thường biểu diễn dữ liệu bằng các đơn vị rời rạc, chẳng hạn như keyword, teg hoặc metadata. Trong khi đó, phương pháp tìm kiếm bằng vector hỗ trợ công cụ tìm kiếm hiểu được ý nghĩa thực sự của câu truy vấn dựa trên ngữ nghĩa và ngữ cảnh đưa ra câu truy vấn bằng cách vector hoá cả câu hoặc từng từ một trong câu truy vấn rồi tính toán mức độ tương đồng giữa vector truy vấn (biểu diễn câu truy vấn) với các vector tài liệu (đại diện cho các dữ liệu được truy vấn) [3].



Hình 2: Vector hoá từng từ trong một câu

Như ví dụ trên, tìm kiếm truyền thống dựa vào việc khớp chính xác để lấy kết quả liên quan. Ngược lại, vector search biểu diễn dữ liệu dưới dạng vectơ dày đặc (vectơ mà hầu hết hoặc tất cả các phần tử đều khác không) trong một không gian vectơ liên tục (là không gian toán học mà dữ liệu được biểu diễn dưới dạng vectơ). Mỗi chiều của vectơ dày đặc tương ứng với một đặc trưng tiềm ẩn hoặc khía cạnh của dữ liệu, một đặc điểm hoặc thuộc tính cơ bản không được quan sát trực tiếp mà được suy ra từ dữ liệu thông qua các mô hình hoặc thuật toán toán học. Các đặc tính này giúp liên kết các mẫu với các mối quan hệ ẩn trong dữ liệu, cho phép biểu diễn các mục một cách ý nghĩa và chính xác hơn dưới dạng vectơ trong một không gian đa chiều. Nhìn vào hình 3, chúng ta sẽ thấy được một quy trình đơn giản của để tạo ra một Vector Embedding [4]



Hình 3: Minh hoạ quá trình tạo Vector Embedding

Các phương pháp tìm kiếm truyền thống có thể gặp khó khăn về khả năng mở rộng đối với các tập dữ liệu lớn hoặc dữ liệu nhiều chiều do các hạn chế về tính toán và bộ nhớ. Ngược lại, việc nhúng vector (vector embeddings) giúp dễ dàng mở rộng các tập dữ liệu lớn hơn, các mô hình phức tạp hơn. Không giống như biểu diễn dữ liệu thưa thớt, nơi hầu hết các giá trị là số không trên các chiều, Embedding là một kỹ thuật đưa một vector có số chiều lớn, thường

ở dạng thưa, về một vector có số chiều nhỏ, thường ở dạng dày đặc và có giá trị khác không ở hầu hết các chiều. Điều này cho phép vector embeddings lưu trữ nhiều thông tin hơn trong một không gian nhỏ hơn, ít chiều hơn, do đó yêu cầu ít bộ nhớ hơn. Kết quả là, các thuật toán và mô hình học máy có thể sử dụng nhúng hiệu quả hơn với ít tài nguyên tính toán hơn.

b) Độ tương đồng trong vector search

Trong vector search, mức độ liên quan được xác định bằng cách tính độ tương đồng giữa vectơ truy vấn và vectơ tài liệu. Để so sánh hai vectơ với nhau và xác định độ tương đồng của chúng, có thể sử dụng một số phép đo khoảng cách, chẳng hạn như Euclidean distance hoặc độ tương đồng Cosine similarity. Ngoài ra chúng ta cũng có thể dùng thuật toán tìm kiếm approximate-nearest neighbor (ANN) để cải thiện hiệu quả và tăng tốc độ tìm kiếm.

c) Embedding model

Embedding model là các mô hình sử dụng thuật toán kết hợp với Large Language Model (LLM) để tạo ra các Vector Embedding thể hiện được ý nghĩa của dữ liệu trong ngữ cảnh đặt ra.

Hiện nay có rất nhiều loại Embedding model khác nhau, mỗi loại có đặc điểm và mục tiêu khác nhau tuỳ thuộc vào cách mô hình đó được đào tạo. Do đó, chúng ta cần xem xét và quyết định chọn mô hình phù hợp với dữ liệu, cũng như mục đích và nhu cầu sử dụng.

Một trong các nguồn phổ biến nhất để tìm benchmark cho embedding model là MME [5] và MTEB [6]

+ MME: A Comprehensive Evaluation Benchmark for Multimodal Large Language Models
Cung cấp các bảng xếp hạng theo Benchmark MME gồm: 2 bảng xếp hạng tổng quan là perception (Tổng điểm tối đa là 2000) và cognition (Tổng điểm tối đa là 800); 14 bảng xếp hạng tương ứng với 14 subtask (mỗi subtask có tổng điểm tối đa là 200).

Có tổng cộng 30 mô hình ngôn ngữ lớn đa phương thức tiên tiến (advanced MLLMs) tham gia vào các bảng xếp hạng này nhưng mỗi bảng xếp hạng chỉ hiển thị 10 mô hình đứng đầu. Đặc biệt ba mô hình đứng đầu mỗi bảng xếp hạng sẽ được đánh dấu bằng biểu tượng cúp rõ ràng.

Rank	Model	Score
1	WeMM	1621.66
2	InfMLLM	1567.99
3	SPHINX	1560.15
4	Lion	1545.80
5	LLaVA	1531.31
6	XComposer-VL	1528.45
7	Qwen-VL-Chat	1487.58
8	mPLUG-Owl2	1450.20
9	Skywork-MM	1419.08
10	GPT-4V	1409.43

(1) **Perception**

Rank	Model	Score
1	GPT-4V	517.14
2	Lion	445.71
3	WeMM	445.00
4	MMICL	428.93
5	XComposer-VL	391.07
6	Qwen-VL-Chat	360.71
7	LLaMA-Adapter V2	356.43
8	Skywork-MM	356.43
9	InfMLLM	347.14
10	BLIVA	331.43

(2) **Cognition**

Rank	Model	Score
1	Otter	195.00
1	Lynx	195.00
1	WeMM	195.00
1	Muffin	195.00
1	SPHINX	195.00
6	GIT2	190.00
6	XComposer-VL	190.00
6	Lion	190.00
6	GPT-4V	190.00
6	InfMLLM	190.00

(3) Existence

Rank	Model	Score
1	Muffin	163.33
2	MMICL	160.00
3	GPT-4V	160.00
3	SPHINX	160.00
5	XComposer-VL	158.33
6	LLaVA	155.00
6	Lion	155.00
6	mPLUG-Owl2	155.00
9	Lynx	151.67
9	Skywork-MM	151.67

(4) Count

Rank	Model	Score
1	Lion	153.33
1	SPHINX	153.33
3	InfMLLM	143.33
4	LLaVA	133.33
4	Qwen-VL-Chat	128.33
5	WeMM	126.67
5	XComposer-VL	126.67
6	GIT2	96.67
7	GPT-4V	95.00
8	Lynx	90.00

(5) Position

Rank	Model	Score
1	InfMLLM	185.00
2	BLIVA	180.00
2	Lion	180.00
4	LLaVA	170.00
4	Lynx	170.00
4	Qwen-VL-Chat	170.00
4	WeMM	168.33
5	LRV-Instruction	165.00
5	XComposer-VL	165.00
5	Muffin	165.00

(6) Color

Rank	Model	Score
1	GPT-4V	192.18
2	Lion	181.63
3	Qwen-VL-Chat	178.57
4	Skywork-MM	175.85
5	SPHINX	164.29
6	InfMLLM	163.27
7	XComposer-VL	161.90
8	LLaVA	160.54
8	WeMM	160.54
9	mPLUG-Owl2	160.20

(7) Poster

Rank	Model	Score
1	WeMM	179.12
2	SPHINX	177.94
3	Otter	172.65
4	mPLUG-Owl2	164.41
5	Cheetor	164.12
6	InfMLLM	161.47
7	Skywork-MM	160.29
8	LLaVA	152.94
9	Lion	150.59
10	XComposer-VL	150.29

(8) Celebrity

Rank	Model	Score
1	WeMM	176.25
2	InfMLLM	165.25
3	Lynx	164.50
4	LLaVA	161.25
5	SPHINX	160.00
6	XComposer-VL	159.75
7	Lion	159.00
8	Otter	158.75
9	GIT2	158.50
10	Octopus	157.25

(9) Scene

Rank	Model	Score
1	Lion	173.00
2	WeMM	172.25
3	LLaVA	170.50
4	SPHINX	168.09
5	LLaMA-Adapter V2	167.84
6	InfMLLM	167.00
7	XComposer-VL	165.25
8	Qwen-VL-Chat	164.00
9	Lynx	162.00
10	LRV-Instruction	160.53

(10) Landmark

Rank	Model	Score
1	WeMM	156.00
2	GPT-4V	148.00
3	GIT2	146.25
4	BLIP-2	136.50
5	MMICL	135.50
6	InstructBLIP	134.25
6	mPLUG-Owl2	134.25
7	SPHINX	134.00
8	BLIVA	133.25
9	Lion	130.75

