BỘ GIÁO DỤC
VÀ ĐÀO TẠO

VIỆN HÀN LÂM KHOA HỌC
VÀ CÔNG NGHỆ VIỆT NAM

HỌC VIỆN KHOA HỌC VÀ CÔNG NGHỆ

Phạm Tuấn Thành

TÍCH HỢP CÁC NỀN TẢNG MÃ NGUỒN MỞ PHỤC VỤ PHÁT TRIỂN MÔ HÌNH HỌC MÁY (MLOPS) VÀ ỨNG DỤNG

LUẬN VĂN THẠC SĨ MÁY TÍNH

Hà Nội – 2026

BỘ GIÁO DỤC
VÀ ĐÀO TẠO

VIỆN HÀN LÂM KHOA HỌC
VÀ CÔNG NGHỆ VIỆT NAM

HỌC VIỆN KHOA HỌC VÀ CÔNG NGHỆ

Phạm Tuấn Thành

TÍCH HỢP CÁC NỀN TẢNG MÃ NGUỒN MỞ PHỤC VỤ PHÁT TRIỂN MÔ HÌNH HỌC MÁY (MLOPS) VÀ ỨNG DỤNG

LUẬN VĂN THẠC SĨ MÁY TÍNH

Ngành: ***Hệ thống thông tin***

Mã số: 8 48 01 04

NGƯỜI HƯỚNG DẪN KHOA HỌC:

PGS. TS. Nguyễn Long Giang

Hà Nội - 2026

LỜI CAM ĐOAN

Tôi xin cam đoan nội dung trong đề tài nghiên cứu của luận văn "Tích hợp các nền tảng mã nguồn mở phục vụ phát triển mô hình học máy (MLOPS) và ứng dụng" là kết quả do tôi đã tìm hiểu, nghiên cứu và thực hiện trong quá trình học tập dưới sự hướng dẫn của PGS.TS. Nguyễn Long Giang. Các nội dung trong luận văn này được trình bày dựa trên quá trình nghiên cứu tài liệu, áp dụng phương pháp và thực nghiệm do tôi thực hiện. Những kết quả trình bày trong luận văn mang tính trung thực, phản ánh đúng trong quá trình nghiên cứu và không sao chép từ bất kỳ công trình khác. Tất cả các tài liệu tham khảo đều được trích dẫn đầy đủ theo quy định. Tôi xin hoàn toàn chịu trách nhiệm về tính chính xác của nội dung của luận văn.

Hà Nội, ngày 9 tháng 7 năm 2026

Tác giả luận văn

Phạm Tuấn Thành

LỜI CẢM ƠN

Lời đầu tiên, tôi xin bày tỏ lòng biết ơn sâu sắc đến Ban Giám đốc Học viện Khoa học và Công nghệ, Viện Hàn lâm Khoa học và Công nghệ Việt Nam cùng các quý Thầy/Cô Học viện đã luôn tận tình truyền đạt kiến thức và tạo mọi điều kiện thuận lợi cho tôi trong suốt quá trình học tập.

Đặc biệt, tôi xin gửi lời cảm ơn chân thành nhất đến PGS.TS. Nguyễn Long Giang, người đã trực tiếp hướng dẫn, dành nhiều thời gian và tâm huyết để chỉ bảo, giúp đỡ tôi vượt qua những khó khăn trong suốt thời gian thực hiện đề tài luận văn này. Nếu không có sự định hướng và những lời khuyên quý báu của Thầy, tôi khó có thể hoàn thành bài luận một cách trọn vẹn.

Bên cạnh đó, tôi cũng xin gửi lời cảm ơn tới bạn bè, đồng nghiệp trong tập thể lớp Hệ thống thông tin đã luôn hỗ trợ và góp ý trong suốt quá trình học tập để tôi hoàn thiện nghiên cứu của mình.

Cuối cùng, tôi xin gửi lời tri ân đến gia đình và bạn bè – những người đã luôn là điểm tựa tinh thần, cổ vũ và động viên tôi trong những lúc khó khăn nhất.

Dù đã dành nhiều tâm huyết và nỗ lực, nhưng do kiến thức và kinh nghiệm còn hạn chế, bài luận văn chắc chắn không tránh khỏi những thiếu sót. Tôi rất mong nhận được sự chỉ bảo và đóng góp ý kiến của quý Thầy/Cô để bài làm của tôi được hoàn thiện hơn.

Tôi xin chân thành cảm ơn!

Hà Nội, ngày 9 tháng 7 năm 2026

Tác giả luận văn

Phạm Tuấn Thành

MỤC LỤC

DANH MỤC KÝ HIỆU, CÁC CHỮ VIẾT TẮT

Viết tắt	Tiếng Anh	Diễn giải
API	Application Programming Interface	Giao diện lập trình ứng dụng
CI/CD	Continuous Integration/Continuous Deployment	Hệ thống tích hợp, triển khai liên tục
GPU	Graphics Processing Unit	Bộ xử lý đồ họa
HTTP	Hypertext Transfer Protocol	Giao thức truyền tải văn bản
IoT	Internet of Things	Internet vạn vật
MLOPS	Machine Learning Operations	Phương pháp để quản lý và triển khai các mô hình học máy
SQL	Structured Query Language	Ngôn ngữ truy vấn có cấu trúc
TTNT	AI - Artificial Intelligence	Trí tuệ nhân tạo
URL	Uniform Resource Locator	Hệ thống định vị tài nguyên

DANH MỤC BẢNG

DANH MỤC HÌNH VẼ VÀ ĐỒ THỊ

MỞ ĐẦU

Tính cấp thiết của luận văn

Trong những năm gần đây, chuyển đổi số đã trở thành xu hướng tất yếu và là yêu cầu mang tính chiến lược đối với các cơ quan nhà nước, doanh nghiệp và tổ chức trên toàn thế giới. Sự phát triển mạnh mẽ của công nghệ dữ liệu lớn (Big Data), điện toán đám mây (Cloud Computing), Internet vạn vật (IoT) và đặc biệt là trí tuệ nhân tạo (Artificial Intelligence – AI - TTNT) đang làm thay đổi căn bản phương thức vận hành, quản lý và cung cấp dịch vụ trong mọi lĩnh vực của đời sống kinh tế – xã hội.

Học máy (Machine Learning), đang đóng vai trò là một nhánh cốt lõi của TTNT, ngày càng được ứng dụng rộng rãi trong nhiều lĩnh vực như tài chính – ngân hàng (chấm điểm tín dụng, phát hiện gian lận), y tế (hỗ trợ chẩn đoán, phân tích hình ảnh y khoa), thương mại điện tử (gợi ý sản phẩm), sản xuất công nghiệp (bảo trì dự đoán), giáo dục (cá nhân hóa lộ trình học tập) và quản lý đô thị thông minh. Các mô hình học máy ngày càng phức tạp, sử dụng khối lượng dữ liệu khổng lồ và yêu cầu hạ tầng tính toán mạnh mẽ.

Tuy nhiên, trên thực tế, việc xây dựng được một mô hình học máy có độ chính xác cao mới chỉ là bước khởi đầu. Thách thức lớn hơn sẽ nằm ở việc triển khai mô hình (deployment), vận hành (operation), giám sát (monitoring), cập nhật (retraining) và quản lý vòng đời (lifecycle management) trong môi trường thực tế. Chính tại đây, khái niệm MLOPS (Machine Learning Operations) ra đời như một hướng tiếp cận tất yếu nhằm giải quyết khoảng cách giữa nghiên cứu – phát triển mô hình và triển khai ứng dụng thực tiễn.

Cuối cùng, xu hướng chuyển đổi số thực chất tại Việt Nam. Tại thị trường Việt Nam, các doanh nghiệp đang tiến dần từ giai đoạn "số hóa dữ liệu" sang giai đoạn "khai thác dữ liệu bằng TTNT". Tuy nhiên, khoảng cách giữa một mô hình thử nghiệm trên máy tính cá nhân và một ứng dụng AI với mục đích phục vụ hàng triệu người dùng là rất lớn. Việc thiếu đi một "khung xương" MLOPS khiến nhiều dự án chỉ dừng lại ở mức thử nghiệm và không đem lại giá trị.

