ĐẠI HỌC QUỐC GIA HÀ NỘI
TRƯỜNG ĐẠI HỌC CÔNG NGHỆ
VIỆN TRÍ TUỆ NHÂN TẠO

BÀI TẬP LỚN MÔN HỌC

KỸ THUẬT VÀ CÔNG NGHỆ DỮ LIỆU LỚN

Hệ thống Big Data xử lý ảnh X-quang ngực với cơ chế ưu tiên bệnh lý

Sinh viên thực hiện:
Hoàng Văn Dương
Nguyễn Văn Huy
Nguyễn Gia Huy

Giảng viên hướng dẫn:
TS. Trần Hồng Việt
ThS. Ngô Minh Hương

Hà Nội, tháng 11 năm 2025

Mở đầu

Sự gia tăng nhanh chóng của dữ liệu y tế và nhu cầu chẩn đoán chính xác đặt ra nhiều thách thức cũng như cơ hội cho việc ứng dụng công nghệ dữ liệu lớn trong y học. Báo cáo này xây dựng một pipeline có khả năng mở rộng và hoạt động theo thời gian thực để phân tích ảnh X-quang ngực, đồng thời lưu trữ và xử lý dữ liệu bệnh lý, sử dụng môi trường dữ liệu lớn bao gồm Apache Kafka, Hadoop File System, Apache Spark và MongoDB.

Bằng cách tận dụng dữ liệu luồng từ hệ thống lưu trữ ảnh y tế, chúng tôi tiến hành phân tích dữ liệu để huấn luyện mô hình học sâu có khả năng phát hiện và ưu tiên các ca bệnh nguy cấp. Các kỹ thuật chính bao gồm xây dựng pipeline để xử lý dữ liệu có khối lượng lớn và tốc độ cao, trích xuất đặc trưng từ ảnh X-quang, và triển khai mô hình dự đoán tiên tiến trên Spark nhằm đảm bảo khả năng phân tán và chịu lỗi tốt.

Kết quả cho thấy hệ thống có thể cung cấp phân tích thời gian thực với độ trễ thấp, đồng thời hỗ trợ bác sĩ trong việc ưu tiên chẩn đoán các ca bệnh quan trọng. Báo cáo này mang lại những hiểu biết có giá trị về việc xây dựng pipeline dữ liệu lớn trong lĩnh vực y tế và gợi mở các hướng phát triển tiềm năng cho ứng dụng trí tuệ nhân tạo trong chẩn đoán hình ảnh.

Mục lục

Chương 1: Tổng quan

1.1 Giới thiệu

1.1.1 Bối cảnh hiện nay

Hiện nay, các bệnh lý về đường hô hấp và phổi như viêm phổi, lao hay các tổn thương phổi nghiêm trọng vẫn chiếm tỷ lệ cao trong cơ cấu bệnh tật toàn cầu. Tại các bệnh viện tuyến đầu, số lượng bệnh nhân đến chụp X-quang lồng ngực mỗi ngày là rất lớn, tạo ra áp lực đáng kể lên đội ngũ bác sĩ chẩn đoán hình ảnh và hệ thống lưu trữ dữ liệu y tế.

Trong bối cảnh nhiều cơ sở y tế thường xuyên đối mặt với tình trạng quá tải, quy trình tiếp nhận và xử lý hình ảnh X-quang ở khu khám bệnh thông thường vẫn chủ yếu hoạt động theo cơ chế trả kết quả tuần tự theo thời điểm bệnh nhân chụp. Điều này làm phát sinh một luồng dữ liệu hình ảnh lớn với tốc độ tăng cao, gây chậm trễ trong việc phát hiện các trường hợp nguy cơ cao nếu bác sĩ chưa kịp đọc phim. Bài toán đặt ra là cần ứng dụng công nghệ dữ liệu lớn và mô hình học sâu để tự động phân tích, sàng lọc và ưu tiên những ca bệnh nghiêm trọng nhằm tối ưu hóa quy trình chẩn đoán.

1.1.2 Vấn đề đặt ra

Mặc dù quy mô dữ liệu và số lượng bệnh nhân ngày càng tăng, nhưng phương thức vận hành truyền thống đang bộc lộ những hạn chế nghiêm trọng trong việc quản lý mức độ khẩn cấp của người bệnh:

- Mặc dù các bệnh viện hiện nay đã áp dụng cơ chế phân loại cấp cứu (triage) để ưu tiên bệnh nhân nguy kịch, nhưng quá trình này vẫn phụ thuộc nhiều vào đánh giá lâm sàng ban đầu. Những trường hợp có bệnh lý nặng nhưng biểu hiện bên ngoài không rõ ràng, như tràn khí màng phổi hoặc tổn thương phổi cấp, có thể bị phân loại vào nhóm ưu tiên thấp hơn và phải chờ đợi trong khi bác sĩ chưa kịp xem phim X-ray. Trong khi đó, nguồn lực của bác sĩ lại bị tiêu tốn cho việc xử lý các trường hợp nhẹ hoặc khám định kỳ những bệnh nhân đến trước nhưng không thực sự khẩn cấp.

- Nguy cơ bỏ lỡ thời gian vàng điều trị: Đối với các ca bệnh phổi nghiêm trọng, việc chậm trễ trong chẩn đoán dù chỉ vài chục phút cũng có thể dẫn đến các biến chứng nặng nề hoặc tử vong. Việc hồ sơ bệnh án khẩn cấp bị lẫn lộn trong hàng trăm hồ sơ bệnh nhẹ khiến bác sĩ khó có thể nhận diện ngay lập tức ai là người cần được can thiệp trước.

- Rủi ro trong lưu trữ dữ liệu cục bộ: Nhiều cơ sở y tế vẫn lưu trữ dữ liệu ảnh trên các máy chủ đơn lẻ hoặc máy trạm cục bộ. Phương thức này tiềm ẩn rủi ro mất mát dữ liệu cao khi xảy ra sự cố phần cứng và hạn chế khả năng mở rộng khi dữ liệu tích lũy lớn dần theo thời gian.

1.1.3 Tầm quan trọng của dự án

Việc nghiên cứu và xây dựng một hệ thống Big Data có khả năng dự đoán và tự động sắp xếp thứ tự ưu tiên bệnh nhân trong quy trình chẩn đoán hình ảnh X-quang lồng ngực mang lại những giá trị thiết thực trên nhiều phương diện, góp phần nâng cao chất lượng chăm sóc y tế và tối ưu hóa hoạt động của các cơ sở y tế.

- Hỗ trợ cứu sống bệnh nhân: Trong các bệnh lý phổi nghiêm trọng như tràn khí màng phổi, suy hô hấp cấp hay viêm phổi nặng, thời gian vàng để can thiệp có thể chỉ tính bằng vài chục phút. Hệ thống tự động phân tích hình ảnh X-quang ngay sau khi chụp giúp nhận diện nhanh các dấu hiệu bệnh lý nguy hiểm, từ đó tự động đẩy hồ sơ bệnh nhân vào hàng ưu tiên cao nhất. Điều này giúp bác sĩ có thể can thiệp kịp thời, giảm nguy cơ biến chứng nặng hoặc tử vong, đồng thời nâng cao khả năng cứu sống các ca bệnh nguy kịch mà nếu chỉ dựa vào đánh giá thủ công có thể bị bỏ sót hoặc chậm trễ.

- Đảm bảo công bằng trong y tế: Một trong những thách thức hiện nay là sự phân bổ nguồn lực y tế chưa tối ưu, đặc biệt khi số lượng bệnh nhân lớn và thời gian làm việc của bác sĩ hạn chế. Hệ thống tự động giúp đánh giá mức độ nghiêm trọng của bệnh dựa trên dữ liệu hình ảnh, thay vì chỉ dựa vào thời gian đến khám. Nhờ đó, các bệnh nhân thực sự cần được ưu tiên sẽ được chăm sóc trước, đảm bảo sự công bằng trong tiếp cận dịch vụ y tế, đồng thời nâng cao hiệu quả sử dụng nguồn lực của bệnh viện.

- Tự động hóa và tối ưu hóa quy trình vận hành: Việc xử lý và quản lý hàng trăm, thậm chí hàng nghìn hồ sơ X-quang mỗi ngày đặt ra yêu cầu cao về tốc độ và độ chính xác. Hệ thống Big Data kết hợp với mô hình học sâu giúp tự động hóa quá trình phân tích, sắp xếp và lưu trữ dữ liệu, giảm thiểu các thao tác thủ công, hạn chế sai sót và rút ngắn thời gian từ lúc chụp ảnh đến khi bác sĩ nhận được kết quả. Điều này không chỉ làm tăng năng suất làm việc của nhân viên y tế mà còn tạo ra một luồng công việc khép kín, có thể mở rộng khi khối lượng dữ liệu tăng lên trong tương lai.

- Đóng góp cho nghiên cứu và phát triển y tế thông minh: Hệ thống này còn cung cấp một nguồn dữ liệu lớn, được chuẩn hóa và có thể sử dụng cho các nghiên cứu y học, từ phân tích dịch tễ học đến phát triển các mô hình dự đoán bệnh lý phổi mới. Việc áp dụng Big Data và AI trong y tế không chỉ giúp giải quyết vấn đề thực tiễn mà còn thúc đẩy xu hướng y tế thông minh, hiện đại hóa công tác chẩn đoán và quản lý bệnh nhân trong các bệnh viện hiện nay.

Như vậy, dự án không chỉ có ý nghĩa về mặt cấp cứu và chăm sóc bệnh nhân, mà còn nâng cao chất lượng vận hành và quản lý dữ liệu y tế, đồng thời đóng góp vào sự phát triển bền vững của hệ thống y tế hiện đại.

1.1.4 Thách thức

Để hiện thực hóa hệ thống Big Data phân tích và xếp hàng ưu tiên bệnh nhân X-quang, dự án phải đối mặt với nhiều thách thức quan trọng về công nghệ và vận hành:

- Yêu cầu xử lý gần thời gian thực: Hệ thống cần xử lý luồng dữ liệu liên tục với độ trễ thấp, đảm bảo các cảnh báo ưu tiên được phát ra ngay sau khi ảnh được chụp. Do ảnh X-quang có dung lượng lớn, việc truyền trực tiếp qua các kênh streaming là khó thực hiện. Giải pháp là lưu trữ ảnh trực tiếp vào HDFS và chỉ truyền metadata nhẹ (bao gồm đường dẫn HDFS, thông tin bệnh nhân, thời gian chụp...) qua Kafka. Spark Streaming sẽ nhận metadata, truy cập HDFS theo đường dẫn ảnh và tiến hành phân tích, giúp dự đoán bệnh lý gần như tức thì.

- Độ chính xác của mô hình AI: Việc tự động xếp hàng ưu tiên bệnh nhân đòi hỏi mô hình học sâu phải đạt độ tin cậy cao trong chẩn đoán 14 loại bệnh lý phổi phổ biến. Hệ thống cần hạn chế tối đa báo động giả (false positive) gây nhiễu loạn cho bác sĩ và bỏ sót ca nguy kịch (false negative), đảm bảo bệnh nhân được can thiệp đúng mức ưu tiên.

- Khả năng tích hợp và mở rộng: Hệ thống phải có kiến trúc linh hoạt để tích hợp với thiết bị chụp X-quang trong tương lai hoặc các nguồn dữ liệu giả lập. Đồng thời, kho lưu trữ HDFS và các module xử lý dữ liệu phải có khả năng mở rộng khi số lượng ảnh tăng, đảm bảo tính chịu lỗi, bảo mật và sẵn sàng của dữ liệu y tế nhạy cảm.