(11) Artwork

Rank	Model	Score
1	GPT-4V	185.00
2	Skywork-MM	162.50
3	WeMM	147.50
4	Qwen-VL-Chat	140.00
5	InfMLLM	132.50
6	LLaVA	125.00
6	XComposer-VL	125.00
7	BLIP-2	110.00
7	LRV-Instruction	110.00
8	LaVIN	107.50

(12) OCR

Rank	Model	Score
1	GPT-4V	142.14
2	WeMM	140.00
3	XComposer-VL	138.57
4	BLIVA	136.43
4	MMICL	136.43
5	InfMLLM	132.14
6	Qwen-VL-Chat	130.71
7	SPHINX	130.00
8	InstructBLIP	129.29
9	LLaVA	127.86

(13) Commonsense Reasoning

Rank	Model	Score
1	GPT-4V	130.00
2	Lion	105.00
3	Skywork-MM	95.00
4	MMICL	82.50
5	Cheetor	77.50
6	Otter	72.50
7	LRV-Instruction	70.00
8	LaVIN	65.00
9	Multimodal-GPT	62.50
10	mPLUG-Owl	60.00

(14) Numerical Calculation

Rank	Model	Score
1	Qwen-VL-Chat	147.50
1	Lion	147.50
3	MMICL	132.50
4	WeMM	130.00
4	LLaMA-Adapter V2	112.50
4	XComposer-VL	112.50
5	Octopus	102.50
5	mPLUG-Owl2	102.50
5	InfMLLM	102.50
6	LRV-Instruction	85.00

(15) Text Translation

Rank	Model	Score
1	GPT-4V	170.00
2	WeMM	117.50
3	LLaMA-Adapter V2	90.00
4	Cheetor	87.50
5	XComposer-VL	85.00
6	MMICL	77.50
7	BLIP-2	75.00
8	LRV-Instruction	72.50
9	Otter	70.00
10	Lion	67.50

(16) Code Reasoning

Hình 4: Leaderboards theo benchmark MME

+ MTEB: Massive Text Embedding Benchmark

Gồm 8 embedding task với tổng cộng 56 tập dữ liệu và 112 ngôn ngữ. Các embedding task được phân thành: Bitext mining, Classification, Clustering, Pair Classification, Reranking, Retrieval, Semantic Textual Similarity và Summarisation. Các tập dữ liệu chứa các văn bản có độ dài khác nhau và được phân thành 3 loại là Sentence to sentence, Paragraph to paragraph, and Sentence to paragraph

Rank	Model	Model Size (Million Parameters)	Memory Usage (GB, fp32)	Embedding Dimensions	Max Tokens	Average (56 datasets)	Classification Average (12 datasets)	Clustering Average (11 datasets)
1	voyage-3-m-exp					74.03	90.16	61.45
2	NV-Embed-v2	7851	29.25	4096	32768	72.31	90.37	58.46
3	jasper_en_vision_language_v1					72.02	88.49	58.04
4	bge-en-icl	7111	26.49	4096	32768	71.67	88.95	57.89
5	stella_en_1.5B_v5	1543	5.75	8192	131072	71.19	87.63	57.69
6	SFR-Embedding-2_R	7111	26.49	4096	32768	70.31	89.05	56.17
7	gte-Qwen2-7B-instruct	7613	28.36	3584	131072	70.24	86.58	56.92
8	stella_en_400M_v5	435	1.62	8192	8192	70.11	86.67	56.7
9	stella_en_400M_v5	435	1.62	8192	8192	70.11	86.67	56.7
10	bge-multilingual-gemma2	9242	34.43	3584	8192	69.88	88.08	54.65

Hình 5: Leaderboards theo benchmark MTEB

d) Quy trình hoạt động

Bước 1: Chọn 1 mô hình nhúng (Embedding model) để thực hiện nhúng dữ liệu

Khi chọn Embedding model cần xem xét các yếu tố sau:

✓ **Số chiều nhúng (Embedding Dimensions):** Đây là độ dài của vector embedding.

Vectơ có số chiều *nhỏ hơn* tiết kiệm không gian lưu trữ hơn.

Vectơ có số chiều *lớn hơn* có thể nắm bắt các mối quan hệ phức tạp và tinh tế hơn trong dữ liệu.

Cần cân bằng giữa hiệu quả lưu trữ và khả năng biểu diễn của mô hình.

✓ **Số lượng token tối đa (Max Tokens):** Đây là số lượng *token* tối đa mà mô hình có thể nén vào một vector embedding duy nhất.
Token có thể hiểu là các đơn vị nhỏ nhất của văn bản, thường là từ hoặc một phần của từ. Ví dụ, câu "Tôi đi học." có thể được chia thành 3 token: "Tôi", "đi", "học".

Số lượng token tối đa cho biết mô hình có thể xử lý bao nhiêu văn bản cùng một lúc để tạo ra một vectơ nhúng duy nhất.

✓ **Kích thước mô hình (Model Size):** Đây là kích thước của mô hình tính bằng gigabyte.

Mô hình *lớn hơn* thường cho hiệu suất tốt hơn.

Tuy nhiên, chúng đòi hỏi nhiều tài nguyên tính toán hơn khi mở rộng Atlas Vector Search cho môi trường sản xuất.

✓ **Trung bình truy xuất (Retrieval Average):** Đây là một điểm số đo lường hiệu suất của các hệ thống truy xuất thông tin.

Điểm số *cao hơn* cho thấy mô hình xếp hạng các tài liệu liên quan cao hơn trong danh sách kết quả truy xuất tốt hơn.

Điểm số này đặc biệt quan trọng khi lựa chọn mô hình cho các ứng dụng RAG (Retrieval Augmented Generation - Sinh văn bản tăng cường bằng truy xuất). RAG là một kỹ thuật trong xử lý ngôn ngữ tự nhiên, kết hợp khả năng sinh văn bản của các mô hình ngôn ngữ lớn với khả năng truy xuất thông tin từ các nguồn bên ngoài.

Bước 2: Biểu diễn dữ liệu thành vector

Tạo Vector Embedding: Mỗi tài liệu (văn bản, hình ảnh, âm thanh) được chuyển đổi thành một vector số học. Vector này đại diện cho ngữ nghĩa của tài liệu đó trong không gian nhiều chiều.

Bước 3: Xây dựng cơ sở dữ liệu vector

+ Tự xây dựng trên local nhưng đây là một quá trình phức tạp, đòi hỏi sự hiểu biết về nhiều khía cạnh, từ lựa chọn công nghệ đến tối ưu hiệu suất.

+ Sử dụng các cơ sở dữ liệu chuyên dụng như Pinecone, Weaviate, Milvus, Qdrant, Chroma, hoặc sử dụng tính năng Atlas Vector Search trong MongoDB Atlas

Bước 4: Lưu trữ

Các vectơ được tạo ra từ bước 1 được lưu trữ trong một cơ sở dữ liệu vectơ (vector database) hoặc một hệ thống lưu trữ chuyên dụng. Cơ sở dữ liệu vectơ được tối ưu hóa để thực hiện các phép toán trên vectơ một cách hiệu quả, đặc biệt là tính toán khoảng cách (distance) hoặc độ tương đồng (similarity) giữa các vectơ.

Bước 5: Thực hiện truy vấn:

Đầu tiên câu truy vấn sẽ được vector hoá: Truy vấn của người dùng cũng được nhúng thành một vector (gọi là vector truy vấn).

Sau đó thực hiện tìm kiếm trong cơ sở dữ liệu: Hệ thống sẽ tìm kiếm các vector tài liệu trong cơ sở dữ liệu có khoảng cách gần nhất với vector truy vấn (tức là có độ tương đồng cao hay có ý nghĩa phù hợp nhất với truy vấn và ngữ cảnh tương ứng).

Cuối cùng xếp hạng các vector tài liệu tìm được và trả về kết quả: Các tài liệu được xếp hạng theo mức độ tương đồng từ cao xuống thấp và trả về cho người dùng.

c) Ưu và khuyết điểm

- **Ưu điểm của Vector Search:**

Kết quả tìm kiếm chính xác hơn: Vector Search có thể tìm thấy những kết quả có ý nghĩa và ngữ cảnh phù hợp với truy vấn, ngay cả khi không có từ khóa trùng khớp hoàn toàn nào.

Khả năng xử lý các loại dữ liệu khác nhau: Vector Search có thể được áp dụng để tìm kiếm trong nhiều loại dữ liệu khác nhau, không chỉ văn bản mà còn hình ảnh, âm thanh, video.

Khả năng mở rộng: Vector Search có thể dễ dàng mở rộng để xử lý các lượng dữ liệu lớn hoặc phức tạp.

- **Nhược điểm của Vector Search:**

Độ phức tạp: Việc xây dựng và quản lý một hệ thống Vector Search đòi hỏi kiến thức chuyên môn về xử lý ngôn ngữ tự nhiên, học máy và cơ sở dữ liệu.