Đề tài tập trung vào việc tích hợp các công cụ mã nguồn mở sẽ cung cấp một lộ trình thực thi khả thi, giúp các doanh nghiệp và tổ chức trong nước nhanh chóng hiện thực hóa sức mạnh của TTNT vào thực tiễn sản xuất, kinh doanh.

Việc nghiên cứu đề tài "Tích hợp các nền tảng mã nguồn mở phục vụ phát triển mô hình học máy MLOPS và ứng dụng" không chỉ mang tính thời sự về mặt công nghệ mà còn có ý nghĩa thực tiễn sâu sắc. Nó giải quyết đồng thời ba bài toán: Chuẩn hóa quy trình phát triển TTNT, làm chủ công nghệ lõi và tối ưu chi phí và đảm

bảo được tính bền vững và khả năng mở rộng cho các ứng dụng thông minh trong kỷ nguyên số.

Sự cần thiết phải tiến hành nghiên cứu luận văn

Việc nghiên cứu và xây dựng một nền tảng số MLOPS phục vụ phát triển mô hình học máy là cần thiết vì các lý do sau:

- **Tăng cường hiệu quả phát triển và triển khai mô hình ML:**

  MLOPS giúp tự động hóa các quy trình như thu thập dữ liệu, huấn luyện, triển khai và giám sát mô hình, từ đó giảm thiểu thời gian, chi phí và sai sót so với phương pháp thủ công. [1]

- **Đáp ứng nhu cầu nội địa:**

  Các nền tảng mã nguồn mở cho phép các doanh nghiệp, đặc biệt là doanh nghiệp vừa và nhỏ tại Việt Nam, tiếp cận công nghệ tiên tiến mà không cần đầu tư lớn vào các giải pháp độc quyền [2, 3]. Điều này đặc biệt quan trọng trong bối cảnh Việt Nam đang thúc đẩy chuyển đổi số.

- **Thúc đẩy nghiên cứu và đổi mới sáng tạo:**

  Việc xây dựng nền tảng MLOPS không chỉ hỗ trợ ứng dụng thực tiễn mà còn tạo cơ hội nghiên cứu các phương pháp mới trong quản lý vòng đời mô hình ML, góp phần nâng cao năng lực nghiên cứu khoa học và công nghệ của Việt Nam.

Do đó, việc nghiên cứu và xây dựng nền tảng số MLOPS không chỉ là yêu cầu cấp thiết để đáp ứng nhu cầu thực tiễn mà còn góp phần thúc đẩy sự phát triển bền vững của ngành công nghệ AI tại Việt Nam, đồng thời khẳng định vị trí trong bối cảnh cạnh tranh công nghệ toàn cầu.

Mục tiêu của luận văn

Mục tiêu của luận văn là nghiên cứu các phương pháp và công nghệ liên quan đến MLOPS, từ đó đề xuất và xây dựng một nền tảng số hỗ trợ quản lý và tự động hóa quá trình phát triển, triển khai và vận hành các mô hình học máy. Việc xây dựng nền tảng này nhằm nâng cao hiệu quả phát triển hệ thống học máy, giảm thiểu các thao tác thủ công và tạo điều kiện thuận lợi cho việc triển khai mô hình vào môi trường thực tế.

Cụ thể, luận văn hướng tới các mục tiêu sau:

- **Nghiên cứu tổng quan về học máy và vòng đời phát triển mô hình học máy:**

  Nội dung sẽ làm rõ các khái niệm cơ bản liên quan đến học máy, các bước trong vòng đời phát triển mô hình như thu thập dữ liệu, tiền xử lý dữ liệu, huấn luyện mô hình, đánh giá mô hình, triển khai và giám sát mô hình.

Việc nghiên cứu vòng đời mô hình học máy giúp xác định những giai đoạn cần được hỗ trợ bởi nền tảng MLOPS.

- **Phân tích các khó khăn và hạn chế trong quá trình phát triển và triển khai mô hình học máy:**
  Mục tiêu của nội dung sẽ tìm hiểu những vấn đề thường gặp trong các dự án học máy như việc quản lý dữ liệu, theo dõi thí nghiệm, kiểm soát phiên bản mô hình, triển khai mô hình vào môi trường sản xuất và giám sát hiệu năng của mô hình sau khi triển khai. Việc phân tích các khó khăn này giúp xác định các yêu cầu cần thiết cho nền tảng MLOPS.
- **Nghiên cứu các kiến trúc và công nghệ MLOPS hiện nay:**
  Nội dung sẽ tập trung tìm hiểu các nền tảng và công cụ MLOPS phổ biến như Kubeflow, MLflow và các giải pháp triển khai trên nền tảng điện toán đám mây. Việc nghiên cứu các công nghệ hiện có giúp lựa chọn các giải pháp phù hợp và làm cơ sở cho việc thiết kế nền tảng MLOPS trong đề tài.
- **Xây dựng và thử nghiệm hệ thống nền tảng MLOPS:**
  Nội dung sẽ triển khai một hệ thống thử nghiệm dựa trên kiến trúc đã đề xuất. Hệ thống sẽ được sử dụng để hỗ trợ quá trình phát triển và triển khai một số mô hình học máy nhằm đánh giá tính khả thi và hiệu quả của nền tảng.

Nội dung nghiên cứu luận văn

Để đạt được các mục tiêu nghiên cứu đã đề ra, luận văn tập trung thực hiện các nội dung nghiên cứu chính sau:

- **Tổng quan về trí tuệ nhân tạo, học máy và vòng đời mô hình học máy:**
  Cung cấp cơ sở lý thuyết cho đề tài bằng cách trình bày các khái niệm cơ bản về trí tuệ nhân tạo và học máy, cũng như các bước trong vòng đời phát triển mô hình học máy. Qua đó, luận văn làm rõ vai trò của từng giai đoạn trong quá trình phát triển hệ thống học máy.
- **Nghiên cứu các phương pháp nghiên cứu mô hình MLOPS:**
  Nội dung sẽ tập trung tìm hiểu các phương pháp quản lý và tự động hóa quy trình phát triển mô hình học máy. Các công nghệ liên quan đến container hóa, pipeline tự động, quản lý dữ liệu và triển khai mô hình cũng được nghiên cứu nhằm phục vụ việc xây dựng nền tảng MLOPS.
- **Phân tích và đánh giá các nền tảng MLOPS hiện có:**
  Luận văn sẽ tiến hành nghiên cứu một số nền tảng MLOPS phổ biến nhằm phân tích ưu điểm, hạn chế và khả năng ứng dụng của chúng trong thực tế.

Kết quả phân tích sẽ giúp xác định những chức năng cần thiết cho nền tảng MLOPS được xây dựng trong đề tài.

- **Xây dựng hệ thống thử nghiệm và đánh giá kết quả:**
  Nội dung bao gồm việc triển khai thử nghiệm nền tảng MLOPS và sử dụng hệ thống để phát triển và triển khai một số mô hình học máy. Kết quả thử nghiệm sẽ được phân tích nhằm đánh giá hiệu quả và tính khả thi của hệ thống.

Cấu trúc luận văn

Về phần cấu trúc luận văn, ngoài phần Mở đầu và Kết luận luận văn, luận văn sẽ được chia ra làm ba chương với các nội dung:

Chương 1: GIỚI THIỆU TỔNG QUÁT VỀ MLOPS

Chương 2: CƠ SỞ LÝ THUYẾT PHỤC VỤ XÂY DỰNG HỆ THỐNG MLOPS

Chương 3: THIẾT KẾ VÀ XÂY DỰNG HỆ THỐNG MLOPS

CHƯƠNG 1: GIỚI THIỆU TỔNG QUÁT VỀ MLOPS

Chương 1 sẽ trình bày tổng quan về tình hình nghiên cứu mô hình MLOPS qua đó xác định mức độ sẵn sàng và nhu cầu đối với MLOPS trong khối cơ sở đào tạo CNTT tại Việt Nam; từ đó đề xuất các yêu cầu chức năng/phi chức năng và gợi ý kiến trúc triển khai phù hợp.