- Đồng bộ và quản lý dữ liệu: Cần đảm bảo metadata và ảnh trong HDFS luôn đồng bộ, tránh tình trạng metadata tồn tại nhưng ảnh chưa kịp lưu hoặc path lỗi. Đồng thời, Spark Streaming phải truy cập HDFS hiệu quả để đảm bảo tốc độ xử lý, tránh tắc nghẽn khi số lượng metadata lớn.

Như vậy, hệ thống vừa tận dụng HDFS cho ảnh lớn, vừa tận dụng Kafka cho metadata nhẹ để xử lý gần thời gian thực, đồng thời vẫn đáp ứng yêu cầu chẩn đoán chính xác, xếp ưu tiên bệnh nhân và mở rộng quy mô.

1.1.5 Mục tiêu của dự án

Dựa trên các vấn đề và thách thức kỹ thuật đã phân tích, dự án này hướng tới hiện thực hóa bốn mục tiêu chính, với luồng dữ liệu X-quang được giả lập để mô phỏng hoạt động thực tế của bệnh viện:

- Tự động hóa luồng thu thập dữ liệu giả lập: Xây dựng các tác nhân phần mềm tạo và đưa ảnh X-quang vào hệ thống như thể chúng được chụp trực tiếp từ máy chụp, đảm bảo luồng dữ liệu liên tục để mô phỏng quá trình vận hành thời gian thực.

- Xây dựng kho lưu trữ phân tán an toàn: Triển khai hệ thống lưu trữ phân tán Hadoop HDFS để quản lý khối lượng lớn ảnh y tế giả lập, đảm bảo khả năng mở rộng, chịu lỗi và bảo mật dữ liệu, mô phỏng điều kiện lưu trữ trong thực tế.

- Phân tích bệnh lý theo thời gian thực: Áp dụng Spark Streaming kết hợp mô hình học sâu DenseNet121 để phân tích và nhận diện 14 loại bệnh lý phổi phổ biến từ dữ liệu giả lập ngay khi chúng đi vào hệ thống, từ đó kiểm chứng khả năng chẩn đoán và cảnh báo ưu tiên.

- Phát triển cơ chế hàng đợi ưu tiên thông minh: Thiết lập thuật toán tính điểm ưu tiên dựa trên mức độ nghiêm trọng của bệnh lý và thời gian chờ đợi từ dữ liệu giả lập, đồng thời xây dựng giao diện trực quan giúp bác sĩ hoặc người mô phỏng nhận diện nhanh các ca bệnh khẩn cấp, kiểm tra hiệu quả của hệ thống trong điều kiện thử nghiệm.

1.2 Các nghiên cứu liên quan

Trong những năm gần đây, việc ứng dụng trí tuệ nhân tạo và học sâu trong chẩn đoán hình ảnh y tế, đặc biệt là X-quang lồng ngực, đã trở thành một hướng nghiên cứu quan trọng nhằm nâng cao chất lượng chẩn đoán, tối ưu hóa quy trình làm việc và rút ngắn thời gian can thiệp đối với bệnh nhân nguy cấp. Các nghiên cứu liên quan có thể được phân loại thành ba nhóm chính: phát triển mô hình học sâu cho chẩn đoán hình ảnh, triển khai pipeline xử lý dữ liệu và tích hợp hệ thống trong môi trường lâm sàng, và tối ưu hóa workflow nhằm ưu tiên bệnh nhân nguy kịch.

- **Mô hình học sâu cho chẩn đoán X-quang:** Hwang và cộng sự đã đánh giá mức độ hiệu quả của một mô hình deep learning thương mại trong nhận diện các bất thường có ý nghĩa lâm sàng trên X-quang lồng ngực tại phòng cấp cứu. Nghiên cứu sử dụng dữ liệu của hơn một nghìn bệnh nhân và đo lường hiệu suất thông qua ROC, độ nhạy và độ đặc hiệu. Kết quả cho thấy mô hình đạt AUC là 0.95 và cải thiện độ nhạy của các bác sĩ nội trú khi sử dụng đầu ra của mô hình, minh chứng cho khả năng hỗ trợ chẩn đoán gần thời gian thực. (Hwang et al., 2019) [1]

- **Pipeline xử lý dữ liệu và triển khai ML thời gian thực:** Kathiravelu và cộng sự phát triển Niffler, một framework DICOM mã nguồn mở, cho phép triển khai các pipeline machine learning và xử lý hình ảnh trên các cụm nghiên cứu. Hệ thống này có khả năng truy xuất dữ liệu liên tục từ PACS của bệnh viện theo luồng DICOM thời gian thực và trích xuất metadata phục vụ phân tích hiệu suất hoạt động của các máy quét. Niffler cũng hỗ trợ truy xuất theo lô theo các bộ lọc do người dùng cung cấp, từ đó phục vụ nhiều dự án nghiên cứu quy mô lớn. (Kathiravelu et al., 2021) [2]

- **Ứng dụng và hiệu quả AI trong thực hành lâm sàng:** Lawrence và cộng sự thực hiện một systematic scoping review nhằm đánh giá việc áp dụng AI trong chẩn đoán hình ảnh. Nghiên cứu tổng hợp 140 bài báo, phân tích kết quả định lượng, chi phí, trải nghiệm người dùng và mức độ triển khai thực tế. Kết quả chỉ ra rằng AI có tiềm năng cải thiện độ chính xác chẩn đoán và rút ngắn thời gian đọc phim, nhưng vẫn tồn tại thách thức về hướng dẫn triển khai, đào tạo và minh bạch mô hình. Nghiên cứu nhấn mạnh AI nên được sử dụng như công cụ hỗ trợ chứ không thay thế hoàn toàn bác sĩ. (Lawrence et al., 2025) [3]

- **Ưu tiên worklist thông minh bằng AI:** Baltruschat và cộng sự mô phỏng workflow tại một bệnh viện đại học, áp dụng AI để xếp thứ tự ưu tiên các X-quang lồng ngực dựa trên mức độ khẩn cấp của các bệnh lý. Mô hình này giúp giảm đáng kể thời gian hoàn tất báo cáo cho các ca bệnh nguy cấp như tràn khí màng phổi. Đồng thời, nghiên cứu giới thiệu cơ chế giới hạn thời gian chờ tối đa nhằm hạn chế rủi ro từ dự đoán âm

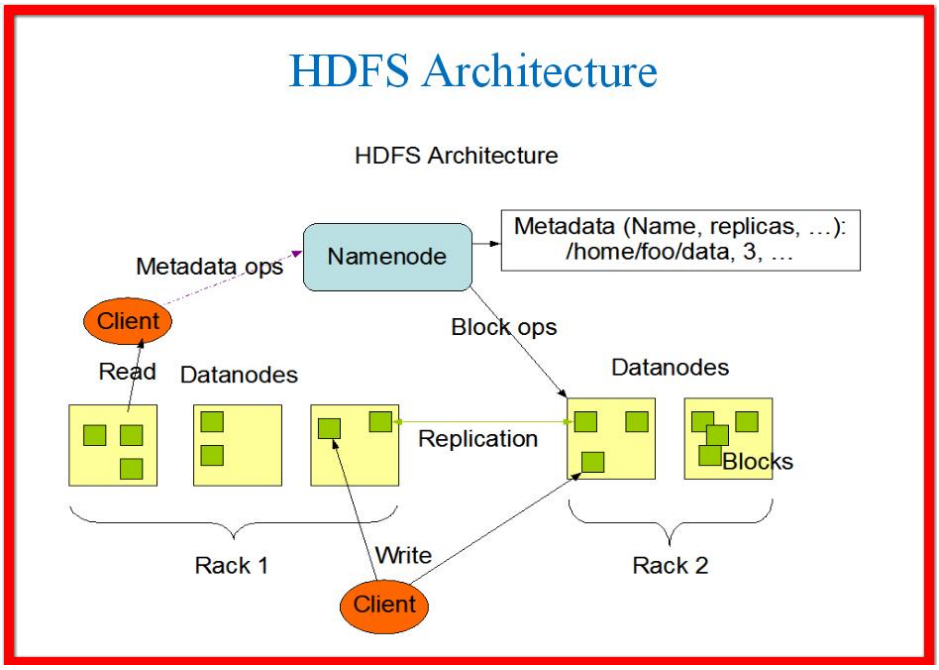
tính giả, đảm bảo hiệu quả và an toàn trong thực hành lâm sàng. (Baltruschat et al., 2021) [4]

- **Hệ thống hỗ trợ quyết định cho bệnh nhân Covid-19:** Rahim và cộng sự xây dựng một Decision Support System kết hợp AI nhằm xác định mức độ ưu tiên điều trị cho bệnh nhân Covid-19 dựa trên X-quang ngực và các đặc điểm lâm sàng. Hệ thống sử dụng mô hình VGG16 và đạt độ chính xác trên 96% trong phân loại Covid-19 và các nhóm bệnh khác. DSS này cho phép bệnh viện quyết định nhanh chóng và chính xác bệnh nhân nào cần được can thiệp trước, đồng thời đảm bảo tính công bằng trong phân bổ nguồn lực y tế. (Rahim & Kurniawan, 2020) [5]

Tóm lại là các nghiên cứu hiện nay nhấn mạnh việc kết hợp giữa mô hình học sâu, pipeline xử lý dữ liệu và tối ưu hóa workflow nhằm nâng cao chất lượng chẩn đoán và sàng lọc bệnh nhân. Hệ thống Big Data và AI trong dự án hiện tại kế thừa những hướng tiếp cận này, tập trung vào xử lý dữ liệu gần thời gian thực, phân tích nhiều loại bệnh lý phổi phổ biến và xếp thứ tự ưu tiên bệnh nhân trong quy trình chẩn đoán X-quang lồng ngực.

Chương 2: Các Công nghệ sử dụng

2.1 Hadoop Distributed File System



2.1.1 Đặc điểm của HDFS

- Lưu trữ dữ liệu phân tán: HDFS thực hiện chia nhỏ dữ liệu thành các khối (block) có kích thước lớn và phân phối chúng trên nhiều máy chủ khác nhau trong cùng một cụm Hadoop. Kích thước block mặc định là 128 MB hoặc 256 MB giúp tối ưu hóa tốc độ đọc – ghi dữ liệu, đặc biệt là khi làm việc với các tệp có dung lượng rất lớn. Nhờ cơ chế phân tán này, dữ liệu được lưu trữ đồng thời trên nhiều máy, giúp tăng khả năng xử lý song song và cải thiện hiệu năng cho các tác vụ phân tích dữ liệu.

- Khả năng chịu lỗi cao (Fault Tolerance): HDFS được thiết kế với cơ chế nhân bản dữ liệu tự động để đảm bảo an toàn khi xảy ra lỗi phần cứng. Mỗi block được tạo nhiều bản sao (replicas) và lưu trữ trên các DataNode khác nhau. Mặc định, mỗi block có 3 bản sao, giúp hệ thống vẫn hoạt động bình thường ngay cả khi một hoặc nhiều DataNode gặp sự cố. Khả năng phục hồi của HDFS giúp hạn chế tối đa tình trạng mất dữ liệu trong các hệ thống quy mô lớn.

- Thiết kế tối ưu cho dữ liệu lớn: HDFS được tối ưu hóa cho việc lưu trữ và xử lý các tệp có dung lượng lớn, thường từ hàng trăm GB đến vài PB. Mô hình "ghi một lần, đọc nhiều lần" phù hợp với các ứng dụng phân tích dữ liệu, nơi dữ liệu thô ít khi bị

chỉnh sửa mà chủ yếu được đọc liên tục bởi các hệ thống xử lý như MapReduce hoặc Spark.