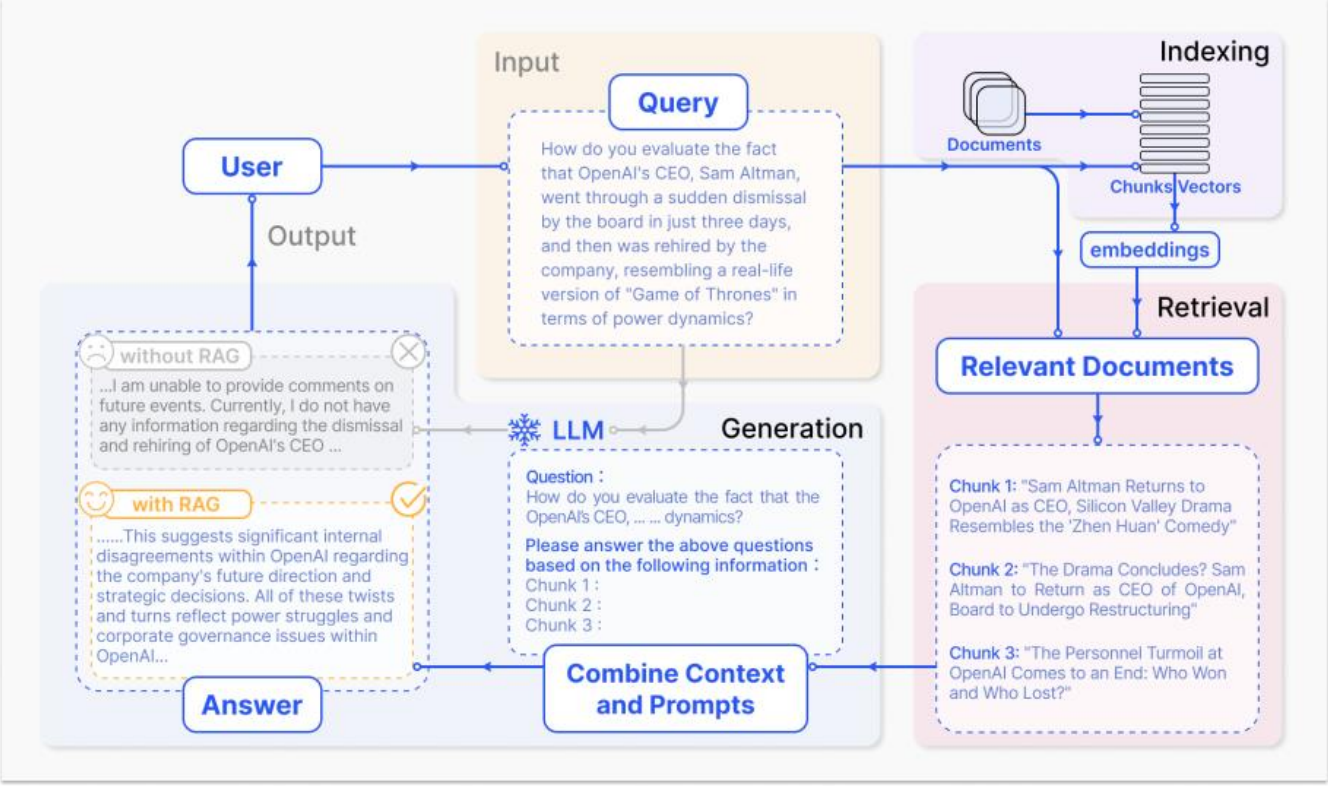
Chi phí tính toán: Việc tính toán khoảng cách giữa các vector có thể tiêu tốn nhiều tài nguyên tính toán.

2.1.4. Retrieval-augmented generation (RAG)

a) Khái niệm

Retrieval-Augmented Generation (RAG là một kỹ thuật được sử dụng để nâng cao khả năng của các mô hình ngôn ngữ lớn (LLM) bằng cách kết hợp kiến thức từ các cơ sở dữ liệu bên ngoài. Phương pháp này thực hiện bằng cách truy xuất thông tin liên quan từ kho lưu trữ tài liệu chứa kiến thức bên ngoài và sử dụng thông tin truy xuất được để câu trả lời thông qua LLM. Bằng cách tham chiếu kiến thức bên ngoài, RAG giúp giảm thiểu việc tạo ra nội dung không chính xác về mặt thực tế, ngăn chặn tình trạng quá tải thông tin cho LLM và nâng cao chất lượng phản hồi đáng kể.

Quy trình hoạt động cơ bản của mô hình RAG theo [7]:



Hình 6: Quy trình hoạt động của mô hình RAG

b) Phân loại

Theo như bài báo [7], mô hình RAG được phân thành 3 loại: Naive RAG, Advanced RAG, Modular RAG

+ Naive RAG

Là mô hình RAG đơn giản nhất với quy trình gồm 3 bước chính như sau:

1. Indexing (Lập chỉ mục):

Chuyển đổi dữ liệu từ nhiều định dạng (PDF, HTML, Word) thành văn bản thuần túy

Chia nhỏ văn bản thành các đoạn nhỏ hơn (chunks) để dễ xử lý và phù hợp với giới hạn ngữ cảnh của mô hình llm

Chuyển các đoạn văn thành vector tài liệu và lưu trong cơ sở dữ liệu vector đã được lập chỉ mục

2. Retrieval (Thu thập):

Vector hoá câu hỏi của người dùng thành vector truy vấn

Trong cơ sở dữ liệu vector (nơi lưu trữ vector tài liệu), tìm kiếm các vector tương đồng nhất với vector truy vấn bằng cách tính điểm tương đồng giữa vector truy vấn và vector tài liệu. Sau đó chọn ra top K vector tài liệu có liên quan nhất tương đương với điểm tương đồng cao nhất

3. Generation (Sinh nội dung):

Kết hợp câu truy vấn với các đoạn văn được biểu diễn bởi vector tài liệu vừa chọn được rồi tạo ra prompt mới

Sinh câu trả lời dựa trên prompt này

Tuy nhiên vì quá đơn giản nên Naive RAG tồn tại các hạn chế cần khắc phục như thu thập thông tin không chính xác hoặc thiếu sót; đôi khi tạo ra nội dung sai lệch (hiện tượng hallucination), nội dung không liên quan, độc hại hoặc thiên vị trong kết quả đầu ra, làm giảm chất lượng và độ tin cậy của câu trả lời; khó tích hợp thông tin một cách mạch lạc, việc tích hợp này cũng có thể xảy ra tình trạng dư thừa khi các thông tin tương tự được truy xuất từ nhiều nguồn khác nhau; Việc xác định mức độ quan trọng và mức độ liên quan của các đoạn khác nhau, đảm bảo tính nhất quán về phong cách và giọng điệu cũng làm tăng thêm sự phức tạp.

+ Advanced RAG

Là mô hình RAG được tạo ra nhằm khắc phục các điểm yếu của Naive RAG với 2 giai đoạn mới: Pre-retrieval và Post-retrieval

Pre-retrieval: là giai đoạn tối ưu cấu trúc indexing và câu truy vấn

Post-retrieval: là giai đoạn thực hiện các công việc gồm sắp xếp lại thứ tự các đoạn văn theo độ liên quan (reranking), nén dữ liệu về ngữ cảnh để tránh quá tải thông tin, giúp LLM chỉ tập trung vào các thông tin quan trọng bằng cách lọc bỏ các tài liệu không liên quan và cho phép LLM đánh giá nội dung được truy vấn trước khi LLM tạo phản hồi

+ Modular RAG

Là mô hình RAG tiên tiến nhất, được cải tiến từ 2 mô hình RAG trước đó. Bên cạnh các yếu tố được kế thừa từ Naive RAG và Advanced RAG, Modular RAG còn tích hợp nhiều chiến lược khác nhau để cải thiện các thành phần của nó, ví dụ như thêm mô-đun tìm kiếm để tìm kiếm sự tương đồng và tinh chỉnh bộ truy xuất thông tin; thêm RAG-Fusion để giải quyết các hạn chế tồn đọng trong các mô hình RAG trước đây; bổ sung nhiều module phụ trợ khác như Memory module, Routing, Predict module, Task Adapter module