1.1. Tổng quan tình hình nghiên cứu trên thế giới và ở Việt Nam

1.1.1. Trên thế giới:

Trên thế giới, lĩnh vực học máy đã trở thành một trong những trụ cột quan trọng của cách mạng công nghiệp 4.0, thúc đẩy sự phát triển của trí tuệ nhân tạo trong nhiều ngành như y tế, tài chính, giao thông, sản xuất, và thương mại điện tử. Tuy nhiên, việc phát triển và triển khai các mô hình học máy một cách hiệu quả, đặc biệt trong môi trường sản xuất thực tế, đòi hỏi sự quản lý chặt chẽ và tích hợp các quy trình phức tạp. Từ nhu cầu này, khái niệm MLOPS ra đời, kết hợp các nguyên tắc của DevOps với quy trình phát triển ML để tự động hóa, tối ưu hóa và quản lý vòng đời của mô hình học máy từ giai đoạn thu thập dữ liệu, huấn luyện, triển khai đến giám sát và bảo trì. MLOPS không chỉ tập trung vào hiệu suất mô hình mà còn giải quyết các thách thức như khả năng mở rộng, quản lý dữ liệu lớn, tính minh bạch, bảo mật, và khả năng tái sử dụng mô hình. [4]

Các tập đoàn công nghệ lớn như Google, Amazon, Microsoft, và IBM đã tiên phong trong việc phát triển các nền tảng MLOPS thương mại. Chẳng hạn, Google Cloud AI Platform cung cấp các công cụ tích hợp để xây dựng, huấn luyện và triển khai mô hình ML trên quy mô lớn, hỗ trợ các quy trình như quản lý dữ liệu, kiểm tra mô hình, và giám sát hiệu suất theo thời gian thực. Tương tự, AWS của Amazon và Azure Machine Learning của Microsoft cung cấp các dịch vụ toàn diện, từ xử lý dữ liệu thô đến triển khai mô hình trên đám mây, tích hợp với các công cụ DevOps để đảm bảo quy trình phát triển liền mạch. Các nền tảng này thường được thiết kế để hoạt động trên hạ tầng đám mây, cung cấp các tính năng như tự động hóa hyperparameter tuning, quản lý phiên bản mô hình, và tích hợp với các hệ thống CI/CD (Continuous Integration/Continuous Deployment). Ngoài ra, các công ty như Databricks, H2O.ai, và Kubeflow cũng đóng góp vào sự phát triển của MLOPS thông qua các giải pháp mã nguồn mở hoặc thương mại, tập trung vào việc xử lý dữ liệu lớn và tích hợp với các hệ sinh thái công nghệ khác. [5]

Bên cạnh các giải pháp thương mại, cộng đồng nghiên cứu học thuật cũng tích cực đóng góp vào sự phát triển của MLOPS. Các công trình nghiên cứu trên thế giới đã tập trung vào nhiều khía cạnh của MLOPS, bao gồm tối ưu hóa quy trình huấn

luyện mô hình, phát triển các phương pháp giám sát hiệu suất mô hình trong môi trường thực tế (model drift detection), và giải quyết các vấn đề về bảo mật dữ liệu và quyền riêng tư, chẳng hạn như thông qua các kỹ thuật học liên kết (Federated Learning) hoặc mã hóa dữ liệu. Một số nghiên cứu nổi bật đã được công bố tại các hội nghị lớn như ICML (International Conference on Machine Learning) và KDD, tập trung vào việc cải thiện khả năng tái sử dụng mô hình, giảm chi phí tính toán, và tăng cường khả năng giải thích của các mô hình ML. Tuy nhiên, với các nền tảng MLOPS hiện nay, đặc biệt là các giải pháp thương mại, thường yêu cầu chi phí cao và phụ thuộc vào hạ tầng đám mây, gây khó khăn cho các tổ chức nhỏ hoặc ở các quốc gia có nguồn lực hạn chế. Ngoài ra, việc tích hợp MLOPS với các hệ thống nội bộ của doanh nghiệp hoặc các yêu cầu đặc thù về dữ liệu và quy định pháp luật tại từng quốc gia vẫn là một thách thức lớn.

Tuy MLOPS đã đạt được nhiều thành tựu song các nghiên cứu và giải pháp hiện tại vẫn còn gặp một số hạn chế. Ví dụ, việc xử lý dữ liệu không đồng nhất, đảm bảo tính công bằng của mô hình, và quản lý vòng đời mô hình trong các môi trường phức tạp với dữ liệu thời gian thực vẫn là những vấn đề chưa được giải quyết triệt để. Hơn nữa, phần lớn các nền tảng MLOPS quốc tế được thiết kế dựa trên bối cảnh công nghệ và kinh tế của các nước phát triển, do đó khó áp dụng trực tiếp tại các quốc gia đang phát triển, nơi hạ tầng công nghệ, nguồn nhân lực, và ngân sách còn hạn chế. Những yếu tố này tạo động lực mạnh mẽ cho việc nghiên cứu và phát triển các nền tảng MLOPS phù hợp hơn với các thị trường mới nổi, bao gồm cả Việt Nam, để có thể đáp ứng nhu cầu ngày càng tăng về ứng dụng AI trong các lĩnh vực kinh tế và xã hội.

1.1.2. Tại Việt Nam:

Trong những năm qua, Việt Nam đã có những bước tiến đáng kể trong những năm gần đây, ứng dụng rộng rãi trong các lĩnh vực như công nghệ thông tin, công nghiệp, nông nghiệp, y tế, tài chính, giao thông vận tải, logistics, và quản lý nhà nước. Theo báo cáo chiến lược AI quốc gia đến năm 2030, TTNT đã hỗ trợ tự động hóa nhiều công đoạn, nâng cao hiệu quả quản lý nhà nước và thúc đẩy phát triển kinh tế - xã hội. Các giải pháp TTNT đang được triển khai trong các ngành như tài chính (phân tích rủi ro, phát hiện gian lận tài chính), lĩnh vực ngân hàng (chatbot, tư vấn tài chính), thương mại điện tử (hệ thống đề xuất sản phẩm), y tế (chẩn đoán hình ảnh, hỗ trợ điều trị), và giao thông (quản lý giao thông thông minh, tối ưu hóa logistics). [6]

Về nghiên cứu và phát triển, Việt Nam đang tận dụng các nguồn lực dữ liệu thực tiễn phong phú, với sự trưởng thành của đội ngũ nghiên cứu trẻ được đào tạo bài

bản tại các nước phát triển, cùng với các tiến bộ trong khoa học dữ liệu và học máy. Các trường đại học danh tiếng như Đại học Quốc gia Hà Nội, Đại học Bách Khoa, các viện nghiên cứu, và cùng các công ty công nghệ hàng đầu như FPT, Viettel, CMC, VNG, VNPT, VinGroup đã tiên phong trong các dự án TTNT.

Đã có hơn 117 đề tài nghiên cứu TTNT cấp nhà nước được phê duyệt trong các chương trình tính từ năm 2006 đến 2020. Các đề tài tập trung vào các lĩnh vực như hệ thống thông minh, xử lý ngôn ngữ tự nhiên, xử lý ảnh (phân loại hạt điều, phát hiện bất thường trên ảnh X-Quang), nhận dạng giọng nói, tương tác người-máy, tin sinh học, và an ninh mạng (phát hiện xâm nhập, giám sát an toàn hệ thống). [7,8]

1.2. Tổng quan về trí tuệ nhân tạo và học máy

Trong những năm gần đây, TTNT đã trở thành một trong những lĩnh vực nghiên cứu và ứng dụng quan trọng trong khoa học máy tính. TTNT được hiểu là khả năng của máy tính hoặc hệ thống phần mềm thực hiện các nhiệm vụ mà thông thường cần đến trí tuệ của con người, như nhận dạng hình ảnh, xử lý ngôn ngữ tự nhiên, học hỏi từ dữ liệu và đưa ra quyết định.

Một trong những nhánh quan trọng nhất của TTNT là học máy. Học máy cho phép máy tính tự động học từ dữ liệu và cải thiện hiệu suất theo thời gian mà không cần lập trình chi tiết cho từng nhiệm vụ cụ thể. Thay vì xây dựng các quy tắc cứng, các thuật toán học máy sử dụng dữ liệu để tìm ra các mẫu (patterns) và mối quan hệ trong dữ liệu.