- Khả năng mở rộng (Scalability): Một trong những ưu điểm nổi bật của HDFS là khả năng mở rộng gần như tuyến tính. Việc mở rộng được thực hiện rất dễ dàng chỉ bằng cách bổ sung thêm các DataNode vào cụm Hadoop mà không làm gián đoạn hệ thống. Điều này đặc biệt hữu ích trong các dự án có lượng dữ liệu tăng trưởng liên tục theo thời gian.

- Chi phí thấp: HDFS được xây dựng để chạy trên phần cứng thương mại phổ biến (commodity hardware), thay vì yêu cầu các hệ thống lưu trữ cao cấp và đắt tiền. Nhờ đó, chi phí xây dựng và vận hành hệ thống được giảm đáng kể, phù hợp với các doanh nghiệp có nhu cầu lưu trữ dữ liệu lớn nhưng muốn tiết kiệm chi phí.

2.1.2 Kiến trúc của HDFS

- NameNode: Đây là nút trung tâm của hệ thống, chịu trách nhiệm quản lý toàn bộ metadata như cấu trúc thư mục, thông tin block của từng tệp và vị trí các block được lưu trữ. NameNode không lưu dữ liệu thực tế mà chỉ lưu thông tin mô tả cách dữ liệu được phân phối trên DataNodes. Vai trò của NameNode cực kỳ quan trọng vì mọi thao tác truy cập dữ liệu đều cần phải thông qua nó để định vị block.

- DataNode: DataNode là nơi lưu trữ trực tiếp các block dữ liệu. Mỗi DataNode chịu trách nhiệm thực hiện các yêu cầu đọc và ghi dữ liệu từ phía client theo sự điều phối của NameNode. Ngoài ra, DataNode định kỳ gửi thông tin trạng thái (heartbeat) về NameNode để xác nhận rằng nó vẫn hoạt động ổn định.

- Secondary NameNode: Mặc dù tên gọi có vẻ giống node dự phòng, Secondary NameNode không thay thế trực tiếp NameNode khi xảy ra lỗi. Thay vào đó, nó đảm nhiệm việc thu thập và hợp nhất metadata nhằm giảm tải cho NameNode và hỗ trợ việc khôi phục dữ liệu nhanh hơn nếu có sự cố.

- Client: Truy xuất HDFS thông qua NameNode để đọc hoặc ghi dữ liệu. Client không tương tác trực tiếp với DataNode mà phải thông qua metadata được NameNode cung cấp, từ đó HDFS điều hướng dữ liệu hiệu quả.

2.1.3 Quy trình hoạt động

- Quy trình ghi dữ liệu: Khi người dùng tải một tệp vào hệ thống, HDFS sẽ tự động chia tệp thành các block và phân phối chúng tới các DataNode khác nhau. Trong khi đó, NameNode lưu lại metadata liên quan như số lượng block, kích thước block và vị trí lưu trữ của từng block. Nhờ cơ chế này, dữ liệu được phân tán rộng rãi và dễ dàng xử lý song song.

- Quy trình đọc dữ liệu: Khi nhận yêu cầu đọc dữ liệu, client liên hệ với NameNode để lấy danh sách vị trí các block của file. Sau đó, client sẽ trực tiếp truy cập các DataNode

chứa block tương ứng để đọc dữ liệu mà không phải thông qua NameNode, giúp giảm tải cho node này và cải thiện hiệu suất.

- Cơ chế chịu lỗi: Nếu một DataNode gặp sự cố và không gửi heartbeat về NameNode trong một khoảng thời gian quy định, NameNode sẽ xác định node này là hỏng và kích hoạt cơ chế phục hồi dữ liệu. Các bản sao mới của block trên node bị hỏng sẽ được tạo lại từ các bản sao còn lại và được phân phối sang các DataNode khác để đảm bảo đúng số lượng replicas.

2.1.4 Ứng dụng của HDFS

Trong dự án này, HDFS được sử dụng như hệ thống lưu trữ trung tâm cho toàn bộ dữ liệu ảnh X-ray và metadata liên quan. Cụm HDFS được triển khai gồm một NameNode và ba DataNode chạy trong môi trường Docker, đảm bảo khả năng phân tán, chịu lỗi và mở rộng khi dữ liệu ngày càng tăng. Việc sử dụng HDFS thay vì các cơ chế lưu trữ truyền thống mang lại nhiều lợi ích cho quy trình xử lý ảnh y khoa với khối lượng lớn.

Trước hết, HDFS đóng vai trò là nơi lưu trữ chính cho các ảnh X-ray do người dùng tải lên thông qua giao diện web. Khi backend nhận được file ảnh, hệ thống sẽ tải file đó vào HDFS thay vì lưu trực tiếp trên backend. Điều này giúp backend không bị phụ thuộc vào dung lượng của máy chủ chạy ứng dụng, đồng thời cho phép dữ liệu được lưu trữ một cách an toàn trên nhiều DataNode khác nhau. Cơ chế phân mảnh file thành các block lớn của HDFS bảo đảm rằng ngay cả những file có dung lượng lớn cũng có thể được lưu trữ hiệu quả và truy xuất nhanh chóng.

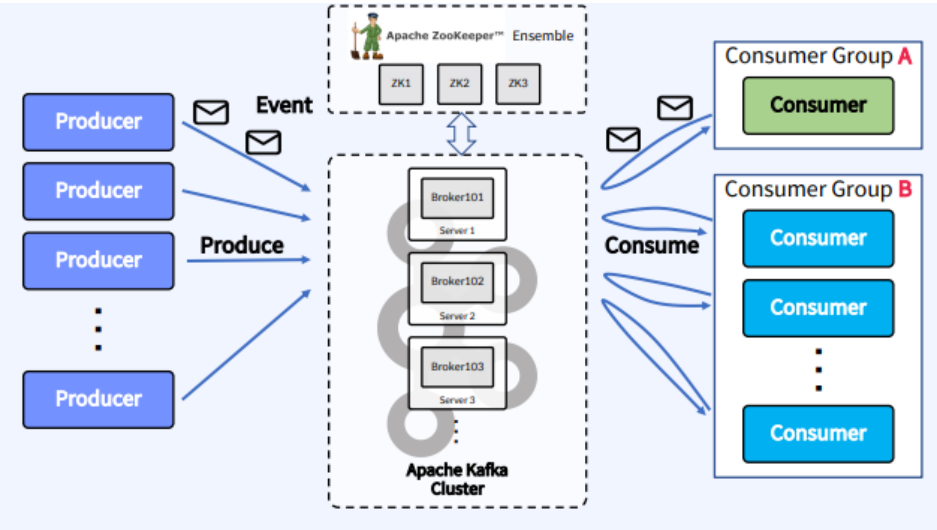
Bên cạnh ảnh, metadata của ảnh cũng được tổ chức và lưu trữ thông qua HDFS khi cần thiết. Mặc dù metadata cuối cùng được lưu trong MongoDB để truy vấn nhanh, HDFS vẫn giữ vai trò lưu trữ các tệp metadata dạng parquet để phục vụ cho các dự án lớn và phức tạp sau này.

Cuối cùng, HDFS giúp hệ thống dễ mở rộng khi số lượng ảnh X-ray tăng mạnh. Việc bổ sung thêm DataNode vào cụm hoàn toàn không ảnh hưởng đến hoạt động hiện tại của hệ thống. Điều này bảo đảm rằng dự án có thể phát triển thành hệ thống phân tích hình ảnh y khoa quy mô lớn mà không cần thay đổi hạ tầng lưu trữ cốt lõi.

2.2 Kafka

Apache Kafka là nền tảng streaming dữ liệu phân tán có khả năng chịu lỗi, độ trễ thấp và khả năng mở rộng linh hoạt. Kafka cho phép xử lý dữ liệu theo luồng thời gian thực, lưu trữ dữ liệu trong topic dưới dạng partition, mỗi partition có thể được đọc và ghi song song. Kafka hỗ trợ nhiều producer gửi dữ liệu đồng thời và nhiều consumer đọc cùng lúc thông qua consumer group, đảm bảo cân bằng tải. Một điểm đặc biệt của Kafka là khả năng tái phát dữ liệu. Nếu consumer gặp lỗi hoặc cần xử lý lại dữ liệu, nó có thể đọc lại từ offset trước đó, giúp pipeline dữ liệu luôn nhất quán.

Zookeeper là thành phần đi kèm Kafka, quản lý cluster Kafka, bao gồm việc giám sát trạng thái các broker, leader election cho các partition, và đảm bảo tính nhất quán trong cluster.



2.2.1 Kiến trúc của Kafka

Kiến trúc Kafka bao gồm nhiều thành phần liên kết chặt chẽ:

- Broker: Là server chịu trách nhiệm lưu trữ dữ liệu và phân phối cho consumer. Mỗi broker có thể quản lý một hoặc nhiều partition của topic. Các broker phối hợp với nhau tạo thành một cluster, giúp tăng khả năng chịu lỗi và mở rộng.
- Topic và Partition: Topic là đơn vị lưu trữ dữ liệu logic trong Kafka, giống như một kênh dữ liệu. Partition là phân vùng vật lý của topic, dữ liệu trong partition được lưu theo thứ tự và có offset duy nhất. Partition cho phép song song hóa việc đọc và ghi, giúp tăng throughput.
- Producer: Là thành phần gửi dữ liệu vào Kafka topic. Producer có thể chọn gửi dữ liệu đến partition cụ thể dựa trên key hoặc để Kafka phân phối tự động.
- Consumer và Consumer Group: Consumer đọc dữ liệu từ topic. Khi nhiều consumer cùng trong một consumer group, Kafka sẽ phân phối các partition cho từng consumer, giúp cân bằng tải.
- Zookeeper: Quản lý metadata của cluster Kafka. Theo dõi trạng thái broker, topic, và partition. Thực hiện leader election để xác định broker chịu trách nhiệm chính cho mỗi partition, đảm bảo cluster hoạt động ổn định ngay cả khi một broker gặp lỗi.

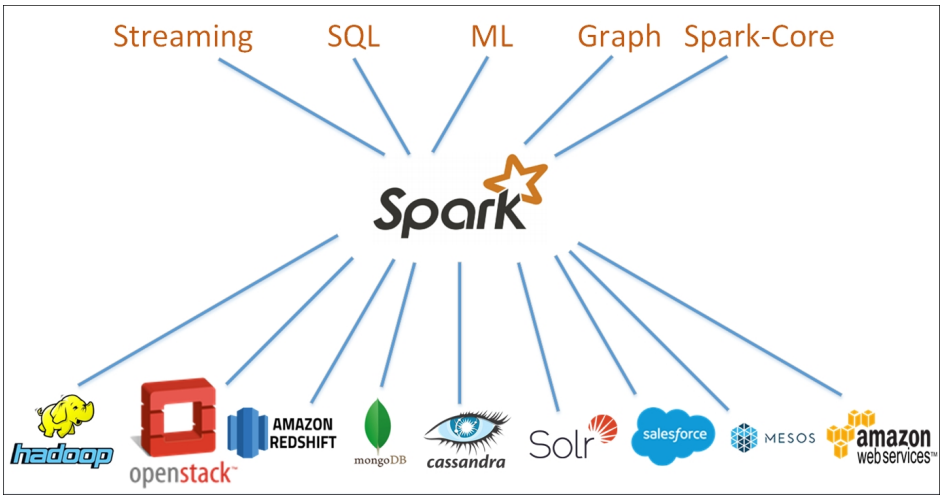
Cơ chế phối hợp: Khi producer gửi dữ liệu, broker nhận dữ liệu và ghi vào partition. Zookeeper theo dõi trạng thái cluster để đảm bảo các broker hoạt động chính xác. Consumer nhận dữ liệu dựa trên offset của partition. Nếu broker hoặc consumer gặp sự cố, Kafka tự động tái cân bằng partition, nhờ vậy pipeline dữ liệu không bị gián đoạn.