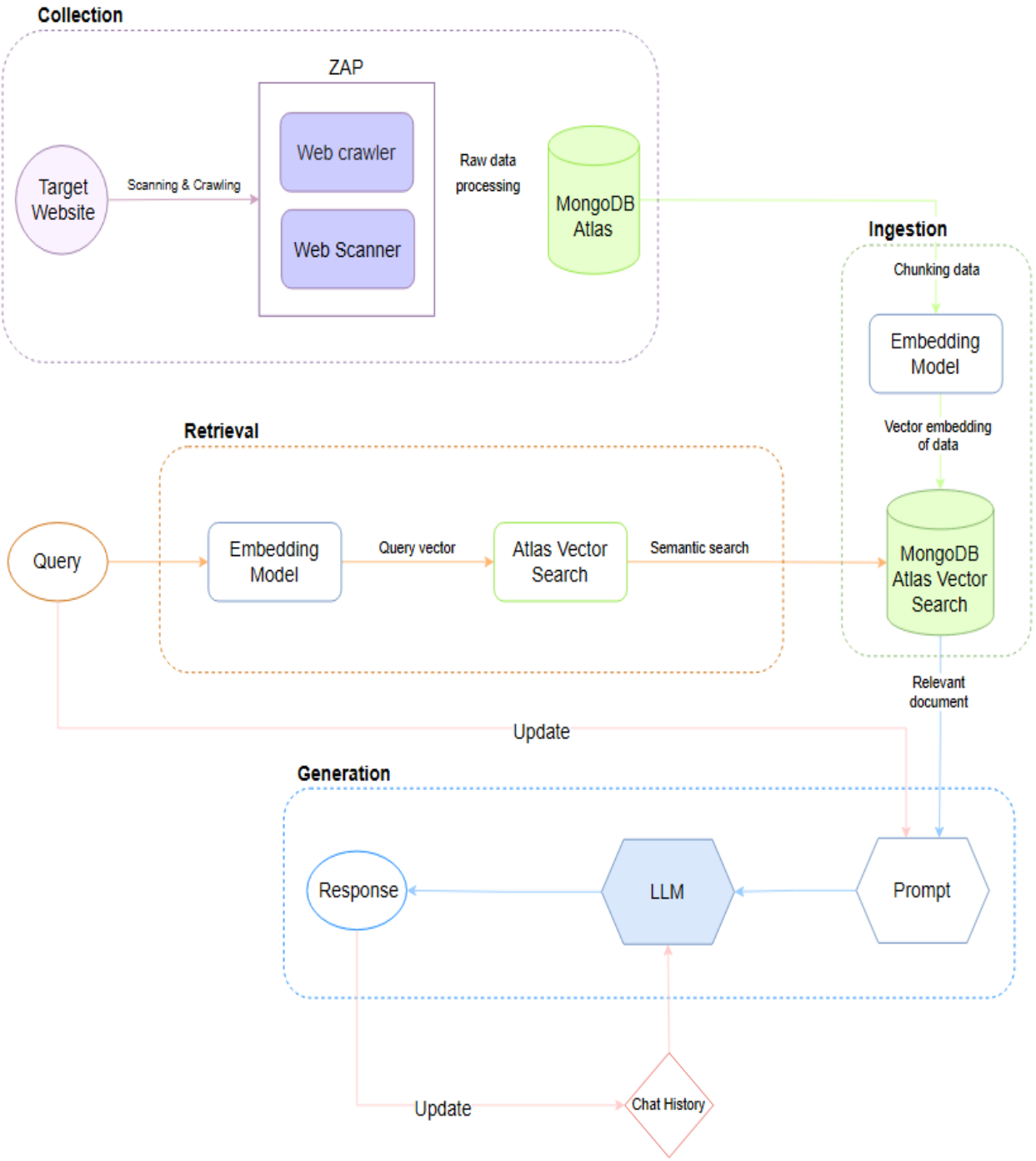
2.2 Các nghiên cứu liên quan

Bài báo [2] cung cấp cái nhìn tổng quan về các honeypot, framework và các công cụ mã nguồn mở được duy trì từ đó giúp chúng ta dễ hình dung hơn về các honeypot truyền thống và honeypot chuyên biệt. Ngoài ra bài báo này còn đề xuất các framewrok honeypot khá hay như Chameleonm, OWASP Honeypot Framework, T-Pot.

DecoyPot [8] là một honeypot giả lập Web API vận hành bởi mô hình ngôn ngữ lớn (LLM) nhằm thu hút những kẻ tấn công một cách chân thực. DecoyPot được thiết kế để bắt chước các API Web hợp pháp và phản hồi các yêu cầu API bằng các response có ý nghĩa và phù hợp ngữ cảnh nhằm đánh lừa những kẻ tấn công , khiến chúng tin rằng chúng đang tương tác với một hệ thống thực. Bên cạnh tích hợp LLM, DecoyPot còn sử dụng RAG để truy xuất thông tin liên quan từ tài liệu API và tạo phản hồi chính xác về ngữ cảnh.

Chương 3. PHÂN TÍCH THIẾT KẾ HỆ THỐNG

3.1. Kiến trúc hệ thống



Hình 7: Mô hình hệ thống nghiên cứu

3.1.1. Quy trình hoạt động

Bước 1: Collection

Web Scanner thực hiện Scaning để kiểm tra website và tìm kiếm, thu thập các lỗ hổng có thể có trong website

Web Crawler thực hiện Crawling để duyệt qua các trang web và lần theo các liên kết trong website nhằm thu thập dữ liệu về các trang web

Cuối cùng các dữ liệu do Web Crawler và Web Scanner thu thập được sẽ được lưu vào 2 collection khác nhau trong kho lưu trữ của MongoDB trên cloud (MongoDB Atlas)

Bước 2: Ingestion

Các dữ liệu trong mỗi collection sẽ được xử lý và chunking (tách thành các phần nhỏ hơn để cải thiện hiệu suất). Sau đó sử dụng mô hình nhúng (Embedding Model) để chuyển đổi chúng thành các Vector Embedding. Cuối cùng lưu trữ Vector Embedding vào kho lưu trữ cơ sở dữ liệu vector

Trước khi lưu trữ, em sẽ tạo collection mới rồi dùng tính năng Atlas Vector Search do mongoDB hỗ trợ để xây dựng nên kho lưu trữ cơ sở dữ liệu vector (MongoDB Atlas Vector Search)

Vì ở đây có 2 loại dữ liệu lưu trong 2 collection khác nhau nên em sẽ tạo thêm 2 kho lưu trữ cơ sở dữ liệu vector. Một kho chứa dữ liệu do Web Scanner thu thập và được dùng để tìm kiếm các tài liệu liên quan tới lỗ hổng của trang web. Kho dữ liệu còn lại sẽ lưu dữ liệu do Web Scanner thu thập được nhằm để tìm kiếm các mẫu trang web liên quan tới câu truy vấn

Bước 3: Retrieval

Sau khi nhận được câu truy vấn, Embedding Model sẽ thực hiện chuyển đổi câu truy vấn ấy thành vector truy vấn (Query Vector). Sau đó Atlas Vector Search sẽ tìm kiếm các Vector Embedding (lưu trong MongoDB Atlas Vector Search) tương đồng nhất về mặt ngữ nghĩa với Query Vector rồi trả về kết quả là nội dung của các tài liệu ứng với Vector Embedding đó

Bước 4: Generation

Tất cả các tài liệu liên quan tới câu truy vấn và cả chính nó đều được đưa vào prompt rồi cung cấp cho LLM với ngữ cảnh kèm theo để LLM sinh ra phản hồi phù hợp nhất

Phản hồi mới do LLM sinh ra (cũng như lịch sử giao tiếp mới với LLM) sẽ được cập nhập vào bộ đệm lưu trữ tạm thời (Chat History) để thông tin do LLM sinh ra phù hợp theo thời gian thực

3.2. Các thành phần trong mô hình

a) ZAP

ZAP là thành phần thực hiện thu thập các dữ liệu cần thiết để cải thiện hiệu quả sinh phản hồi của LLM. Đồng thời nó cũng sẽ thực hiện giám sát lưu lượng mạng

OWASP Zed Attack Proxy (có thể gọi tắt là ZAP) là một công cụ scanner miễn phí, phổ biến và được duy trì bởi hàng trăm nghìn tình nguyện viên trên toàn thế giới, nó cũng rất dễ cài đặt và hỗ trợ việc crawling nên em chọn Zed Attack Proxy đóng vai trò là ZAP trong mô hình hệ thống

b) MongoDB

Tất cả các dữ liệu phát sinh trong honeypot đều sẽ được lưu trữ trong cơ sở dữ liệu của MongoDB Atlas, một nơi lưu trữ dữ liệu tập trung trên đám mây

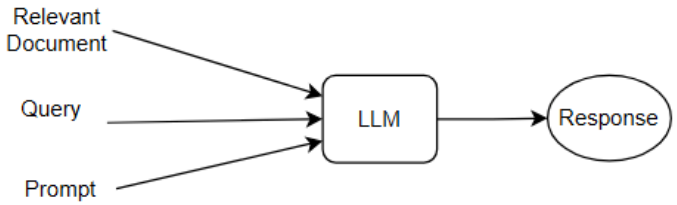
Ngoài ra MongoDB Atlas Vector Search còn có nhiệm vụ tìm kiếm các tài liệu liên quan tới câu truy vấn mà honeypot nhận được

c) LLM

Thành phần cốt lõi và quan trọng nhất của honeypot, có nhiệm tự động sinh ra các phản hồi theo dạng HTTP Message dựa trên prompt và các tài liệu liên quan, bên cạnh đó nội dung phản hồi cũng phải phù hợp với ngữ cảnh và đánh lừa được người dùng.

Các dữ liệu đầu vào đưa vào LLM gồm có: câu truy vấn, các dữ liệu liên quan tới câu truy vấn (được tìm thấy trong MongoDB Atlas bằng kỹ thuật Vector

Search). Vì đây là đề tài về Web Honeypot nên LLM được mong đợi tạo ra các phản hồi giống HTTP Messages nhất có thể và nó phải phù hợp với ngữ cảnh theo thời gian thực, tức là ngoài việc có nội dung tương ứng với câu truy vấn, các câu phản hồi mà LLM tạo ra còn cần có sự liên kết với các câu phản hồi và câu truy vấn trước đó trong một phiên tương tác với 1 người dùng.