Các phương pháp đặc thù của Học máy hiện nay được chia thành ba nhóm chính:

- **Học có giám sát (Supervised Learning):**

  Là phương pháp học từ dữ liệu đã được gán nhãn. Mục tiêu của thuật toán là học được mối quan hệ giữa dữ liệu đầu vào và đầu ra để dự đoán kết quả cho dữ liệu mới. Các bài toán phổ biến gồm phân loại và hồi quy. [6] [9]

- **Học không giám sát (Unsupervised Learning):**

  Thuật toán học từ dữ liệu không có nhãn, nhằm tìm ra cấu trúc hoặc mối quan hệ tiềm ẩn trong dữ liệu. Các kỹ thuật phổ biến bao gồm phân cụm dữ liệu và giảm chiều dữ liệu. [6] [9]

Nhờ sự phát triển mạnh mẽ của phần cứng, đặc biệt là GPU và các nền tảng điện toán đám mây, cùng với sự gia tăng nhanh chóng của dữ liệu, các mô hình học máy ngày càng trở nên mạnh mẽ và được ứng dụng rộng rãi trong nhiều lĩnh vực như tài chính, y tế, thương mại điện tử, giao thông thông minh và sản xuất công nghiệp.

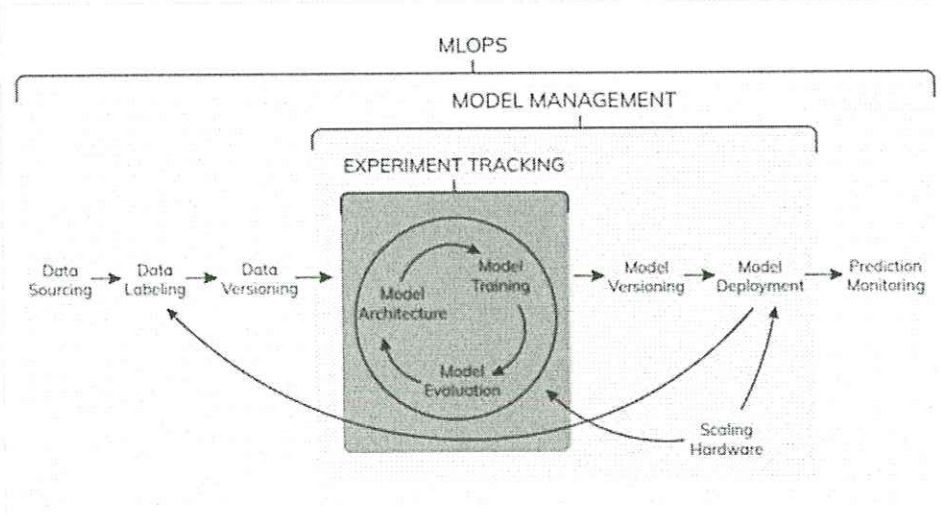
Tuy nhiên, việc xây dựng và triển khai các mô hình học máy trong môi trường thực tế không chỉ dừng lại ở giai đoạn huấn luyện mô hình mà còn liên quan đến toàn bộ quá trình quản lý vòng đời của mô hình. Điều này đặt ra nhu cầu cần có các giải pháp và nền tảng hỗ trợ quản lý hiệu quả quá trình phát triển và vận hành các hệ thống học máy.

1.3. Khái niệm và vai trò của MLOPS

Machine Learning Operations (MLOPS) là một phương pháp để quản lý và triển khai các mô hình học máy trong một môi trường Production. Mục tiêu của MLOPS là cung cấp các công cụ, quy trình và phương pháp để tự động hóa quá trình huấn luyện, triển khai và quản lý các mô hình học máy.

MLOPS sẽ tập trung vào việc tạo ra một chu trình phát triển phần mềm cho các mô hình máy học, từ việc thu thập và xử lý dữ liệu, huấn luyện mô hình, triển khai mô hình và giám sát mô hình trong môi trường Production. Các công cụ MLOPS bao gồm các hệ thống quản lý mã nguồn, các framework và thư viện mã nguồn mở, các nền tảng huấn luyện mô hình trên hạ tầng cloud, các hệ thống tự động hóa triển khai và cập nhật mô hình, và các công cụ giám sát và kiểm tra độ tin cậy của mô hình.

MLOPS giúp cho các nhà phát triển và nhà khoa học dữ liệu có thể phát triển các mô hình máy học một cách nhanh chóng và hiệu quả hơn, tăng tính ổn định và độ tin cậy của các mô hình, và giảm thời gian và chi phí cho các bộ phận quản lý và triển khai các mô hình. [8] [10] [11]



Hình 1.1 Chu trình MLOPS

1.4. Nhu cầu sử dụng MLOPS tại Việt Nam

Với việc TTNT đang dần trở thành một trong những động lực chính thúc đẩy

quá trình chuyển đổi số, đã có rất nhiều lĩnh vực có nhu cầu khai thác dịch vụ nền tảng MLOPS với mục đích phục vụ chia sẻ, phát triển và ứng dụng mô hình trí tuệ nhân tạo cho hệ sinh thái khoa học

1.4.1. Nhu cầu MLOPS trong môi trường đào tạo:

Dựa vào bối cảnh quy mô và mạng lưới đào tạo đang mở rộng nhanh và nhu cầu rõ rệt cho năng lực tổ chức dạy–học lĩnh vực TTNT, nhu cầu triển khai MLOPS trong môi trường đào tạo có thể được tóm tắt qua 4 trục ưu tiên chính, hướng tới mục tiêu minh bạch, tái lập và tuân thủ các tiêu chuẩn quốc tế [12]

- **Xây nền móng dữ liệu và thí nghiệm:**

Xây dựng kho dữ liệu đặc trưng có nhiều phiên bản và lịch sử nhằm mục đích quản lý xuyên suốt nhiều học kỳ. Ngoài ra, có thể ghi lại tham số và các chỉ số để so sánh kết quả, phục vụ chấm điểm tự động và ngăn chặn gian lận học thuật.

- **Quản lý mô hình:**

Thiết lập cơ chế kiểm duyệt qua nhiều cấp trước khi đưa mô hình vào ứng dụng thực tế hoặc chấm điểm.

- **Tự động hóa Pipeline:**

Xây dựng các pipeline mẫu giúp giảm công sức lặp lại nhằm rút ngắn thời gian trong việc triển khai bài thực hành.

- **Giám sát vận hành:**

Mục đích theo dõi độ chính xác, hiện tượng trôi dữ liệu, hiệu suất GPU và chi phí vận hành.

1.4.2. Nhu cầu MLOPS trong môi trường doanh nghiệp

Đối với khối doanh nghiệp, nền tảng MLOPS cần được thiết kế như một hệ quản trị toàn doanh nghiệp cho TTNT. Mục tiêu cốt lõi nhằm cân bằng giữa tốc độ ra mắt sản phẩm và việc kiểm soát rủi ro, tuân thủ pháp lý. [12]

- **Hệ thống quản lý TTNT**

MLOPS đóng vai trò là hạ tầng kỹ thuật thực thi hỗ trợ doanh nghiệp thiết lập các chính sách về đạo đức, minh bạch và trách nhiệm giải trình.

- **Quản lý mô hình:**

Trong bối cảnh các quy định về bảo vệ dữ liệu cá nhân ngày càng nghiêm

ngặt, MLOPS hỗ trợ cung cấp các cơ chế định danh, đánh giá rủi ro và nhật ký xử lý (audit trail) xuyên suốt vòng đời hệ thống.

- **Tối ưu hóa khả năng vận hành sản xuất**

Doanh nghiệp yêu cầu khả năng giám sát hợp nhất từ độ chính xác đến chi phí hạ tầng (GPU/Cloud) theo thời gian thực. Khả năng phát hành thử nghiệm (Canary/A-B testing) và tự động quay lui (Rollback) giúp đảm bảo tính sẵn sàng cao cho các dịch vụ AI.