2.2.2 Ứng dụng của Kafka

Trong dự án X-ray, Kafka là trục dữ liệu trung tâm, nhận và phân phối metadata giữa các thành phần: Spark Streaming, backend và các pipeline xử lý. Zookeeper đảm bảo cluster

Kafka ổn định, partition leader được phân phối chính xác, consumer xử lý dữ liệu theo thứ tự và có thể đọc lại dữ liệu nếu cần. Nhờ Kafka và Zookeeper, hệ thống có khả năng mở rộng linh hoạt và chịu lỗi cao, phù hợp với luồng dữ liệu real-time từ hàng nghìn bản ghi metadata.

2.3 Apache Spark



Apache Spark là framework xử lý dữ liệu phân tán, hỗ trợ batch processing và streaming processing. Spark nổi bật với in-memory computing, lưu dữ liệu tạm thời trong RAM, giảm truy cập đĩa và tăng tốc xử lý. Spark có nhiều module:

- Spark Core: Xử lý RDD và DataFrame, cung cấp các phép toán phân tán như map, reduce, filter.
- Spark SQL: Truy vấn dữ liệu dạng bảng, tích hợp với HDFS, HBase, Cassandra.
- Spark Streaming: Xử lý dữ liệu real-time theo micro-batch, sử dụng API tương tự Spark Core.
- MLlib: Thư viện học máy phân tán, hỗ trợ regression, classification, clustering, giảm chiều dữ liệu.
- GraphX: Xử lý dữ liệu dạng đồ thị, tận dụng khả năng phân tán và song song hóa.

2.3.1 Kiến trúc của Apache Spark

- Driver: Driver là thành phần trung tâm của Spark, chịu trách nhiệm điều phối toàn bộ ứng dụng. Nó phân tích các job, chia nhỏ thành các task và phân phối xuống các Worker để thực thi. Driver cũng theo dõi tiến trình thực thi, xử lý lỗi nếu xảy ra và thu thập kết quả từ các Worker để tổng hợp dữ liệu cuối cùng cho người dùng.

- Cluster Manager (YARN/Mesos/Kubernetes): Cluster Manager quản lý tài nguyên trong cluster, quyết định task nào được chạy trên node nào dựa trên tài nguyên hiện có. Nó cân bằng tải giữa các Worker, giám sát tiến trình và đảm bảo các task bị lỗi có thể được chạy lại. Các hệ thống như YARN, Mesos hay Kubernetes đều có vai trò này nhưng với cách quản lý khác nhau.

- Worker Node: Worker Node là các máy chủ thực hiện các task được Driver giao. Mỗi Worker chạy một hoặc nhiều Executor để xử lý dữ liệu, lưu trữ tạm thời các partition và phối hợp với các Worker khác nhằm đảm bảo tính song song và chịu lỗi cho toàn cluster.

- Executor: Executor là tiến trình chạy trên Worker, thực hiện task, xử lý các partition của RDD/DataFrame và lưu dữ liệu tạm thời trong bộ nhớ. Executor gửi kết quả trở lại Driver và giúp tận dụng tối đa tài nguyên CPU, RAM của node để tăng hiệu suất tính toán phân tán.

- RDD/DataFrame: RDD và DataFrame là các cấu trúc dữ liệu phân tán của Spark, cho phép lưu trữ và xử lý dữ liệu trên nhiều node một cách song song. RDD cung cấp khả năng chịu lỗi thông qua lineage, trong khi DataFrame tối ưu hóa query nhờ Catalyst Optimizer, giúp Spark xử lý dữ liệu lớn hiệu quả và nhanh chóng.

Cơ chế Spark Streaming: Dữ liệu từ Kafka được chia thành micro-batch, Driver phân phối task cho Executor trên các Worker, thực hiện song song các phép biến đổi dữ liệu, kết quả được ghi lại vào MongoDB hoặc trả về Driver để backend xử lý. Spark đảm bảo khả năng chịu lỗi nhờ cơ chế lineage, checkpoint, và phân phối task lại khi một Executor gặp lỗi.

2.3.2 Ứng dụng của Apache Spark

Trong dự án, Spark Streaming được sử dụng để xử lý dữ liệu metadata từ Kafka theo thời gian thực, đảm bảo rằng các thông tin về X-ray được phân tích và dự đoán nhanh chóng ngay khi dữ liệu được gửi đến hệ thống. Nó nhận dữ liệu liên tục, thực hiện các phép biến đổi và áp dụng mô hình AI để tạo ra kết quả dự đoán, sau đó kết hợp lưu trữ song song vào MongoDB và HDFS để đảm bảo dữ liệu luôn sẵn sàng cho các bước xử lý tiếp theo. Spark Streaming đóng vai trò trung tâm trong pipeline, giúp hệ thống vận hành liên tục, giảm độ trễ, và duy trì khả năng mở rộng khi số lượng dữ liệu tăng, đồng thời đảm bảo backend và frontend luôn nhận được dữ liệu cập nhật kịp thời.

2.4 MongoDB

MongoDB là cơ sở dữ liệu NoSQL dạng document, nổi bật với khả năng lưu trữ dữ liệu theo cấu trúc JSON hoặc BSON, linh hoạt về schema, nghĩa là bạn không cần phải định nghĩa cứng nhắc các bảng hay cột như trong cơ sở dữ liệu quan hệ. Điều này đặc biệt hữu ích khi dữ liệu có cấu trúc thay đổi hoặc phức tạp, như metadata X-ray, kết quả dự đoán từ mô hình AI, và các thông tin bổ sung khác. MongoDB hỗ trợ truy vấn linh hoạt, lập chỉ mục để tăng tốc độ tìm kiếm, và cung cấp cơ chế replica set để đảm bảo dữ liệu luôn sẵn sàng và

chịu lỗi, phù hợp với các hệ thống cần hoạt động liên tục. Ngoài ra, MongoDB còn hỗ trợ sharding, cho phép mở rộng theo chiều ngang khi khối lượng dữ liệu tăng lên, đảm bảo hiệu năng truy xuất không bị giảm.

2.4.1 Kiến trúc của MongoDB

- Server: Đây là tiến trình cốt lõi chịu trách nhiệm lưu trữ dữ liệu, quản lý truy vấn, và xử lý các yêu cầu từ client. Nó dùng BSON để lưu trữ dữ liệu, cho phép MongoDB lưu các document linh hoạt mà không cần schema cứng nhắc như SQL. Server cũng quản lý journaling, đảm bảo dữ liệu không bị mất khi hệ thống gặp sự cố.

- Database, Collection, Document: Dữ liệu được tổ chức theo cấu trúc ba lớp: từ database đến collection rồi cuối cùng đến document. Mỗi document là một bản ghi kiểu JSON (BSON) có thể chứa các trường linh hoạt, kể cả mảng hoặc document lồng nhau. Cấu trúc này giúp MongoDB xử lý dữ liệu linh hoạt và mở rộng mà không cần thay đổi schema mỗi khi thêm trường mới.

- Client: Client là giao diện để người dùng hoặc ứng dụng tương tác với server. Có thể là command line, GUI hoặc API thư viện trong các ngôn ngữ lập trình. Client gửi truy vấn, nhận dữ liệu và có thể thực hiện các thao tác như insert, update, delete, hay tìm kiếm với cú pháp trực quan.

- Storage Engine: MongoDB sử dụng storage engine để quản lý cách dữ liệu được ghi vào đĩa. Ví dụ như WiredTiger là mặc định, hỗ trợ cơ chế nén dữ liệu, quản lý cache, và transaction nhỏ. Storage engine quyết định hiệu năng đọc hay ghi và cách dữ liệu phục hồi khi crash.

2.4.2 Ứng dụng MongoDB

Trong hệ thống X-ray này, MongoDB là trung tâm lưu trữ dữ liệu dự đoán và metadata, đóng vai trò kết nối giữa Spark Streaming, backend và frontend. Nó lưu trữ các kết quả dự đoán từ mô hình AI, thông tin bệnh nhân, và đường dẫn HDFS của các ảnh X-ray, giúp backend truy xuất nhanh chóng để hiển thị dữ liệu cho người dùng. Nhờ đặc tính schema linh hoạt, MongoDB cho phép mở rộng hoặc bổ sung các thông tin mới mà không cần thay đổi cấu trúc cơ sở dữ liệu cứng nhắc, đồng thời hỗ trợ truy vấn real-time để frontend luôn nhận dữ liệu cập nhật. Cơ chế replica set hoặc sharding còn đảm bảo hệ thống chịu lỗi và mở rộng dễ dàng khi lượng dữ liệu tăng lên, đặc biệt quan trọng với các pipeline dữ liệu real-time như Spark Streaming.

2.5 Backend – FastAPI

Backend trong dự án được triển khai bằng FastAPI, một framework Python nổi bật với khả năng xử lý request nhanh, hỗ trợ asynchronous và real-time. Backend chịu trách nhiệm quản lý luồng dữ liệu giữa Spark Streaming, MongoDB và frontend. Nó thực hiện các nhiệm vụ quan trọng như: nhận dữ liệu đầu vào từ client hoặc các producer khác, truy vấn dữ liệu dự

đoán từ MongoDB, xử lý các yêu cầu API, và đẩy dữ liệu real-time đến frontend thông qua WebSocket. Backend được thiết kế để hoạt động ổn định, mở rộng dễ dàng, và tương thích với môi trường containerized, đảm bảo khả năng deploy trên nhiều máy chủ hoặc cloud mà không gặp xung đột môi trường.

2.5.1 Kiến trúc của Backend

Backend được tổ chức theo mô hình API server, nơi các endpoint REST hoặc WebSocket chịu trách nhiệm nhận request và trả dữ liệu. Các thành phần chính bao gồm:

- API Router/Controller: Xử lý các request từ frontend hoặc các service khác, định tuyến đến logic xử lý phù hợp.
- Service Layer: Chứa các hàm thực hiện truy vấn MongoDB, xử lý dữ liệu và chuẩn bị response.
- Data Access Layer: Kết nối trực tiếp với MongoDB, thực hiện các thao tác đọc/ghi dữ liệu dự đoán và metadata.
- WebSocket Manager: Quản lý kết nối WebSocket giữa backend và frontend, đảm bảo dữ liệu được đẩy real-time khi có bản ghi mới từ MongoDB.

Mô hình này giúp backend tách biệt rõ ràng các trách nhiệm: xử lý request, truy xuất dữ liệu và giao tiếp với frontend, đồng thời hỗ trợ scalability và dễ bảo trì.

2.5.2 Ứng dụng của Backend

Backend FastAPI nhận file X-ray từ frontend và lưu vào HDFS để lưu trữ dữ liệu phân tán. Metadata của file được gửi tới Kafka, nơi Spark Streaming đọc để thực hiện phân tích AI trên dữ liệu trong HDFS. Kết quả phân tích sau khi Spark xử lý được ghi vào MongoDB, bao gồm dự đoán, xác suất và thông tin liên quan. WebSocket của backend đọc dữ liệu từ MongoDB và đẩy dữ liệu realtime tới frontend, giúp người dùng nhận kết quả ngay khi Spark hoàn tất xử lý mà không cần refresh. Backend cũng cung cấp API truy xuất lịch sử dự đoán và metadata hình ảnh từ MongoDB, đồng thời healthcheck đảm bảo service luôn sẵn sàng phục vụ cả HTTP request và WebSocket.