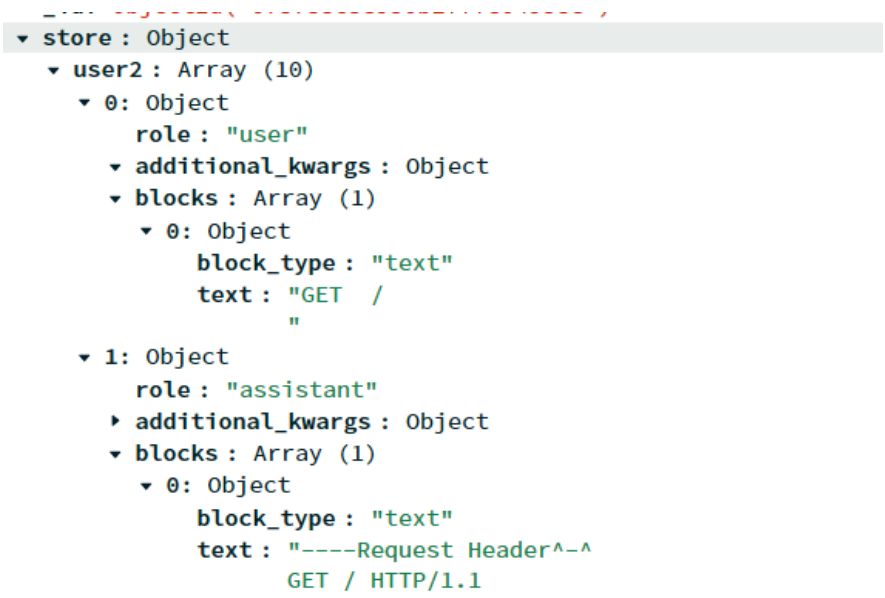


Hình 8: Dữ liệu cung cấp cho LLM

d) Chat History

Sau khi LLM sinh ra một phản hồi, toàn bộ dữ liệu về câu phản hồi, nội dung của nó và cả câu truy vấn dùng để sinh ra nó sẽ được lưu trong Chat History, thành phần được xây dựng dựa trên bộ nhớ đệm đơn giản giúp lưu trữ cuộc trò chuyện.

Mỗi cứ mỗi lần LLM sinh phản hồi thì một cặp document lại được lưu lại gồm document chứa dữ liệu của "user" và document chứa dữ liệu của "assistant" (tức chỉ llm)



Hình 9 : Cấu trúc lưu trữ lịch sử của Chat History

Chương 4. HIỆN THỰC HỆ THỐNG

4.1 Môi trường thực nghiệm

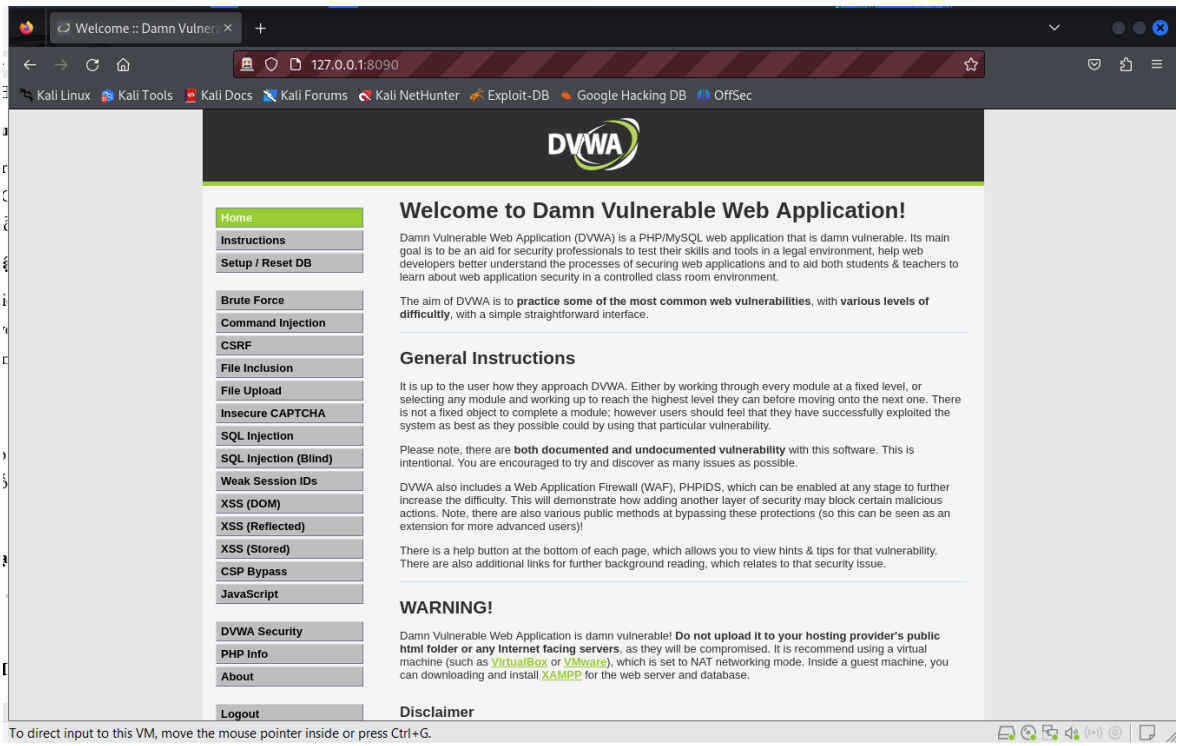
Triển khai trong môi trường Linux, sử dụng python3, thư viện của mongodb (pymongo), framework hỗ trợ LLM của LlamaIndex (llama-index) và các thư viện khác

4.2 Thu thập mẫu

a) Mẫu

Mô hình triển khai cần 2 loại dữ liệu bao gồm các thông điệp HTTP tương ứng với tấn công web, các thông điệp Request và Response trong quá trình tương tác với web. Vì vậy em thực hiện Web Scanning để thu thập thông tin về các lỗ hổng liên quan tới web application và Web crawling để thu thập các webpage của web application

Để tiện cho việc vừa thu thập các mẫu tấn công và mẫu webpage, em chọn trang DVWA làm đối tượng Web Scanning và Web crawling



b) Công cụ

Hình 10: Website mục tiêu

Sau khi chọn được website để honeypot giả lập, em dùng ZAP để thực hiện Web Scanning bằng chức năng Active Scan, Fuzzing và Manual Explore

Ngoài ra ZAP cũng hỗ trợ tính năng Traditional spider và AJAX spiders:

+ Traditional spider: Lần theo các liên kết trên trang web mục tiêu, hoạt động như một con nhện hỗ trợ phát hiện các liên kết URL bằng cách kiểm tra HTML trong các phản hồi từ một ứng dụng web; giúp tìm kiếm các trang mới, các form, các tham số và các phần tử khác trên trang web.

+ AJAX spider: Tìm kiếm các yêu cầu AJAX được thực hiện bởi trình duyệt, sau đó phân tích các yêu cầu này để tìm ra các tham số, các đường dẫn động và các chức năng ẩn.

Với 2 tính năng này chúng ta hoàn toàn có thể thực hiện Crawling do đó em tiến hành Web Crawling bằng ZAP để thu thập dữ liệu về các web page

4.3 Khởi tạo và xây dựng cơ sở dữ liệu vector

MongoDB Atlas là cơ sở dữ liệu đám mây của MongoDB với thiết kế giúp tăng tốc và đơn giản hoá cách xây dựng cơ sở dữ liệu và gần đây nó còn được bổ sung tính năng MongoDB Atlas Vector Search giúp tìm kiếm tài liệu dựa trên vector. Do đó em sử dụng MongoDB Atlas để xây dựng cơ sở dữ liệu vector và lưu trữ các tài liệu khác có liên quan.

a) Lưu trữ các mẫu đã thu thập

Các data raw thu thập được từ ZAP sẽ được xử lý để tạo ra mẫu hoàn chỉnh gồm 5 trường: responseBody, requestBody, responseHeader, requestHeader và url (riêng trường _id là trường định danh, tự động được tạo ra khi lưu document vào MongoDB)

Hình 11: : Cấu trúc document lưu trong MongoDB Atlas

b) Khởi tạo và xây dựng cơ sở dữ liệu vector

Chọn 1 collection trong MongoDB rồi thiết lập nó thành cơ sở dữ liệu vector có cấu trúc như definition thông qua lời gọi hàm create_search_index để tạo Atlas Vector Search Index