- **Thích ứng với xu thế TTNT**

MLOPS hiện đại cần mở rộng để quản lý các thành phần đặc thù của AI tạo sinh như quản lý prompt hoặc cơ chế phản hồi từ người dùng

Các công cụ và nền tảng MLOPS hiện nay

Trong những năm gần đây, nhiều công cụ và nền tảng đã được phát triển nhằm hỗ trợ triển khai MLOPS. Các công cụ này giúp tự động hóa quy trình phát triển và triển khai mô hình học máy. [13] [14]

Một số công cụ chính trong MLOPS đang phổ biến hiện nay bao gồm:

Kubeflow

Kubeflow là một nền tảng mã nguồn mở được xây dựng trên Kubernetes nhằm hỗ trợ triển khai và quản lý các pipeline học máy. Kubeflow cung cấp các công cụ để huấn luyện, triển khai và quản lý mô hình trong môi trường container hóa.

MLflow

MLflow là nền tảng giúp theo dõi các thí nghiệm, lưu trữ và triển khai mô hình. MLflow giúp các nhà nghiên cứu và phát triển mô hình có thể tái sử dụng mô hình và quản lý các phiên bản mô hình trong suốt vòng đời của chúng.

1.5. Kết luận chương

Trong chương này đã trình bày về tình hình nghiên cứu mô hình MLOPS toàn cầu nói chung và Việt Nam nói riêng, cùng với tổng quan lý thuyết về khái niệm và vai trò của mô hình, qua đó sử dụng để làm cơ sở phát triển và ứng dụng.

CHƯƠNG 2 : CƠ SỞ LÝ THUYẾT PHỤC VỤ XÂY DỰNG HỆ THỐNG MLOPS

Chương 2 sẽ trình bày tổng quan về các mã nguồn mở với mục đích phục vụ nghiên cứu mô hình MLOPS qua hai phân hệ Data Ingestion (Thu thập/nhập dữ liệu) và Data Augmentation (Tăng cường dữ liệu)

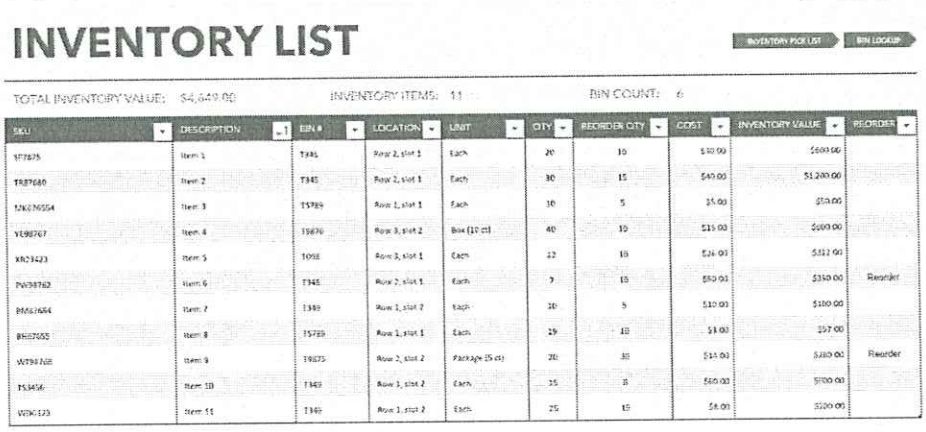
2.1. Phân hệ Data Ingestion

Phân hệ Data Ingestion là một thành phần trong hệ thống dữ liệu MLOPS có nhiệm vụ lấy dữ liệu từ nhiều nguồn khác nhau và đưa vào hệ thống lưu trữ hoặc xử lý

Các nguồn dữ liệu đầu vào bao gồm:

2.1.1. Nguồn dữ liệu dạng bảng:

Dữ liệu dạng bảng là tập hợp các bản ghi dữ liệu được sắp xếp tổ chức lần lượt theo từng hàng và cột, trong đó mỗi hàng biểu diễn một thực thể hoặc một quan sát và mỗi cột biểu diễn một thuộc tính có kiểu dữ liệu xác định (chuỗi, số, ngày, boolean, v.v.). [1] [15]



Hình 2.1: Minh họa về dữ liệu dạng bảng

Dữ liệu dạng bảng là nền tảng cho hầu hết các hoạt động phân tích dữ liệu, báo cáo và mô hình hoá, đóng vai trò nền tảng nhờ cấu trúc hàng–cột tối ưu, giúp việc truy vấn, tính toán và xử lý diễn ra nhanh chóng và chính xác hiệu quả. Cụ thể, dữ liệu bảng thường được dùng cho các mục đích sau:

- Quản trị và báo cáo: tổng hợp số liệu thời gian thực hoặc theo kỳ để tạo biểu đồ, KPI, bảng tổng hợp hỗ trợ lãnh đạo ra quyết định dựa trên số liệu thực tế.
- Khám phá dữ liệu (EDA): làm sạch, thống kê mô tả, xác định xu hướng và nhận diện các điểm bất thường trước khi xây dựng mô hình.

- Chuẩn hoá dữ liệu và ETL: Hợp nhất các nguồn dữ liệu rời rạc về một cấu trúc chung (schema), sẵn sàng cho các hệ thống hạ nguồn.
- Machine learning và mô phỏng: feature engineering, tạo tập huấn luyện/kiểm thử, và chuyển đổi biểu diễn cho các mô hình học máy.
- Tích hợp hệ thống (data integration): phối hợp dữ liệu huấn luyện từ nhiều hệ thống giao dịch, CRM, ERP, cảm biến... để tạo ra cái nhìn toàn cảnh.

Lưu trữ lịch sử và truy vết: lưu lại các bản sao lịch sử (snapshot) hoặc nhật ký hệ thống dạng bảng với mục đích truy vết, phục hồi hoặc phân tích xu hướng dài hạn. [1] [15]

2.1.2. Nguồn dữ liệu dạng tuần tự (Sequential Data)

Dữ liệu dạng tuần tự là loại dữ liệu trong đó thứ tự xuất hiện của các phần tử mang ý nghĩa quan trọng và có ảnh hưởng trực tiếp đến quá trình phân tích, xử lý hoặc dự đoán. Mỗi phần tử trong chuỗi dữ liệu thường phụ thuộc vào một hoặc nhiều phần tử trước đó, thể hiện mối quan hệ theo thời gian, không gian hoặc logic. Khác với dữ liệu tĩnh (non-sequential data) – nơi các bản ghi độc lập và không ảnh hưởng lẫn nhau – dữ liệu tuần tự chứa ngữ cảnh (context), nghĩa là giá trị hiện tại không thể hiểu hoặc dự đoán chính xác nếu tách rời khỏi lịch sử của nó. Nhờ đặc tính này, dữ liệu tuần tự trở thành nền tảng quan trọng trong các hệ thống xử lý dữ liệu động, nơi yếu tố thời gian hoặc trình tự là không thể bỏ qua. [16]

Dữ liệu dạng tuần tự có thể tồn tại dưới nhiều biến thể, đây là các biến thể thông dụng nhất của dữ liệu dạng tuần tự, tuỳ vào mục đích sử dụng và lĩnh vực của hệ thống hay ứng dụng. Một số loại phổ biến bao gồm:

- **Chuỗi kí tự (String Sequence)**

Là chuỗi các ký tự (chữ cái, số, ký hiệu) sắp xếp theo thứ tự cụ thể, được sử dụng rộng rãi trong xử lý ngôn ngữ tự nhiên (NLP), mã hóa văn bản, hoặc trong phân

- **Chuỗi thời gian (Time Series):**

Là tập hợp các điểm dữ liệu được thu thập tuần tự theo thời gian, trong đó các điểm dữ liệu được đo lường và ghi lại theo các khoảng thời gian liên tiếp. Các dữ liệu sẽ thường xảy ra cách đều nhau về mặt tần suất (theo ngày, giờ, phút…).

- **Chuỗi sự kiện (Event Sequence)**

Là danh sách các sự kiện hoặc hành động xảy ra liên tiếp theo thời gian. Ví dụ: nhật ký hệ thống (system logs), hành vi người dùng trên trang web, hoặc các giao dịch trong ngân hàng. Phân tích chuỗi sự kiện giúp phát hiện lỗi, tối ưu hóa luồng hoạt động hoặc phát hiện hành vi bất thường.