2.6 Frontend

Frontend được xây dựng bằng React, thư viện JavaScript phổ biến để tạo giao diện người dùng theo component, tối ưu cho SPA (Single Page Application). Sử dụng TypeScript giúp kiểm tra lỗi khi compile, cải thiện maintainability và độ an toàn của code so với JavaScript thuần túy. TailwindCSS là CSS framework kiểu utility-first, giúp thiết kế giao diện nhanh, chuẩn responsive và dễ tùy chỉnh mà không cần viết nhiều CSS thủ công. Nginx được sử dụng làm web server, chịu trách nhiệm phân phối tệp tĩnh, load balancing, và reverse proxy nếu cần. Kết hợp các công nghệ này giúp frontend nhanh, hiện đại, dễ bảo trì và mở rộng cho các tính năng realtime hoặc dashboard phức tạp.

2.6.1 Kiến trúc của Frontend

- Component Tree: Frontend được tổ chức theo component tree, nghĩa là toàn bộ UI được chia thành các component độc lập, mỗi component chịu trách nhiệm quản lý state và hiển thị UI riêng. Kiến trúc này giúp tách biệt các phần của giao diện, tăng khả năng tái sử dụng và dễ bảo trì. Component tree cũng giúp quản lý các tương tác phức tạp, giảm sự phụ thuộc giữa các phần của giao diện. Nhờ vậy, frontend có thể mở rộng dễ dàng khi cần thêm tính năng hoặc cập nhật giao diện.

- State Management: React sử dụng các hook như useState, useReducer cho state local và Context API để chia sẻ dữ liệu toàn cục giữa nhiều component. Cách quản lý này giúp dữ liệu từ backend hoặc WebSocket được lưu trữ và cập nhật đồng bộ giữa các component. State management hiệu quả đảm bảo UI luôn phản ánh chính xác dữ liệu hiện tại, tránh render thừa hoặc lỗi hiển thị. Nó cũng tạo nền tảng vững chắc cho các tính năng realtime và tương tác phức tạp với backend.

- Virtual DOM và Rendering: React dựa trên virtual DOM để theo dõi sự khác biệt giữa trạng thái hiện tại và trạng thái mới của UI. Chỉ những phần thay đổi mới được render lại, giúp tối ưu hiệu năng và giảm tải cho trình duyệt. Cơ chế này đặc biệt quan trọng khi frontend nhận dữ liệu realtime, tránh render toàn bộ giao diện mỗi khi dữ liệu cập nhật. Virtual DOM giúp trải nghiệm người dùng mượt mà, giảm latency và tăng tốc độ phản hồi của ứng dụng.

- Kết nối Backend (HTTP & WebSocket): Frontend tương tác với backend qua HTTP API để gửi và nhận dữ liệu. Đối với dữ liệu realtime, frontend sử dụng WebSocket để mở kênh hai chiều, nhận dữ liệu ngay khi backend có cập nhật mới. Cách này giảm độ trễ so với polling, đảm bảo người dùng nhận dữ liệu kịp thời và đồng bộ giữa các client. Kết nối trực tiếp này cũng giúp hệ thống duy trì tính realtime cho các tính năng phân tích và dashboard.

- TailwindCSS: TailwindCSS cung cấp các utility class cho layout, spacing, font, màu sắc và responsive design.

Nó giúp xây dựng giao diện nhanh chóng mà không cần viết CSS dài dòng hay tách riêng nhiều file CSS. Kiến trúc utility-first giúp duy trì consistency trong toàn bộ ứng dụng, dễ sửa đổi và mở rộng giao diện. Kết hợp TailwindCSS với React giúp UI vừa hiện đại vừa dễ bảo trì, đồng thời tối ưu hóa tốc độ phát triển.

2.6.2 Ứng dụng của Frontend

Frontend của dự án cho phép người dùng upload hình X-ray, hiển thị tiến trình upload và gửi dữ liệu trực tiếp đến backend. Ứng dụng đồng thời hiển thị kết quả phân tích realtime, bao gồm biểu đồ xác suất và metadata hình ảnh, thông qua các API và kênh WebSocket kết nối với backend. Kiến trúc component tree kết hợp với TailwindCSS giúp tách UI theo chức năng, dễ mở rộng và đảm bảo giao diện trực quan, đẹp mắt, đồng thời responsive trên mọi thiết bị. Nginx được sử dụng để phục vụ các tệp build tĩnh của React, tối ưu tốc độ tải

trang, giảm latency và nâng cao trải nghiệm người dùng khi truy cập ứng dụng. Sự kết hợp giữa React, TypeScript, TailwindCSS và Nginx đảm bảo frontend vừa hiện đại, trực quan, dễ bảo trì, vừa kết nối mượt mà với backend realtime, đáp ứng yêu cầu của một hệ thống phân tích hình ảnh X-ray thời gian thực.

2.7 Docker

Docker là nền tảng containerization cho phép đóng gói ứng dụng cùng toàn bộ môi trường chạy (dependencies, thư viện, config) vào một đơn vị chuẩn gọi là container. Các container Docker nhẹ hơn so với máy ảo truyền thống, khởi động nhanh, tách biệt hoàn toàn môi trường chạy. Docker dễ dàng triển khai trên nhiều hệ thống khác nhau mà không lo xung đột môi trường. Nó giúp giảm rủi ro chạy được trên máy chính nhưng không chạy trên server, đồng thời tăng khả năng tái sử dụng và tự động hóa build. Docker còn hỗ trợ scaling ứng dụng, giúp mở rộng hệ thống dễ dàng khi cần.

2.7.1 Kiến trúc của Docker

- Docker Engine: Đây là tiến trình trung tâm để chạy container. Nó bao gồm Docker Daemon chịu trách nhiệm build, run, stop và quản lý container. Docker CLI được dùng để tương tác với daemon thông qua các lệnh. Engine còn quản lý network, volume và resource allocation cho các container.

- Image: Là file read-only định nghĩa môi trường chạy của container, bao gồm hệ điều hành, thư viện, runtime và code ứng dụng. Image có thể version để dễ quản lý và chia sẻ. Người dùng có thể push/pull image lên Docker Hub hoặc registry riêng. Image có thể được xây dựng theo layered filesystem, giúp reuse các layer chung giữa nhiều image.

- Container: Là instance chạy của image, có filesystem, network stack và process riêng biệt. Mỗi container có thể mount volume để lưu dữ liệu persistent, đảm bảo dữ liệu không mất khi container restart. Containers có thể kết nối qua network bridge hoặc overlay, cho phép giao tiếp giữa các service. Container cũng có thể scale theo nhu cầu bằng cách chạy nhiều instance của cùng một image.

- Docker Compose: Là công cụ quản lý nhiều container cùng lúc. Compose cho phép định nghĩa network, volume, dependencies và service trong một file YAML duy nhất. Nó giúp triển khai hệ thống phức tạp với nhiều service một cách nhanh chóng. Compose cũng hỗ trợ scale, restart policy và healthcheck cho từng service, giúp hệ thống ổn định và dễ bảo trì.

2.7.2 Ứng dụng của Docker

Trong dự án X-ray, Docker được sử dụng để triển khai Hadoop HDFS, MongoDB, Spark, Kafka, FastAPI backend và React frontend. Mỗi service chạy trong container riêng, đảm bảo môi trường độc lập nhưng vẫn kết nối qua network chung. Volume Docker lưu trữ dữ

liệu hình ảnh và database, tránh mất dữ liệu khi container restart. Docker Compose giúp tự động khởi tạo toàn bộ hệ thống chỉ với một lệnh duy nhất, tiết kiệm thời gian setup. Nhờ Docker, hệ thống dễ mở rộng, bảo trì và triển khai trên nhiều máy chủ khác nhau mà không gặp xung đột môi trường.

2.8 Mô hình AI dự đoán bệnh

DenseNet (Dense Convolutional Network), là một kiến trúc mạng nơ-ron sâu được giới thiệu lần đầu vào năm 2017. Khác với các mạng CNN truyền thống, DenseNet nổi bật với cơ chế dense connectivity, theo đó mỗi layer trong một dense block nhận đầu vào là toàn bộ feature map của tất cả các layer trước đó và đồng thời truyền đầu ra của nó đến mọi layer sau. Cách kết nối này giúp mạng tận dụng tối đa thông tin từ các đặc trưng trước đó, giảm nguy cơ mất thông tin khi đi qua các lớp sâu, đồng thời tăng khả năng học các đặc trưng phức tạp từ hình ảnh, từ các chi tiết nhỏ như biên độ cạnh đến các cấu trúc lớn hơn. DenseNet cũng tích hợp các kỹ thuật chuẩn hóa và tối ưu hóa phổ biến như batch normalization, ReLU activation, convolution 3x3, giúp mạng ổn định trong quá trình huấn luyện, tăng độ chính xác, và cải thiện khả năng trích xuất đặc trưng. Nhờ vậy, DenseNet trở thành một kiến trúc mạnh mẽ và linh hoạt cho các bài toán phân loại hình ảnh đa dạng.

- **Cơ chế hoạt động của mô hình DenseNet:** DenseNet được xây dựng từ các dense block xen kẽ với transition layer, nơi mỗi dense block chịu trách nhiệm học và trích xuất đặc trưng từ ảnh, còn transition layer giảm kích thước feature map và kiểm soát số lượng channel, giúp mạng vừa sâu vừa hiệu quả về mặt tham số. Trong mỗi dense block, mỗi layer nhận đầu vào là tất cả feature map từ các layer trước đó, đồng thời xuất ra feature map riêng để nối vào các layer tiếp theo, tạo ra một luồng dữ liệu phong phú. Cơ chế này giúp gradient truyền ngược dễ dàng, giảm nguy cơ vanishing gradient, cho phép huấn luyện mạng sâu mà vẫn giữ thông tin đầy đủ từ đầu vào. DenseNet nhờ đó có khả năng học sâu hiệu quả, đồng thời tái sử dụng các đặc trưng đã học, giúp mạng không phải học lại các thông tin trùng lặp và tối ưu hóa khả năng học từ dữ liệu hạn chế.
- **Đặc điểm nổi bật của DenseNet:**
  - Dense connectivity: Mỗi layer nhận đầu vào từ tất cả các layer trước đó trong cùng dense block và truyền đầu ra đến mọi layer sau. Cơ chế này giúp mạng tận dụng tối đa các feature map đã học, giảm mất mát thông tin và cải thiện khả năng học các đặc trưng phức tạp.
  - Feature reuse: DenseNet không học lại các đặc trưng đã học trước đó mà tận dụng lại các feature map cũ, giúp mạng học hiệu quả hơn và giảm số lượng tham số cần thiết.
  - Gradient truyền ngược hiệu quả: Nhờ dense connectivity, gradient có thể đi trực tiếp từ layer cuối về các layer đầu, giảm vấn đề vanishing gradient, giúp huấn luyện mạng sâu dễ dàng và ổn định.

  - Hiệu quả tham số: DenseNet thường sử dụng ít tham số hơn so với các kiến trúc CNN truyền thống có độ sâu tương đương, bởi không cần học lại các đặc trưng đã tồn tại, vừa giảm bộ nhớ vừa tiết kiệm tính toán.
  - Tính linh hoạt trong học sâu: DenseNet có khả năng học cả các đặc trưng chi tiết lẫn tổng quát từ dữ liệu, phù hợp với các bài toán multi-label classification, nơi một ảnh có thể thuộc nhiều nhãn cùng lúc.
  - Khả năng học từ dữ liệu hạn chế: Nhờ việc tái sử dụng feature và truyền gradient tốt, DenseNet học hiệu quả ngay cả khi số lượng dữ liệu huấn luyện không quá lớn, rất thích hợp cho các lĩnh vực mà dữ liệu nhãn khó có sẵn.
  - Tối ưu hiệu suất trên ảnh đa dạng: DenseNet có khả năng nhận diện các đặc trưng ở nhiều mức độ khác nhau, từ chi tiết nhỏ đến cấu trúc lớn, giúp mạng xử lý tốt các loại hình ảnh phức tạp hoặc nhiễu.