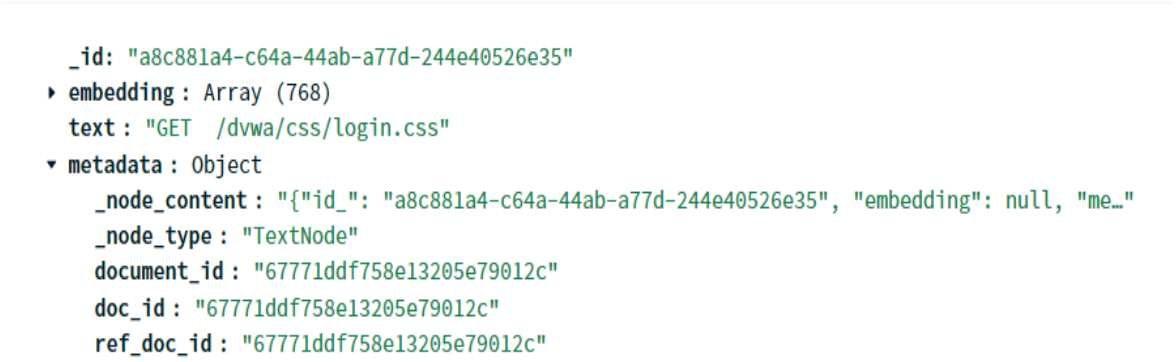
```
# Create your index model, then create the search index
search_index_model = SearchIndexModel(
definition={
    "fields": [
    {
        "type": "vector",
        "path": "embedding",
        "numDimensions": 768,
        "similarity": "cosine"
    },
    {
        "type": "filter",
        "path": "metadata._node_content"
    }
    ]
},
name="vector_index",
type="vectorSearch",
)

collection.create_search_index(model=search_index_model)
```

Hình 12: Thông số thiết lập cho cơ sở dữ liệu vector

4.4 Nhúng data và lưu các vector tài liệu vào cơ sở dữ liệu vector

Em viết một chương trình thực hiện tạo Data Embedding bằng Embedding Model, sau đó lưu vào collector được chọn làm cơ sở dữ liệu vector

Mỗi một Data Embedding sau khi lưu vào cơ sở dữ liệu vector sẽ được gọi là vector tài liệu và có cấu tạo như ảnh



Hình 13: Cấu trúc document lưu trữ Vector Embedding

4.5 Các bước triển khai

Bước 1: Dùng ZAP để thực hiện Web Scanning và Web Crawling khi thu thập data

Sử dụng chức năng Spider của ZAP để thực hiện crawling và lấy các trang web của website

Sử dụng chức năng Active Attack, Passive Scan và Manual Explore để thực hiện scanning và thu thập các dữ liệu liên quan tới lỗ hổng trong website

Bước 2: Lưu data bản raw vào collection trong MongoDB Atlas

Đầu tiên em sẽ xử lý lại các dữ liệu thu thập được để tinh gọn chúng và chỉ lấy các thông tin cần thiết. Sau đó phân loại và lưu chúng vào collection trong MongoDB Atlas:

+ Collection Response&Request lưu dữ liệu của Web Scanner

+ Collection Similar_Attack_Pattern lưu data của Web Crawler

Bước 3: Xây dựng cơ sở dữ liệu vector

Tạo 2 collection mới trong mongoDB rồi tiến hành cấu hình collection đó thành nơi lưu trữ các vector tài liệu. Để thực hiện việc này, em sử dụng tính năng Atlas Vector Search của MongoDB để xây dựng kho lưu trữ cơ sở dữ liệu vector gồm 2 kho:

+ Kho lưu trữ v1: sẽ là nơi tìm kiếm các mẫu tấn công tương đồng với câu truy vấn

+ Kho lưu trữ v2: sẽ là nơi tìm kiếm các mẫu trang web phù hợp với câu truy vấn

Bước 4: Thực hiện nhúng dữ liệu rồi lưu vào Collection dùng làm cơ sở dữ liệu vector

Đầu tiên, em chọn 1 mô hình nhúng phù hợp với kịch bản thực nghiệm. Sau đó em dùng mô hình này để chuyển đổi dữ liệu dạng chuỗi thành các vector (Vector Embedding). Cuối cùng là lưu các Vector Embedding này vào các kho lưu trữ đã tạo ở bước 3

Bước 5: Cấu hình mô hình LLM phù hợp với nhu cầu

Sau khi thử nghiệm nhiều lần thì em chọn các thông số như sau để cấu hình cho mô hình LLM:

Mô hình LLM thứ 1:

```
llm = Groq(model="llama3-8b-8192", api_key='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',
temperature=0.7,
presence_penalty= 0,
max_tokens=8192,
top_p= 0.95,
frequency_penalty= 0,
CallbackManager=Settings.callback_manager
)
```

Hình 14: Thông số mô hình LLM do Groq cung cấp

Mô hình LLM thứ 2:

```
""" #Using llm moders were gotten from g4f
llm = Custom(
    model="gpt-4o",
    provider="OpenaiChat",
    )
embed_model = HuggingFaceEmbedding(model_name="all-mpnet-base-v2")
Settings.embed_model = embed_model
"""
```

Hình 15: Thông số mô hình LLM do gpt4free cung cấp

Bước 6: Thực hiện truy vấn

Đưa câu truy vấn vào với định dạng:

{Method} {URL}

{Request Body}

```
+++
POST  /vulnerabilities/exec/
ip=127.0.0.1&Submit=Submit
+++
POST  /vulnerabilities/exec/
ip=google.com&Submit=Submit
+++
GET  /vulnerabilities/sqli_blind/

+++
GET  /vulnerabilities/sqli_blind/?id=123&Submit=Submit

+++
GET  /vulnerabilities/xss_d/

+++
GET  /vulnerabilities/xss_d/?default=English
```

Hình 16: Danh sách các mẫu truy vấn

Chương 5. THỰC NGHIỆM VÀ ĐÁNH GIÁ

5.1 Hiệu suất tìm kiếm mẫu tương đồng bằng Atlas Vector Search

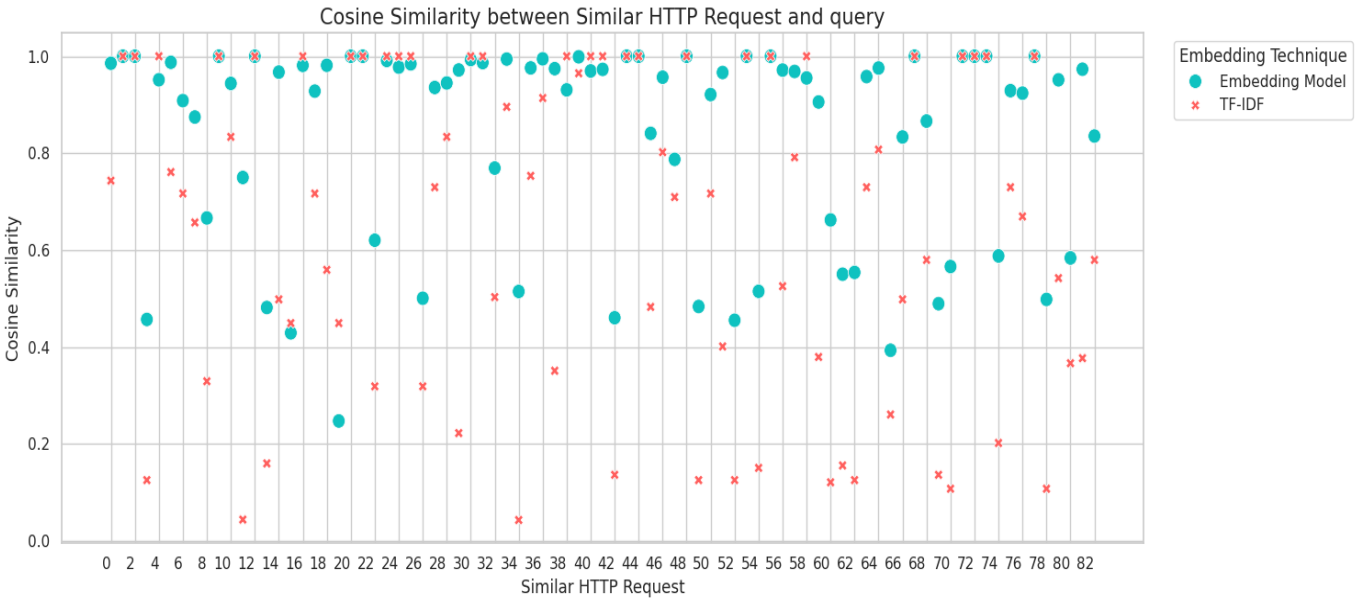
Để đánh giá mức độ tìm kiếm mẫu tương đồng bằng Atlas Vector Search, em đã thực hiện tìm kiếm mẫu bằng 83 câu truy vấn mới mà mình thu thập được. Kết quả thu được là 83 câu HTTP Request tương ứng với 83 câu truy vấn. Sau đó em sử dụng Embedding Model để vector hoá chúng rồi tính toán độ tương đồng bằng cosine similarity, và với sự kết hợp này, điểm đánh giá sẽ tập trung thể hiện mức độ tương đồng về mặt ngữ nghĩa giữa câu truy vấn và mẫu tương đồng tìm được. Embedding Model có rất nhiều loại nhưng trong nghiên cứu của mình em chọn dùng all-mpnet-base-v2, đây là mô hình sentence-transformers (còn gọi là SBERT) [9]. Mô hình này ánh xạ các câu và đoạn văn thành không gian vectơ dày đặc 768 chiều và có thể được sử dụng cho các tác vụ như phân cụm hoặc tìm kiếm ngữ nghĩa.