- **Dữ liệu hình ảnh động hoặc video (Sequential Frames)**

Là chuỗi các khung hình được ghi nhận liên tục trong thời gian, kết hợp cả thông tin về không gian (hình ảnh) và thời gian (chuyển động). Mỗi khung hình là một ảnh tĩnh, nhưng khi kết hợp lại, chúng mô tả chuyển động hoặc sự thay đổi theo thời gian. Ứng dụng trong nhận dạng hành động, theo dõi vật thể, phân tích chuyển động, hoặc hệ thống giám sát thông minh.

Mục tiêu chính của việc thu thập và xử lý dữ liệu dạng tuần tự là khai thác mối quan hệ phụ thuộc theo thời gian hoặc thứ tự để phục vụ cho các nhiệm vụ như:

Nhiệm vụ	**Mô tả**
Dự báo (Forecasting)	Sử dụng chuỗi dữ liệu trong quá khứ với mục đích dự đoán giá trị trong tương lai. Ví dụ: Dự báo giá cổ phiếu, chứng khoán
Nhận dạng mẫu (Pattern Recognition)	Phát hiện các quy luật hoặc cấu trúc lặp lại trong dữ liệu tuần tự. Ví dụ: Nhận dạng giọng nói, chữ viết, hành vi người dùng
Phân tích hành vi (Behavior Analysis)	Phân tích chuỗi hành động của con người để tìm ra xu hướng, thói quen hoặc các rủi ro tiềm ẩn. Ví dụ: Phân tích hành vi người dung nhằm mục đích phát hiện gian lận
Phát hiện bất thường (Anomaly Detection)	Xác định các điểm dữ liệu bất thường không tuân theo quy luật chung. Ví dụ: Ứng dụng cho an ninh mạng, chẩn đoán y tế, giám sát công nghiệp.

Bảng 2.1: Nhiệm vụ thu thập và xử lý dữ liệu dạng tuần tự

Dữ liệu dạng tuần tự là một trong những nguồn dữ liệu cốt lõi của thời đại kỉ nguyên số, có vai trò quan trọng trong việc hiểu và dự đoán hành vi, xu hướng của hệ thống và con người.

2.1.3. Nguồn dữ liệu dạng Streaming

Dữ liệu Streaming là luồng dữ liệu liên tục được tạo ra, truyền tải dữ liệu và xử lý theo thời gian thực hoặc gần thời gian thực, trong đó dữ liệu được xử lý ngay khi phát sinh thay vì lưu trữ cục bộ, gom theo batch. Khác với dữ liệu batch truyền

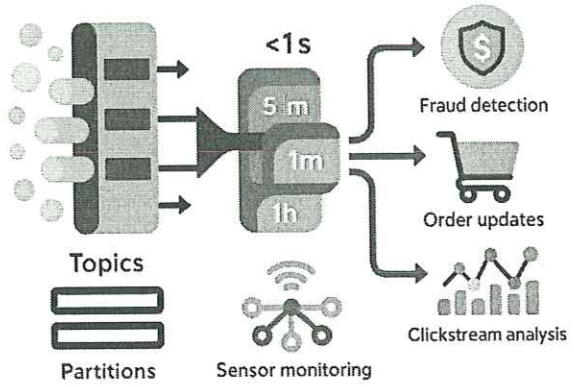
thống được xử lý theo từng lô với khoảng thời gian nhất định, dữ liệu streaming được sinh ra và xử lý một cách liên tục, thường không có điểm bắt đầu hay kết thúc rõ ràng. Việc khai thác luồng dữ liệu streaming sẽ giúp đưa ra các quyết định kịp thời, phát hiện những điểm bất cập nhanh chóng và giúp tối ưu hóa trải nghiệm người dùng. [17]

Các đặc điểm chính nhất của dữ liệu streaming là tốc độ sinh dữ liệu thường liên tục và tốc độ rất cao, có thể lên đến hàng nghìn hoặc hàng triệu bản ghi mỗi giây, dữ liệu được truyền đi liên tục. Do đó, các hệ thống xử lý streaming phải có khả năng mở rộng linh hoạt và độ trễ thấp để đảm bảo dữ liệu được xử lý kịp thời. Ngoài ra, dữ liệu streaming thường có tính chất không đồng nhất, có thể đến từ nhiều nguồn khác nhau với định dạng và cấu trúc đa dạng.

Ví dụ:

Một trong những ví dụ thực tế về dữ liệu streaming là dữ liệu từ các hệ thống trong lĩnh vực thương mại điện tử. Hàng ngàn người tiêu dung truy cập website, các hành động tìm kiếm mặt hàng, xem chi tiết phụ kiện, nhấn vào giỏ hang hay thanh toán sẽ được ghi nhận và gửi liên tục về hệ thống. Hệ thống sẽ phân tích luồng dữ liệu này ngay lập tức đưa ra đề xuất cho người tiêu dung các mặt hàng phù hợp, hay phân tích các hành vi gian lận trong mua bán, qua đó vừa nâng cao trải nghiệm người dung, vừa đảm bảo hệ thống được hoạt động ổn định.

Ngoài ra, các trang mạng xã hội cũng là nơi sinh ra một lượng dữ liệu streaming khổng lồ. Tính trên các nền tảng phổ biến như Facebook hay Youtube, đã có hàng trăm, hàng triệu người dùng đồng thời đăng tải bài viết, bình luận, thích và chia sẻ nội dung. Các hệ thống phân tích phải xử lý luồng dữ liệu này để cá nhân hóa các tin tức, phát hiện xu hướng đang nổi, lọc các nội dung có tính chất spam và nội dung vi phạm. [17]



Hình 2.2: Minh họa về dữ liệu streaming

2.1.4. Dữ liệu Sự kiện (Event)

Dữ liệu sự kiện (Event) được coi là loại dữ liệu ghi nhận trong hệ thống về những thay đổi trạng thái hoặc hành động xảy ra trong hệ thống tại một thời điểm nhất định. Mỗi event là một thực thể độc lập chứa đựng thông tin về một sự việc, tương tác hay thay đổi trạng thái đã xảy ra, bao gồm loại sự kiện, thời điểm xảy ra, đối tượng liên quan và các thuộc tính chi tiết khác. Khác với streaming data tập trung vào luồng dữ liệu liên tục, dữ liệu sự kiện sẽ đóng vai trò phản ánh trực tiếp hoạt động của người dùng hoặc hệ thống theo thời gian thực. [18]

Đặc điểm quan trọng của dữ liệu event là tính bất biến, khi các event đã xảy ra và được ghi nhận thì nó sẽ không thể bị chỉnh sửa. Event chỉ có thể được bổ sung thêm các event mới để phản ánh trạng thái tiếp theo của hệ thống. Điều này giúp duy trì lịch sử đầy đủ về mọi thay đổi trong hệ thống, hỗ trợ việc audit, debug và phân tích sau này. Mỗi event thường mang tính nguyên tử, đại diện cho một hành động hoặc thay đổi hoàn chỉnh không thể chia nhỏ thêm. Ngoài ra, trong thực tế, dữ liệu sự kiện được sinh ra với khối lượng lớn và tốc độ cao, đòi hỏi hệ thống xử lý hiệu quả. [18]

Các nguồn lưu trữ thông dụng trả về dữ liệu dạng streaming và event:

Công nghệ	Loại hình	Đặc điểm nổi bật	Độ trễ (Latency)	Nhà cung cấp / Hệ sinh thái
Apache Kafka	Distributed Streaming Platform	Hiệu suất cực cao, khả năng lưu trữ dữ liệu dài hạn, hệ sinh thái Connector đồ sộ.	Rất thấp (Miliseconds)	Open-source, Confluent, Amazon MSK
Amazon Kinesis	Managed Streaming Service	Dễ dàng mở rộng, tích hợp cực tốt với các dịch vụ khác của AWS (Lambda, S3).	Thấp (Miliseconds)	AWS (Amazon Web Services)
Google Cloud Pub/Sub	Messaging Service (Global)	Serverless hoàn toàn, không cần quản lý hạ tầng, hỗ trợ	Thấp	Google Cloud

		phân phối tin nhắn toàn cầu.		Platform (GCP)
Azure Event Hubs	Big Data Streaming	Chuyên dụng cho nạp dữ liệu (ingestion) quy mô lớn, hỗ trợ giao thức Kafka.	Thấp	Microsoft Azure