- **Ưu điểm của DenseNet:**
  - Độ chính xác cao: DenseNet cải thiện hiệu suất phân loại hình ảnh nhờ dense connectivity và khả năng trích xuất đặc trưng mạnh mẽ, đạt kết quả tốt trên nhiều benchmark.
  - Dễ huấn luyện mạng sâu: Gradient truyền ngược tốt giúp mạng sâu hội tụ nhanh, giảm nguy cơ lỗi vanishing gradient, ổn định trong quá trình training.
  - Phù hợp cho multi-label classification: DenseNet xử lý hiệu quả các bài toán đa nhãn, ví dụ cùng một ảnh X-ray có thể chứa nhiều bệnh lý.
  - Tiết kiệm tài nguyên tính toán: Nhờ tái sử dụng feature, mạng yêu cầu ít tham số và bộ nhớ hơn so với các mạng CNN truyền thống, giảm chi phí triển khai.
  - Khả năng generalize tốt: DenseNet học được đặc trưng tổng quát, giúp mô hình hoạt động hiệu quả trên dữ liệu mới, chưa gặp trong huấn luyện.

- **Ứng dụng thực tế:** DenseNet có ứng dụng rộng rãi trong nhiều lĩnh vực khác nhau. Trong y tế, nó được sử dụng để phân tích X-quang ngực, CT, MRI, giúp nhận diện nhiều bệnh lý cùng lúc, hỗ trợ bác sĩ chẩn đoán nhanh và chính xác hơn. Trong an ninh và giám sát, DenseNet được dùng để nhận diện đối tượng, khuôn mặt, hoặc hành vi từ video và hình ảnh tĩnh, phục vụ hệ thống camera giám sát thông minh. Trong công nghiệp và sản xuất, DenseNet giúp kiểm tra chất lượng sản phẩm, phát hiện khuyết tật trên bề mặt hoặc dây chuyền sản xuất. Trong nông nghiệp và môi trường, mạng này hỗ trợ phân tích ảnh vệ tinh hoặc drone, giám sát cây trồng, phát hiện sâu bệnh, hay giám sát ô nhiễm môi trường. DenseNet còn được ứng dụng trong robotics và xe tự lái, nhận diện vật thể và môi trường xung quanh để đưa ra quyết định điều khiển chính xác. Nhờ khả năng học sâu và tái sử dụng đặc trưng hiệu quả, DenseNet phù hợp cho các hệ thống realtime và xử lý ảnh quy mô lớn.

- **Lý do sử dụng DenseNet-121 cho dự án X-ray:** Trong dự án X-ray của chúng tôi, DenseNet-121 được lựa chọn vì kiến trúc 121 layers cân bằng tốt giữa độ sâu mạng và khả năng tính toán, đồng thời pretrained trên NIH ChestX-ray14 dataset, vốn chứa nhiều loại bệnh lý phổ biến. Cấu trúc dense connectivity giúp model nhận diện các

bệnh lý phức tạp, đồng thời tái sử dụng đặc trưng hiệu quả, rất phù hợp cho nhiệm vụ multi-label classification. DenseNet-121 cũng có khả năng infer nhanh, thích hợp cho pipeline realtime của dự án, kết hợp với HDFS để lưu trữ ảnh, Kafka và Spark để xử lý dữ liệu, và WebSocket để đẩy kết quả trực tiếp tới frontend. Kiến trúc này đảm bảo backend có thể phân tích X-ray chính xác, hiệu quả và cung cấp kết quả realtime cho người dùng.

Chương 3: Triển Khai Hệ Thống

3.1 Tổng quan hệ thống

Hệ thống được xây dựng nhằm hỗ trợ việc phân tích và đánh giá hình ảnh X-ray theo hướng tự động hóa, đồng thời xác định mức độ nghiêm trọng của bệnh lý và ưu tiên xử lý bệnh nhân một cách hợp lý. Mục tiêu chính của hệ thống là tạo ra một nền tảng phân tích ảnh X-ray vận hành hoàn toàn tự động, cho phép tiếp nhận dữ liệu từ nguồn bên ngoài, tiến hành xử lý và cung cấp các kết quả phân tích, đồng thời xếp thứ tự ưu tiên dựa trên mức độ nguy hiểm của bệnh lý.

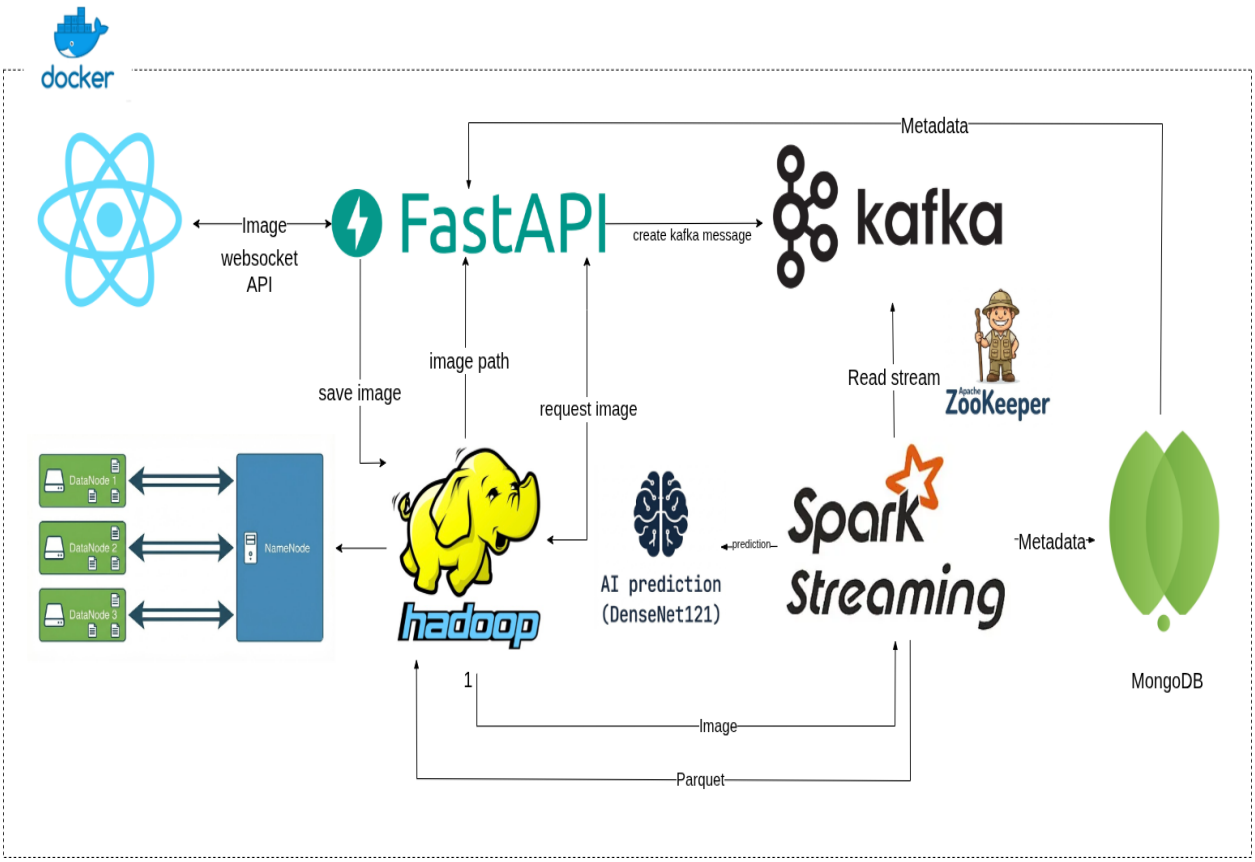
Kiến trúc tổng thể của hệ thống được thiết kế theo mô hình phân tán, tận dụng các nền tảng như hệ thống lưu trữ phân tán, công cụ xử lý dữ liệu streaming và cơ sở dữ liệu hiệu năng cao để đảm bảo khả năng vận hành trơn tru. Trong cấu trúc này, mỗi thành phần chịu trách nhiệm cho một nhiệm vụ riêng, từ lưu trữ dữ liệu, xử lý và phân tích bằng mô hình AI, đến quản lý và lưu trữ kết quả phục vụ truy vấn, báo cáo và sắp xếp thứ tự ưu tiên bệnh nhân. Sự kết hợp giữa các thành phần này giúp hệ thống duy trì tính linh hoạt, khả năng chịu lỗi và hiệu suất xử lý liên tục theo thời gian thực.

Với cách tổ chức như vậy, hệ thống không chỉ đáp ứng các yêu cầu về phân tích và xử lý tự động mà còn đảm bảo khả năng lưu trữ dài hạn, tổng hợp dữ liệu và xác định thứ tự ưu tiên bệnh một cách khoa học. Nhờ đó, toàn bộ quy trình phân tích và đánh giá hình ảnh X-ray được chuẩn hóa, tối ưu hóa và tạo nền tảng vững chắc cho các chức năng nâng cao hoặc mở rộng quy mô trong tương lai.

3.2 Kiến trúc của hệ thống

Hệ thống X-Ray Prediction được triển khai theo kiến trúc phân tán, kết hợp nhiều công cụ và thành phần hiện đại để đảm bảo vận hành tự động, ổn định và có khả năng mở rộng. Các thành phần chính bao gồm:

- HDFS: Lưu trữ hình ảnh X-ray và kết quả phân tích, đảm bảo dung lượng lớn, khả năng mở rộng và độ bền cao.
- Apache Kafka: Quản lý luồng dữ liệu và sự kiện giữa các thành phần, giúp hệ thống xử lý dữ liệu liên tục và đồng bộ.
- Apache Spark Streaming: Thực hiện xử lý dữ liệu liên tục và phân tích gần thời gian thực.
- TorchXRayVision với mô hình DenseNet121: Phân tích và dự đoán bệnh lý trên hình ảnh X-ray, cung cấp xác suất và đánh giá mức độ nghiêm trọng.
- MongoDB: Lưu trữ kết quả phân tích, hỗ trợ truy vấn nhanh và tổng hợp dữ liệu cho giao diện và báo cáo.



- FastAPI: Cung cấp API để giao tiếp với giao diện web, truy xuất dữ liệu và quản lý kết quả.
- React + TailwindCSS: Giao diện người dùng trực quan, hiển thị thống kê, danh sách ưu tiên và chi tiết kết quả phân tích.
- Docker và Docker Compose: Quản lý môi trường triển khai, đảm bảo các thành phần hoạt động độc lập, dễ triển khai và bảo trì.

Sự kết hợp giữa các công cụ này giúp hệ thống vận hành tự động, xử lý dữ liệu liên tục, đảm bảo hiệu suất cao, chịu lỗi tốt và dễ mở rộng khi khối lượng dữ liệu tăng. Đồng thời, dữ liệu và kết quả được quản lý có hệ thống, thuận tiện cho các phân tích nâng cao, thống kê xu hướng và huấn luyện mô hình trong tương lai.