Ngoài ra, em cũng đánh giá dựa trên tần suất xuất hiện từ vựng (không quan tâm tới ngữ nghĩa, nội dung văn bản). Với cách đánh giá này, em lựa chọn kỹ thuật nhúng TF-IDF [10] (một phương pháp tính toán trọng số của từng từ trong văn bản) để thực hiện vector hoá sau đó tính toán cosine similarity giữa 2 vector

Bảng 1: Thống kê dữ liệu dùng để đánh giá hiệu suất tìm kiếm mẫu tương đồng

	Số câu truy vấn	Mẫu tìm kiếm
Collection	test	Similar_Attack_Pattern
Số lượng Document	83	Similar_Attack_Pattern: 250
test: tập hợp các mẫu để đánh giá kết quả phản hồi của llm Request&Response: tập hợp các mẫu do Web Crawler thu thập Similar_Attack_Pattern: tập hợp các mẫu do Web Scanner thu thập		

Để tiện quan sát, đánh giá kết quả, em dùng biểu đồ phân tán Scatter plot để trực quan hoá kết quả như hình 15:



Hình 17: Biểu đồ biểu diễn mức độ tương quan giữa câu truy vấn và HTTP Request tìm được trong kho lưu trữ dữ liệu

Dưới đây sẽ là bảng thống kê kết quả thu được khi so sánh câu truy vấn với mẫu tương đồng tìm được bằng vector search:

Bảng 2: So sánh câu truy vấn với HTTP Request tìm được dựa trên sự tương đồng ngữ nghĩa

Index	query	Similar HTTP Request	score
0	GET /vulnerabilities/xss_r/?name=+%3CScript%3Ealert%28%E2%80%9Cbingo%E2%80%9D%29%3C%2FScript%3E	GET /vulnerabilities/xss_r/?name=%3E%20%3Cscript%3Ealert(4)%3C/script%3E	0.983

1	GET /vulnerabilities/xss_d	GET /vulnerabilities/xss_d/	0.993
2	GET /dvwa/css	GET /vulnerabilities/xss_d/?default=&%2360;LINK%20REL&%2361;&%2334;stylesheet&%2334;%20HREF&%2361;&%2334;http&%2358;&%2347;&%2347;testsite.com&%2347;xss.css&%2334;&%2362;	0.783
3	GET /vulnerabilities/fi	GET /vulnerabilities/fi/?page=include.php	0.917
4	GET /vulnerabilities/xss_d/?default=%3Cscript%3Ealert(%22Hello%22)%3C/script%3E	GET /vulnerabilities/xss_d/?default=%3E%20%3Cscript%3Ealert(4)%3C/script%3E	0.992
5	GET /vulnerabilities/weak_id	GET /vulnerabilities/weak_id/	0.996
6	POST /vulnerabilities/weak_id/	GET /vulnerabilities/weak_id/	0.884
7	POST /vulnerabilities/xss_s/ txtName=noname&mtxMessage=%3Cscript%3Ealert%28%22Bingo%22%29%3C%2Fscript%3E&btnSign=Sign+Guestbook	POST /vulnerabilities/xss_s/ txtName=noname&mtxMessage=%3Cscript%3Ealert%28document.domain%29%3C%2Fscript%3E&btnSign=Sign+Guestbook	0.987
8	GET /dvwa/css/main.css	GET /vulnerabilities/xss_d/?default=%3CSTYLE%20type=%22text/css%22%3EBODY%7Bbackground:url(%22javascript:alert('XSS')%22)%	0.775

		7D%3C/STYLE%3E	
9	GET /vulnerabilities/brute/?username=bon&password=4321&Login=Login	GET /vulnerabilities/brute/?username=1&password=2&Login=Login	0.972
10	GET /vulnerabilities/fi/?page=file3.php	GET /vulnerabilities/fi/?page=include.php	0.963

Bảng 3: So sánh câu truy vấn với HTTP Request tìm được dựa trên sự tương đồng từ vựng

Index	query	Similar HTTP Request	score
0	GET /vulnerabilities/xss_d/?default=%3Cscript%3Ealert(%22Hello%22)%3C/script%3E	GET /vulnerabilities/xss_d/?default=%3E%20%3Cscript%3Ealert(4)%3C/script%3E	0.7433
1	GET /vulnerabilities/javascript/	GET /vulnerabilities/javascript/	1
2	GET /vulnerabilities/csp/	GET /vulnerabilities/csp/	1
3	POST /vulnerabilities/exec/ip=127.0.0.1%3B+echo+%22my+friends%22&Submit=Submit	POST /vulnerabilities/javascript/token=8b479aefbd90795395b3e7089ae0dc09&phrase=<	0.1254
4	GET /vulnerabilities/javascript	GET /vulnerabilities/javascript/	1

5	GET /vulnerabilities/xss_d/?default=%3Cscript%3Ealert(%22XSS%20DOM%22)%3C/script%3E	GET /vulnerabilities/xss_d/?default=%22%3E%3Cscript%3Ealert('XSS')%3C/script%3E	0.761
6	GET /vulnerabilities/fi/?page=file2.php	GET /vulnerabilities/fi/?page=include.php	0.7168
7	GET /vulnerabilities/xss_r/?name=bingo	GET /vulnerabilities/xss_r/	0.657
8	POST /vulnerabilities/exec/ ip=google.com&Submit=Submit	POST /vulnerabilities/javascript/ token=8b479aefbd90795395b3e7089ae0dc09&phrase=<A HREF="//google">XSS</A>&send=Submit	0.3294
9	GET /vulnerabilities/sqli/	GET /vulnerabilities/sqli/	1
10	GET /vulnerabilities/brute/?username=bon&password=bon1234&Login=Login	GET /vulnerabilities/brute/?username=1&password=2&Login=Login	0.8336

5.2 Hiệu suất phản hồi của LLM

Việc đánh giá được thực hiện trên 50 cặp response khác nhau (gồm Captured HTTP response và LLM-generated response). Tương tự như khi đánh giá hiệu suất tìm kiếm mẫu tương đồng, ở đây em cũng đánh giá với 2 tiêu chí:

+ Độ tương đồng về mặt ngữ nghĩa, phù hợp với ngữ cảnh:

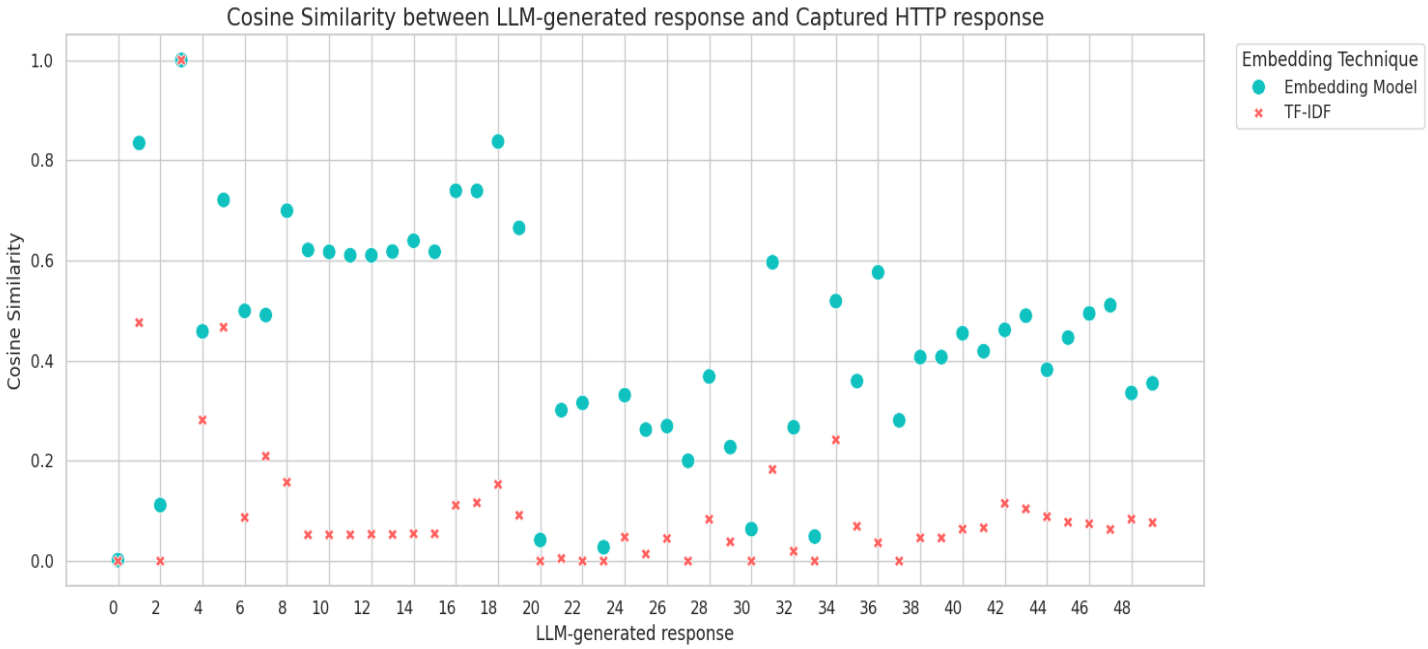
Được tính toán bằng cách sử dụng Embedding Model để vector từng câu trong văn bản rồi tính độ tương đồng bằng cosine similarity.