Bảng 2.2: Một số nguồn lưu trữ thông dụng cho dạng dữ liệu Streaming/Event

2.1.5. Nguồn dữ liệu dạng tệp

Dữ liệu dạng tệp ngoài bảng còn bao gồm hai họ chính là hình ảnh và văn bản (file). Chúng thường xuất hiện trong luồng ingestion/ETL cùng với dữ liệu bảng, nhưng có đặc tính, yêu cầu lưu trữ và tiền xử lý khác biệt. Phần này tập trung vào hai họ tệp phi cấu trúc thường gặp trong hệ thống ingestion/ETL: hình ảnh và tài liệu văn bản (documents: txt, json,…). [16] [19]

Dataset Preview

Subset: adversarial_qa_dbert_answer_the_following_q

Split: train

inputs (sequence)	inputs_pretokenized (string)	targets (sequence)	targets_pretokenized (string)
[9246, 8, 826, 5454, 96, 188, 3381, 865, 6,…	" Given the following passage "A generation later, the…	[24242, 820, 26939, 1]	" subjective idealism "
[9246, 8, 826, 5454, 96, 188, 3381, 865, 6,…	" Given the following passage "A generation later, the…	[44, 88, 159, 11, 1]	" atheism "
[9246, 8, 826, 5454, 96, 188, 3381, 865, 6,…	" Given the following passage "A generation later, the…	[6917, 1]	" humans "
[9246, 8, 826, 5454, 96, 188, 3381, 865, 6,…	" Given the following passage "A generation later, the…	[1683, 1714, 1]	" 1710 "
[9246, 8, 826, 5454, 96, 188, 3381, 865, 6,…	" Given the following passage "A generation later, the…	[901, 3389, 107, 82, 196, 1]	" Alciphron "
[9246, 8, 826, 5454, 96, 13284, 526, 6997,…	" Given the following passage "Hume maintained that all…	[8, 1997, 1]	" the sun "
[9246, 8, 826, 5454, 96, 13284, 526, 6997,…	" Given the following passage "Hume maintained that all…	[1997, 1]	" sun "

Hình 2.3: Dữ liệu ảnh và văn bản thường được dùng trong huấn luyện mô hình TTNT

Dữ liệu dạng tệp là nền tảng cho hầu hết các hoạt động phân tích dữ liệu, báo cáo. Trong đó, những loại dữ liệu phi cấu trúc như hình ảnh và văn bản là thành phần thiết yếu trong các hệ thống học máy hiện đại, đặc biệt trong các bài toán liên quan đến thị giác máy tính (computer vision), xử lý ngôn ngữ tự nhiên (NLP), và các mô hình đa phương thức. Không giống như dữ liệu bảng, các tập tin ảnh và văn bản thường tồn tại dưới dạng blob nhị phân hoặc chuỗi ký tự không có schema cố định, đòi hỏi quy trình ingestion và ETL chuyên biệt để đảm bảo khả năng huấn luyện mô hình hiệu quả, an toàn và có thể mở rộng.

Cụ thể, dữ liệu ảnh và văn bản thường được sử dụng cho các mục đích trong mảng TTNT gồm:

Mục đích	**Dữ liệu ảnh**	**Dữ liệu văn bản**
Huấn luyện mô hình TTNT	Dùng để huấn luyện các mô hình phân loại ảnh, nhận diện đối tượng, phát hiện bất thường.	Dùng để huấn luyện mô hình ngôn ngữ, phân tích cảm xúc, trích xuất thực thể, sinh văn bản tự động
Tiền xử lý và chuẩn hóa dữ liệu đầu vào	Resize ảnh, chuyển định dạng, trích xuất metadata, tạo ra thumbnail	Chuẩn hóa encoding, loại bỏ các ký tự nhiễu, phát hiện ngôn ngữ, trích xuất nội dung từ tập tài liệu PDF, DOCX, OCR
Tạo tập huấn luyện/kiểm thử	Phân loại, gán nhãn, chia tập train/validation/test, kiểm tra lỗi file (corrupt)	Phân loại, gán nhãn, chia tập dữ liệu, kiểm tra encoding, đánh giá chất lượng nội dung

Bảng 2.3: Một số mục đích trong mảng TTNT

Các nguồn lưu trữ thông dụng trả về dữ liệu dạng tệp

Dựa trên các mục đích sử dụn, dữ liệu dạng tệp (hình ảnh, văn bản,…) thường được lưu trữ và phát sinh từ nhiều loại nguồn khác nhau, bao gồm cả hệ thống nội bộ và dịch vụ bên ngoài. Các nguồn này có thể là hệ quản trị cơ sở dữ liệu, các tệp dữ liệu được nhập từ bên ngoài, các dịch vụ web cung cấp dữ liệu theo thời gian thực, hoặc các quy trình xử lý và tổng hợp dữ liệu nội bộ nhằm phục vụ cho việc phân tích, báo cáo hoặc ra quyết định trong hệ thống. [16] [19]

Nguồn	**Mô tả**	**Ví dụ**	**Yêu cầu xử lý chính**
Tệp nén chứa ảnh huấn luyện mô hình	Các tập ảnh được đóng gói thành ZIP/TAR	COCO, ImageNet, Google Drive ZIP	Tải tệp nén,giải nén, kiểm tra định dạng

	để chia sẻ hoặc lưu trữ		ảnh, resize, trích xuất metadata
Tệp nén chứa văn bản thô hoặc JSON	Các tập văn bản hoặc JSON dùng cho NLP, đóng gói để lưu trữ nhiều định dạng	Common Crawl, OpenWebText, HuggingFace datasets	Tải tệp nén, giải nén, phát hiện encoding, chuẩn hóa newline, kiểm tra định dạng JSON
Google Drive / Dropbox / OneDrive	Nền tảng lưu trữ đám mây nơi người dùng chia sẻ tập dữ liệu	Google Drive link chứa ZIP ảnh hoặc JSON	Xử lý liên kết chia sẻ, xác thực token (nếu cần), tải xuống, giải nén, resume/retry
Kho dữ liệu công khai qua HTTP/HTTPS	Các tập dữ liệu được host công khai trên web	Kaggle Datasets, academic URLs	Tải qua wget, kiểm tra checksum, giải nén, xác định loại nội dung (ảnh/văn bản)

Bảng 2.4: Một số nguồn phát sinh dữ liệu dạng tệp

2.2. Phân hệ Data Augmentation

2.2.1. Quá trình làm sạch dữ liệu

Quá trình làm sạch dữ liệu bảng

Quy trình làm sạch dữ liệu dạng bảng bao gồm các tập hợp các bước chuyển đổi, chuẩn hóa và lọc các bản ghi thô với mục đích thu được tập dữ liệu mang tính nhất quán, có cấu trúc và phù hợp cho các bước tiền xử lý, trích xuất đặc trưng, huấn luyện mô hình hoặc phân tích. Mục tiêu không chỉ loại bỏ nhiễu về mặt định dạng và ký tự mà còn bảo toàn ý nghĩa ngữ nghĩa cần thiết cho các công việc phía sau, giảm khối lượng dữ liệu không cần thiết và bảo vệ thông tin nhạy cảm trước khi lưu trữ hoặc chia sẻ. [1] [15]

Dưới đây là bảng tóm tắt các tác vụ làm sạch dữ liệu bảng phổ biến:

Tác vụ	**Mô tả**
Loại bỏ và hợp nhất bản ghi trùng lặp	Phát hiện các bản ghi lặp lại, trùng lặp trong bản ghi làm méo phân bố, gây sai lệch thông tin thống kê và tiêu tốn tài nguyên lưu trữ. Ví dụ: trong bảng khách hàng, có các dòng mã khách hàng và thông tin giống nhau.