3.3 Pipeline xử lý

Pipeline xử lý trong hệ thống X-Ray Prediction được thiết kế để vận hành hoàn toàn tự động, kết hợp khả năng xử lý ảnh theo lô và metadata theo thời gian thực, nhằm đảm bảo hiệu suất, tính ổn định và khả năng mở rộng. Các bước xử lý của hệ thống bao gồm:

- **Bước 1: Tiếp nhận ảnh từ người dùng và hệ thống tự động:**

- Bác sĩ upload từng ảnh một: Bác sĩ upload ảnh X-ray trực tiếp thông qua frontend. Ảnh được gửi qua HTTP API, kèm theo metadata như ID bệnh nhân, loại ảnh, thời gian chụp... Ngay sau khi ảnh được nhận, backend sẽ lưu ảnh tạm thời và tạo metadata để đưa vào luồng xử lý tiếp theo. Ảnh này thường được ưu tiên xử lý ngay bằng AI để phát hiện bệnh lý sớm.
- Gửi ảnh theo lô từ folder hệ thống: Song song với luồng người dùng, hệ thống còn có một cơ chế tự động quét folder nội bộ. Cứ mỗi 5 giây, hệ thống gom một batch gồm khoảng 5 ảnh và gửi trực tiếp vào HDFS để lưu trữ. Sau khi ảnh được lưu vào HDFS, metadata của từng ảnh trong batch sẽ được tạo và đẩy vào Kafka để tiếp tục xử lý. Luồng này phù hợp với các tình huống tầm soát diện rộng, hoặc xử lý ảnh từ các khoa không yêu cầu phản hồi tức thì.

- **Bước 2: Lưu ảnh theo lô vào HDFS:**

  Ngay sau khi nhận ảnh, backend sẽ gom ảnh lại theo lô rồi lưu vào hệ thống lưu trữ phân tán Hadoop HDFS. Việc lưu theo lô giúp tối ưu hiệu suất ghi đĩa, giảm số lượng kết nối I/O, và phù hợp với kiến trúc Big Data. Ảnh được lưu theo cấu trúc thư mục logic, giúp dễ dàng truy xuất và phân tích sau này. Đây là bước quan trọng để đảm bảo dữ liệu ảnh được bảo toàn và sẵn sàng cho cả xử lý real-time lẫn batch.

- **Bước 3: Gửi metadata vào Kafka:**

  Sau khi ảnh được lưu vào HDFS, backend tạo một message Kafka chứa metadata: đường dẫn ảnh trong HDFS, ID bệnh nhân và các thông tin bổ sung. Kafka đóng vai trò là hàng đợi sự kiện, giúp hệ thống xử lý ảnh theo luồng mà không bị nghẽn. Mỗi ảnh là một sự kiện riêng biệt, dễ theo dõi và mở rộng. Việc chỉ gửi metadata thay vì ảnh sẽ giúp giảm tải cho hệ thống truyền dữ liệu và tăng tốc độ xử lý do ảnh rất nặng sẽ ảnh hưởng đến tốc độ truyền dữ liệu.

- **Bước 4: Spark Streaming đọc Kafka:**

  Spark Streaming liên tục lắng nghe các topic Kafka để nhận metadata ảnh mới. Khi nhận được message, Spark sẽ trích xuất thông tin cần thiết và chuẩn bị cho bước xử lý tiếp theo. Spark có áp dụng bộ lọc để ưu tiên ảnh từ luồng upload của bác sĩ hơn là luồng từ folder hệ thống. Spark hoạt động theo micro-batch, xử lý ảnh theo nhóm nhỏ để cân bằng giữa hiệu suất và độ trễ.

- **Bước 5: Spark đọc ảnh từ HDFS và đưa vào AI Model:**

  Dựa trên đường dẫn ảnh từ Kafka, Spark Streaming truy xuất ảnh gốc từ HDFS và chuyển dữ liệu vào mô hình DenseNet121. Mô hình AI chịu trách nhiệm phân tích và dự đoán các bệnh lý như viêm phổi, tràn dịch màng phổi, lao phổi, các khối u phổi,.... Cách tổ chức này đảm bảo Spark quản lý luồng dữ liệu hiệu quả, tối ưu hóa throughput, trong khi AI model thực sự thực hiện việc đọc và phân tích nội dung ảnh, đồng thời vẫn giữ khả năng xử lý gần như thời gian thực.

- **Bước 6: Lưu kết quả vào MongoDB và HDFS:**

Kết quả dự đoán từ mô hình AI sẽ kết hợp với metadata và được Spark lưu vào hai nơi:

- MongoDB: lưu metadata và kết quả để phục vụ truy vấn nhanh từ backend hoặc dashboard.
- HDFS: lưu metadata và kết quả theo định dạng Parquet để phục vụ phân tích batch, huấn luyện lại mô hình, hoặc tạo báo cáo.

Việc lưu song song giúp hệ thống vừa phản hồi nhanh, vừa đảm bảo dữ liệu đầy đủ cho phân tích sâu.

- **Bước 7: Frontend truy vấn Backend:**

  Sau khi Spark và AI model xử lý xong, kết quả được lưu vào MongoDB và HDFS. Khi bác sĩ mở frontend, ứng dụng sẽ truy vấn backend để tự động lấy dữ liệu mới nhất. Backend sẽ đọc metadata từ MongoDB và đọc HDFS nếu muốn xem chi tiết ảnh.

  Frontend không chỉ hiển thị kết quả dự đoán bệnh lý mà còn sắp xếp và gắn nhãn mức độ ưu tiên cho từng ca bệnh. Cơ chế hiển thị này giúp bác sĩ nhanh chóng nhận diện ca bệnh quan trọng, giảm thời gian chờ và tăng hiệu quả điều trị. Đồng thời, frontend có thể cung cấp bộ lọc để hỗ trợ quản lý ca bệnh.

3.4 Công thức tính toán ưu tiên cho bệnh nhân

Một trong những chức năng cốt lõi của hệ thống là khả năng tự động sắp xếp lại hàng đợi bệnh nhân thay vì sử dụng cơ chế truyền thống (đến trước phục vụ trước). Để thực hiện điều này, hệ thống sử dụng một thuật toán tính điểm ưu tiên tổng hợp (P), cân bằng giữa mức độ nghiêm trọng của bệnh lý (S) và thời gian chờ đợi (T).

Công thức tính toán được định nghĩa như sau:

$$P = (S \times w_s) + (T \times w_t) \tag{3.1}$$

Trong đó, công thức cụ thể được áp dụng trong hệ thống là:

$$\text{Priority Score} = (\text{Severity Level} \times 10) + (\text{Hours Waiting} \times 0.5) \tag{3.2}$$

3.4.1 Giải thích các tham số

- **Priority Score (P):** Điểm số cuối cùng dùng để xếp hạng. Giá trị càng cao, bệnh nhân càng được ưu tiên hiển thị lên đầu danh sách trên giao diện của bác sĩ.
- **Severity Level (S):** Mức độ nghiêm trọng của bệnh lý, được xác định bởi hệ thống AI dựa trên xác suất dự đoán cao nhất.
  - Giá trị S nằm trong khoảng số nguyên **[0, 4]**.
  - Hệ thống phân chia các mức độ như sau:

    - **Mức 0 (Bình thường):** Không phát hiện tổn thương hoặc xác suất bệnh lý rất thấp.
    - **Mức 1 (Nhẹ):** Các dấu hiệu nghi ngờ nhỏ, chưa rõ ràng.
    - **Mức 2 (Trung bình):** Các bệnh lý mãn tính hoặc tổn thương khu trú (ví dụ: Nốt phổi nhỏ, xơ hóa).
    - **Mức 3 (Nặng):** Các tổn thương lan rộng hoặc bệnh lý cần chú ý (ví dụ: Viêm phổi, thâm nhiễm lớn).
    - **Mức 4 (Nguy kịch):** Các tình trạng cấp cứu hô hấp (ví dụ: Tràn khí màng phổi - Pneumothorax, Phù phổi cấp - Edema).

- **Hours Waiting (T):** Thời gian chờ của bệnh nhân tính bằng giờ, được tính từ thời điểm ảnh X-quang được đưa vào hệ thống (T_{upload}) đến thời điểm hiện tại (T_{now}).

$$T = T_{now} - T_{upload}$$

  Tham số này đảm bảo sự công bằng, giúp những bệnh nhân có bệnh nhẹ nhưng đã chờ đợi quá lâu sẽ dần dần tăng thứ hạng, tránh tình trạng bị bỏ quên (starvation).

3.4.2 Phân tích trọng số (Weights)

Việc lựa chọn trọng số $w_s = 10$ và $w_t = 0.5$ mang ý nghĩa quan trọng trong y tế:

- **Trọng số nghiêm trọng ($w_s = 10$):** Với thang đo [0-4], khoảng cách điểm số giữa hai mức độ bệnh liền kề là 10 điểm. Đây là một khoảng cách lớn, nhằm khẳng định rằng tình trạng bệnh lý là yếu tố quyết định hàng đầu.

- **Trọng số thời gian ($w_t = 0.5$):** Hệ số 0.5 cho mỗi giờ chờ đợi là con số nhỏ, đóng vai trò phụ trợ.

- **Tương quan đánh đổi:** Để một bệnh nhân ở mức độ nhẹ hơn (ví dụ Mức 3) có thể vượt qua một bệnh nhân ở mức độ nặng hơn (Mức 4), họ phải chờ đợi khoảng thời gian là:

$$T = \frac{\Delta S \times 10}{0.5} = \frac{1 \times 10}{0.5} = 20 \text{ giờ}$$

  Điều này đảm bảo an toàn: một ca cấp cứu (Mức 4) dù mới đến cũng sẽ được ưu tiên xử lý trước ca bệnh nặng (Mức 3) trừ khi ca Mức 3 đã phải chờ đợi gần 1 ngày. Cơ chế này giúp bác sĩ luôn tiếp cận được ca nguy hiểm nhất trong thời gian ngắn nhất.

3.4.3 Ví dụ minh họa

Giả sử tại cùng một thời điểm, hệ thống có 3 bệnh nhân với các chỉ số như sau:

Bệnh nhân	Chẩn đoán AI	Mức độ (S)	Thời gian chờ (T)	Tính toán ($10S + 0.5T$)
BN A	Tràn khí màng phổi	**4** (Nguy kịch)	0.1 giờ (6 phút)	$40 + 0.05$
BN B	Viêm phổi	**2** (Trung bình)	4 giờ	$20 + 2$
BN C	Bình thường	**0** (Bình thường)	10 giờ	$0 + 5$

Bảng 3.1: Ví dụ xếp hạng ưu tiên bệnh nhân

Kết quả xếp hạng:

1. **BN A (40.05 điểm):** Dù mới đến nhưng bệnh lý nguy kịch (Mức 4) tạo ra điểm số nền rất cao, ngay lập tức được đẩy lên đầu danh sách.
2. **BN B (22.00 điểm):** Bệnh lý mức trung bình cộng với thời gian chờ tạo ra mức độ ưu tiên vừa phải.
3. **BN C (5.00 điểm):** Dù đã chờ rất lâu, nhưng do không có bệnh lý nên điểm ưu tiên vẫn thấp nhất, nhường quyền khám cho các ca có bệnh thực thể.

Như vậy, công thức này hoàn thành tốt mục tiêu kép: ưu tiên tuyệt đối cho cấp cứu và đảm bảo không bỏ sót bệnh nhân trong hàng đợi.

3.5 Khả năng chịu lỗi

Trong một hệ thống xử lý ảnh X-ray quy mô lớn, khả năng chịu lỗi là yếu tố sống còn để đảm bảo dữ liệu không bị mất và pipeline luôn hoạt động ổn định. Khi ảnh được lưu vào HDFS, cơ chế replication sẽ tự động nhân bản dữ liệu sang nhiều DataNode. Nếu một node gặp sự cố, NameNode sẽ điều phối để đọc từ bản sao khác, nhờ đó ảnh gốc vẫn luôn sẵn sàng cho các bước xử lý tiếp theo. Điều này giúp hệ thống không bị gián đoạn ngay cả khi phần cứng hỏng hóc.