+ Độ tương đồng về tần suất các từ vựng trong văn bản:

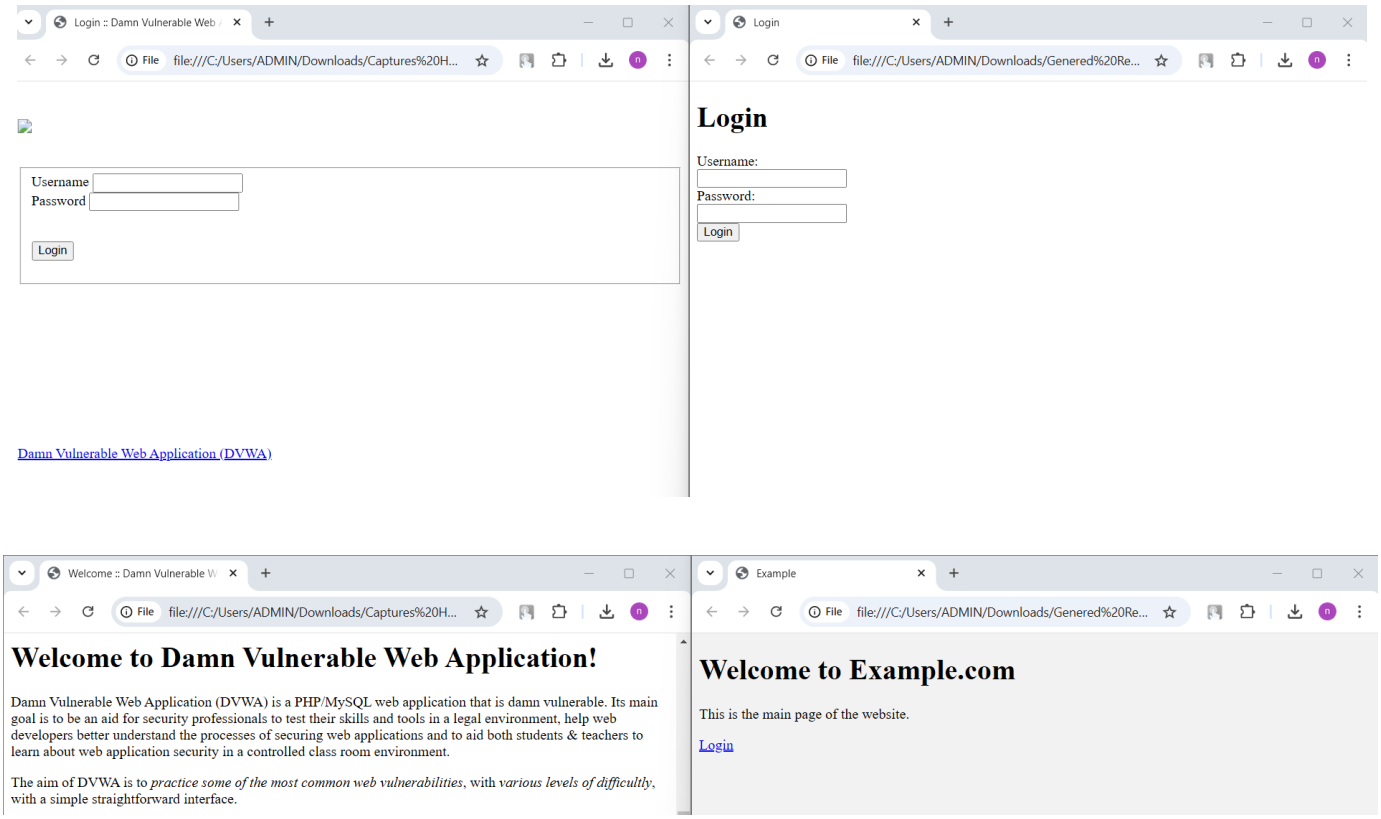
Vì cần so sánh từng từ một nên em dùng TF-IDF để vector từng từ một rồi thực hiện tính độ tương đồng giữa 2 văn bản bằng cosine similarity

Bảng 4: Thống kê dữ liệu dùng để đánh giá hiệu suất phản hồi của LLM

	Mẫu dùng để đánh giá	Mẫu được lưu trữ để cung cấp cho LLM
Collection	test	Request&Response, Similar_Attack_Pattern
Số lượng Document	50	Request&Response: 530 Similar_Attack_Pattern: 2236
test: tập hợp các mẫu để đánh giá kết quả phản hồi của llm Request&Response: tập hợp các mẫu do Web Crawler thu thập Similar_Attack_Pattern: tập hợp các mẫu do Web Scanner thu thập		



Hình 18: Biểu đồ biểu diễn mức độ tương quan giữa phản hồi do LLM sinh ra và HTTP Response bắt được ứng với câu truy vấn



Hình 19: Trang web nhận được tương ứng với HTTP Response. Ảnh bên phải là từ Response thực, ảnh bên trái là từ response giả do llm tạo ra

Chương 6. KẾT LUẬN VÀ HƯỚNG PHÁT TRIỂN

6.1 Kết luận

Từ các kết quả đánh giá nhận được, mô hình này còn khá kém trong việc tạo ra các phản hồi có độ tương đồng về mặt hình thức, tức là các HTTP Response sinh ra sẽ có bố cục khác với HTTP Response thực sự. Tuy nhiên khi xét mức độ tương đồng về mặt ngữ nghĩa thì kết quả lại khả quan hơn.

6.2 Hướng phát triển

Nguyên nhân phản hồi do mô hình sinh ra còn kém về mặt từ vựng có thể là do lệnh nhắc và các tài liệu cung cấp cho LLM chưa được chi tiết. Do đó em sẽ cải thiện việc xử lý dữ liệu để chỉ trích xuất các đặc trưng cần thiết nhưng vẫn giữ yếu tố khái quát cho mô hình và tìm hiểu cách viết prompt hiệu quả hơn.

Bên cạnh đó mô hình này vẫn còn bị hạn chế về kích thước thông điệp truy vấn LLM và thời gian sinh phản hồi còn khá chậm nhưng vấn đề này có thể giải quyết nếu chọn mô hình AI phiên bản thương mại.

TÀI LIỆU THAM KHẢO

[1] M. Nawrocki, M. Wählisch, T. C. Schmidt, C. Keil, and J. Schönfelder, "A Survey on Honeypot Software and Data Analysis," Aug. 22, 2016, *arXiv*: arXiv:1608.06249. doi: 10.48550/arXiv.1608.06249.

[2] N. Ilg, P. Duplys, D. Sisejkovic, and M. Menth, "A survey of contemporary open-source honeypots, frameworks, and tools," *J. Netw. Comput. Appl.*, vol. 220, p. 103737, Nov. 2023, doi: 10.1016/j.jnca.2023.103737.

[3] "What is vector search? | IBM." Accessed: Jan. 11, 2025. [Online]. Available: https://www.ibm.com/think/topics/vector-search

[4] "What are Vector Embeddings? Applications, Use Cases & More," DataStax. Accessed: Jan. 11, 2025. [Online]. Available: https://www.datastax.com/guides/what-is-a-vector-embedding

[5] C. Fu *et al.*, "MME-Survey: A Comprehensive Survey on Evaluation of Multimodal LLMs," Dec. 08, 2024, *arXiv*: arXiv:2411.15296. doi: 10.48550/arXiv.2411.15296.

[6] N. Muennighoff, N. Tazi, L. Magne, and N. Reimers, "MTEB: Massive Text Embedding Benchmark," in *Proceedings of the 17th Conference of the European Chapter of the Association for Computational Linguistics*, A. Vlachos and I. Augenstein, Eds., Dubrovnik, Croatia: Association for Computational Linguistics, May 2023, pp. 2014–2037. doi: 10.18653/v1/2023.eacl-main.148.

[7] Y. Gao *et al.*, "Retrieval-Augmented Generation for Large Language Models: A Survey," Mar. 27, 2024, *arXiv*: arXiv:2312.10997. doi: 10.48550/arXiv.2312.10997.

[8] A. Sezgin and A. Boyacı, "Decoypot: A Large Language Model-Driven Web Api Honeypot for Realistic Attacker Engagement," Nov. 04, 2024, *Social Science Research Network, Rochester, NY*: 5009535. doi: 10.2139/ssrn.5009535.

[9] N. Reimers and I. Gurevych, "Sentence-BERT: Sentence Embeddings using Siamese BERT-Networks," Aug. 27, 2019, *arXiv*: arXiv:1908.10084. doi: 10.48550/arXiv.1908.10084.

[10] A. Aizawa, "An information-theoretic perspective of tf–idf measures," *Inf. Process. Manag.*, vol. 39, no. 1, pp. 45–65, Jan. 2003, doi: 10.1016/S0306-4573(02)00021-3.