Ở tầng truyền dữ liệu, Kafka và Spark Streaming phối hợp để duy trì tính liên tục. Kafka lưu trữ message trong log phân tán và có cơ chế leader–follower cho partition, nên nếu một broker chết thì broker khác sẽ tiếp quản. Spark Streaming sử dụng checkpoint để ghi lại trạng thái đọc offset. Khi job Spark bị crash hoặc restart, nó sẽ tiếp tục từ checkpoint thay vì mất dữ liệu, đảm bảo mọi metadata ảnh đều được xử lý.

Trong quá trình inference, hàm AI có cơ chế bắt lỗi để tránh pipeline dừng lại khi một ảnh không đọc được hoặc model gặp sự cố. Thay vì làm gián đoạn toàn bộ, hệ thống sẽ trả về một record "Error" cho ca đó, còn các ca khác vẫn tiếp tục chạy. Đây là cách hệ thống giảm thiểu rủi ro và duy trì dịch vụ liên tục. Cuối cùng, khi ghi kết quả, MongoDB có replica set để đảm bảo dữ liệu vẫn khả dụng, còn HDFS lưu kết quả dưới dạng Parquet với tính chất append-only, giúp giảm nguy cơ ghi đè và mất mát. Nhờ những cơ chế này, pipeline có thể tự phục hồi và duy trì ổn định ngay cả khi một thành phần gặp sự cố.

Chương 4: Kết quả, hướng phát triển và kết luận

4.1 Kết quả đạt được

Sau quá trình thiết kế và triển khai, nhóm nghiên cứu đã xây dựng thành công hệ thống Big Data xử lý ảnh X-quang ngực với cơ chế ưu tiên bệnh lý. Hệ thống đã vận hành ổn định trên môi trường giả lập và đạt được các kết quả cụ thể như sau:

4.1.1 Vận hành hệ thống và Pipeline dữ liệu

Hệ thống đã thiết lập thành công một pipeline xử lý dữ liệu khép kín và tự động hóa cao:

- **Lưu trữ phân tán hiệu quả:** Hệ thống HDFS đã được triển khai trên nền tảng Docker, cho phép lưu trữ an toàn khối lượng lớn ảnh X-quang và kết quả phân tích dưới dạng Parquet. Cơ chế phân tán giúp đảm bảo tính sẵn sàng của dữ liệu ngay cả khi giả lập sự cố tắt một DataNode.
- **Xử lý thời gian thực:** Sự kết hợp giữa Apache Kafka và Spark Streaming cho phép truyền tải và xử lý metadata với độ trễ thấp. Thời gian từ lúc ảnh được đưa vào hệ thống đến khi có kết quả hiển thị trên giao diện người dùng trung bình chỉ mất từ 1 đến 2 giây (tùy thuộc vào hạ tầng phần cứng thử nghiệm), đáp ứng tốt yêu cầu về cảnh báo sớm trong y tế.
- **Tích hợp Mô hình AI:** Mô hình DenseNet121 đã được tích hợp thành công vào luồng Spark Streaming. Mô hình hoạt động ổn định, thực hiện dự đoán song song trên các executor của Spark mà không gây tắc nghẽn luồng dữ liệu.

4.1.2 Hiệu quả của cơ chế xếp hàng ưu tiên

Hệ thống đã có khả năng thay đổi quy trình làm việc truyền thống người đến trước ưu tiên trước sang quy trình dựa trên mức độ nghiêm trọng:

- **Phân loại chính xác:** Hệ thống tự động nhận diện và phân loại 14 bệnh lý phổi. Các ca bệnh nguy hiểm như *Pneumothorax* (Tràn khí màng phổi) hay *Edema* (Phù phổi) được gán trọng số cao.
- **Sắp xếp danh sách chờ:** Trên giao diện Frontend, danh sách bệnh nhân không còn hiển thị theo thứ tự thời gian đơn thuần. Những bệnh nhân có dấu hiệu bệnh lý nguy kịch được tự động đẩy lên đầu danh sách và được đánh dấu đỏ, giúp bác sĩ nhận biết ngay lập tức.
- **Giảm thiểu rủi ro:** Trong các kịch bản thử nghiệm với luồng dữ liệu hỗn hợp gồm cả ca bệnh nặng và nhẹ, hệ thống đảm bảo các ca bệnh nặng được đưa vào danh sách ưu tiên xử lý trước, giảm thiểu nguy cơ bỏ sót trong "thời gian vàng".

4.1.3 Giao diện người dùng và Trải nghiệm

- **Dashboard trực quan:** Giao diện hiển thị rõ ràng thông tin bệnh nhân, ảnh X-quang gốc và biểu đồ xác suất các bệnh lý.
- **Cập nhật tức thời:** Nhờ ứng dụng WebSocket, kết quả phân tích từ Spark -> MongoDB được đẩy trực tiếp lên màn hình bác sĩ mà không cần tải lại trang, tạo trải nghiệm mượt mà và chuyên nghiệp.

4.2 Hạn chế và Hướng phát triển

Mặc dù hệ thống đã hoạt động đúng theo thiết kế và đạt được các mục tiêu ban đầu, dự án vẫn còn một số hạn chế cần khắc phục để có thể triển khai trong môi trường bệnh viện thực tế. Dưới đây là các định hướng phát triển trong tương lai:

4.2.1 Cải thiện Mô hình AI

- **Độ chính xác:** Hiện tại mô hình DenseNet121 sử dụng pretrained weights từ bộ dữ liệu công khai. Trong tương lai, cần fine-tune mô hình với dữ liệu đặc thù của địa phương hoặc bệnh viện cụ thể để nâng cao độ chính xác.
- **Bổ sung Heatmap/Grad-CAM:** Tích hợp kỹ thuật trực quan hóa vùng tổn thương (Grad-CAM) đè lên ảnh gốc, giúp bác sĩ không chỉ biết bệnh nhân mắc bệnh gì mà còn biết vị trí tổn thương nằm ở đâu trên phổi, tăng tính thuyết phục của chẩn đoán.

4.2.2 Xây dựng mô hình AI ưu tiên

- Hiện tại hệ thống đang sử dụng công thức cố định cho việc ưu tiên bệnh nhân nên vẫn còn nhiều hạn chế. Cần xây dựng một mô hình AI dựa trên metadata của hệ thống.

4.2.3 Tối ưu hóa và Mở rộng Hạ tầng

- **Triển khai Kubernetes (K8s):** Hiện tại hệ thống chạy trên Docker Compose, phù hợp cho thử nghiệm. Để triển khai thực tế với khả năng mở rộng linh hoạt khi lượng bệnh nhân tăng đột biến, cần chuyển đổi sang nền tảng Kubernetes để quản lý cluster hiệu quả hơn.
- **Tích hợp chuẩn DICOM/PACS:** Hệ thống hiện tại xử lý ảnh dạng file thông thường. Để tương thích với hệ thống y tế hiện đại, cần phát triển module giao tiếp trực tiếp với hệ thống PACS qua giao thức DICOM.

4.2.4 Bảo mật và Quy trình nghiệp vụ

- **Bảo mật dữ liệu y tế:** Bổ sung các cơ chế mã hóa dữ liệu trên đường truyền và dữ liệu lưu trữ, đảm bảo tuân thủ các tiêu chuẩn như HIPAA hay GDPR về bảo vệ thông tin bệnh nhân.
- **Cơ chế phản hồi :** Xây dựng tính năng cho phép bác sĩ xác nhận hoặc sửa lại kết quả dự đoán của AI. Dữ liệu phản hồi này sẽ được lưu lại để huấn luyện lại mô hình định kỳ, giúp hệ thống ngày càng thông minh hơn theo thời gian.

4.3 Kết luận

Dự án này đã giải quyết thành công bài toán quá tải trong quy trình chẩn đoán hình ảnh tại các cơ sở y tế. Bằng việc kết hợp sức mạnh của công nghệ dữ liệu lớn (Hadoop, Spark, Kafka) và trí tuệ nhân tạo, nhóm nghiên cứu đã xây dựng được một giải pháp toàn diện từ khâu thu thập, lưu trữ, xử lý đến hiển thị kết quả.

Hệ thống không chỉ dừng lại ở việc hỗ trợ chẩn đoán mà còn mang lại giá trị nhân văn sâu sắc thông qua cơ chế ưu tiên bệnh nhân. Việc tự động sàng lọc và đưa các ca bệnh nguy kịch lên đầu danh sách chờ giúp tối ưu hóa nguồn lực y tế, đảm bảo tính công bằng dựa trên tình trạng bệnh và quan trọng nhất là hỗ trợ các bác sĩ tận dụng tối đa "thời gian vàng" để cứu chữa người bệnh.

Kết quả của đồ án là minh chứng rõ nét cho xu hướng chuyển đổi số trong y tế, khẳng định tiềm năng to lớn của việc ứng dụng Big Data và AI vào việc nâng cao chất lượng khám chữa bệnh và chăm sóc sức khỏe cộng đồng. Đây là nền tảng vững chắc để tiếp tục phát triển các hệ thống y tế thông minh hơn, hiệu quả hơn trong tương lai.

Tài liệu tham khảo

[1] Eui Jin Hwang et al. "Deep learning for chest radiograph diagnosis in the emergency department". In: *Radiology* 293.3 (2019), pp. 573–580.

[2] Pradeeban Kathiravelu et al. "A DICOM framework for machine learning and processing pipelines against real-time radiology images". In: *Journal of Digital Imaging* 34.4 (2021), pp. 1005–1013.

[3] Rachel Lawrence et al. "Artificial intelligence for diagnostics in radiology practice: a rapid systematic scoping review". In: *EClinicalMedicine* 83 (2025).

[4] Ivo Baltruschat et al. "Smart chest X-ray worklist prioritization using artificial intelligence: a clinical workflow simulation". In: *European radiology* 31.6 (2021), pp. 3837–3845.

[5] Arham Rahim, Muhamad Kurniawan, et al. "Machine learning based decision support system for determining the priority of covid–19 patients". In: *2020 3rd International Conference on Information and Communications Technology (ICOIACT)*. IEEE. 2020, pp. 319–324.

Phân công công việc

STT	Thành viên	Nhiệm vụ chi tiết
1	**Nguyễn Văn Huy**	**Xây dựng Pipeline xử lý dữ liệu:** - Thiết lập hạ tầng Docker và Docker Compose cho toàn bộ hệ thống (HDFS, Kafka, Zookeeper, Spark). - Cấu hình cụm Kafka và Zookeeper để quản lý luồng metadata. - Lập trình module Spark Streaming: Nhận metadata từ Kafka, tương tác với HDFS để đọc ảnh và gọi module AI xử lý.
2	**Hoàng Văn Dương**	**- Phát triển module AI:** - Nghiên cứu và lựa chọn kiến trúc mạng DenseNet121 cho bài toán phân loại đa nhãn; - Xây dựng thuật toán tính điểm ưu tiên dựa trên xác suất bệnh lý. - **Phát triển Backend:** Xây dựng API tiếp nhận ảnh từ người dùng, đẩy dữ liệu vào HDFS/Kafka và quản lý kết nối WebSocket để gửi kết quả realtime xuống Frontend. - **Kết nối toàn bộ hệ thống:** Thực hiện kết nối, sửa lỗi, toàn bộ hệ thống, đảm bảo hệ thống chạy ổn định, mượt mà.
3	**Nguyễn Gia Huy**	**Phát triển Giao diện người dùng:** - Thiết kế giao diện Dashboard sử dụng ReactJS và TailwindCSS. - Xây dựng chức năng Upload ảnh và hiển thị trạng thái xử lý. - Lập trình hiển thị danh sách bệnh nhân theo hàng đợi ưu tiên dựa trên kết quả trả về. - Tích hợp WebSocket để nhận dữ liệu realtime từ Backend và vẽ biểu đồ phân tích bệnh lý.

Bảng 2: Bảng phân công công việc chi tiết của nhóm