Xử lý giá trị thiếu (missing values)	Phát hiện và xử lý các giá trị bị thiếu dữ liệu và áp dụng chiến lược xử lý bằng cách xóa, thay thế hoặc gán giá trị thay thế. Dữ liệu thiếu ảnh hưởng đến mô hình và phân tích. Ví dụ: cột "tuổi" bị thiếu giá trị
Sửa lỗi đơn vị và chuẩn hóa đơn vị đo	Phát hiện các giá trị sử dụng đơn vị khác nhau, quy đổi để về đơn vị chuẩn của dự án. Nếu đơn vị không đồng nhất sẽ dẫn đến kết quả sai khi so sánh hoặc tổng hợp. Ví dụ: Các dữ liệu trong bản ghi ghi bằng "m" và "cm" → quy đổi toàn bộ về hệ số mét trước khi tính trung bình.
Chuẩn hóa định dạng dữ liệu thời gian (datetime)	Các định dạng thời gian khác nhau, cần chuyển dữ liệu về một định dạng thống nhất. Ví dụ: chuyển dữ liệu "DD-MM-YYYY" trong bản ghi thành về 1 dạng "YYYY-MM-DD".

Bảng 2.5: Một số tác vụ làm sạch dữ liệu bảng

Các tác vụ trên cần thực thi theo pipeline có phiên bản, kèm logging tham số và kết quả trên từng bản ghi để đảm bảo tính lặp lại, dễ điều tra và sữa chữa khi phát hiện lỗi downstream. Chọn chiến lược xử lý (loại bỏ, sửa, gắn nhãn, hay đưa review) phải dựa trên chi phí lỗi so với lợi ích, đặc điểm nguồn dữ liệu và yêu cầu nghiệp vụ.

Quá trình làm sạch dữ liệu tuần tự

Một số tác vụ phổ biến trong quá trình làm sạch dữ liệu dạng tuần tự bao gồm: [16] [19].

Tác vụ	Mô tả
Kiểm tra và chuẩn hóa cấu trúc dữ liệu	Xác định dữ liệu có tuân thủ schema (trường, kiểu dữ liệu, ràng buộc) hay không; ví dụ: phát hiện cột "tuổi" có kiểu string thay vì int và tự động ép kiểu hoặc loại bỏ bản ghi lỗi
Xử lý giá trị thiếu (MissingValues Handling)	Phát hiện và xử lý giá trị null/trống bằng cách xóa, điền hoặc suy luận; ví dụ: điền giá trị trung bình hoặc loại bỏ bản ghi thiếu thông tin quan trọng
Loại bỏ dữ liệu trùng lặp (Duplicate Removal)	Xác định và xử lý các bản ghi trùng dựa trên khóa; ví dụ: xóa các bản ghi có cùng ID và timestamp hoặc hợp nhất theo quy tắc

Chuẩn hóa định dạng (Standardization/ Normalization)	Đưa dữ liệu về một định dạng thống nhất; ví dụ: chuẩn hóa ngày giờ, chữ hoa/chữ thường hoặc mã code từ nhiều nguồn khác nhau
Xử lý dữ liệu ngoại lệ (Outlier Detection)	Phát hiện và xử lý các giá trị bất thường; ví dụ: loại bỏ giá trị nhiệt độ vượt ngưỡng vật lý hoặc đánh dấu để kiểm tra
Kiểm tra theo quy tắc nghiệp vụ (Business Rule Validation)	Kiểm tra dữ liệu theo các luật logic cụ thể; ví dụ: giá trị phải nằm trong khoảng cho phép, hoặc tổng tiền phải ≥ 0
Chuyển đổi dữ liệu (Transformation)	Thực hiện các phép biến đổi để phù hợp với xử lý phía sau; ví dụ: chuyển đổi đơn vị (kg → g), tách/gộp cột hoặc mã hóa dữ liệu

Bảng 2.6: Một số tác vụ làm sạch dữ liệu tuần tự

Quá trình làm sạch dữ liệu sự kiện

Dữ liệu dạng sự kiện sẽ thường chứa nhiều vấn đề về chất lượng do được thu thập từ nhiều nguồn khác nhau, theo thời gian thực và với quy mô lớn. Việc làm sạch dữ liệu trở thành bước quan trọng để đảm bảo tính chính xác và độ tin cậy của các phân tích sau này. [17] [18]

Một số tác vụ phổ biến trong quá trình làm sạch dữ liệu sự kiện bao gồm:

Tác vụ	Mô tả
Xử lý giá trị thiếu (Missing Value Detection and Handling)	Phát hiện và loại bỏ các thuộc tính bị thiếu do lỗi thu thập hoặc truyền tải; ví dụ: loại bỏ event thiếu thông tin quan trọng, hoặc điền giá trị trung bình (số), giá trị phổ biến (phân loại), hoặc dùng KNN/hồi quy để suy đoán
Phát hiện và loại bỏ trùng lặp (Duplicate Detection and Removal)	Xác định các tiêu chí nhận dạng sự kiện bị ghi nhận nhiều lần; ví dụ: hai event có cùng user_id và timestamp được coi là trùng → giữ 1 bản ghi hoặc hợp nhất
Chuẩn hóa dữ liệu (Data Normalization)	Là quá trình chuyển đổi dữ liệu về định dạng thống nhất về tên thuộc tính và giá trị; ví dụ: chuẩn hóa timestamp về cùng múi giờ, chuyển đơn vị đo (ms→s), chuẩn hóa text (lowercase, bỏ khoảng trắng), thống nhất tên field

Phát hiện và xử lý ngoại lệ (Outlier Detection and Handling)	Phát hiện giá trị bất thường, xác định ngoại lệ là lỗi cần được loại bỏ hay sửa chữa; ví dụ: số lượng giao dịch tăng đột biến → kiểm tra bằng Z-score, IQR hoặc Isolation Forest và quyết định giữ hay xóa.
Xác thực tính nhất quán (Consistency Validation)	Đảm bảo dữ liệu không mâu thuẫn và đúng logic; ví dụ: event "logout" không thể xảy ra trước "login", hoặc trạng thái đơn hàng phải theo đúng thứ tự
Lọc và chọn lọc dữ liệu (Filtering and Selection)	Loại bỏ dữ liệu không cần thiết trong khoảng thời gian quan tâm; ví dụ: chỉ lấy event trong 30 ngày gần nhất, hoặc chỉ giữ các event "purchase" phục vụ phân tích doanh thu

Bảng 2.7: Một số tác vụ làm sạch dữ liệu sự kiện

Quá trình làm sạch dữ liệu dạng tệp:

Một số tác vụ trong quy trình làm sạch dữ liệu hình ảnh

Quy trình làm sạch dữ liệu hình ảnh là bước then chốt đảm bảo rằng dữ liệu ảnh đầu vào cho các pipeline huấn luyện và suy luận đạt kết quả chất lượng cao, đồng nhất và an toàn. Trước hết, quy trình làm giảm thiểu, loại bỏ các lỗi kỹ thuật như file bị hỏng, sai định dạng hoặc bị trùng lặp nhằm tránh gây nhiễu trong quá trình huấn luyện. Tiếp theo, các hình ảnh có chất lượng kém (ví dụ: bị mờ, quá sáng/tối hoặc nhiễu cao) sẽ được phát hiện và loại bỏ hoặc gắn nhãn để không làm ảnh hưởng đến độ chính xác của mô hình. Cuối cùng, quy trình còn đóng vai trò bảo vệ quyền riêng tư bằng cách nhận diện dữ liệu nhạy cảm và áp dụng các biện pháp như cách ly hoặc che giấu trước khi lưu trữ hoặc chia sẻ. Kết quả là một tập dữ liệu đã được chuẩn hoá về kích thước và định dạng, đi kèm metadata đầy đủ (như checksum, MIME type, nguồn gốc, điểm chất lượng), sẵn sàng cho các bước tiếp theo như gán nhãn, tăng cường dữ liệu, tạo embedding và huấn luyện. [16] [19]

Bên cạnh đó, quy trình làm sạch còn hướng đến hiệu quả vận hành thông qua việc tối ưu tài nguyên bằng xử lý streaming và bất đồng bộ đối với các file lớn, cung cấp cơ chế checkpoint và logging để dễ dàng truy vết và tái tạo, cũng như tích hợp các "cổng kiểm soát chất lượng" (quality gates) nhằm tự động loại bỏ hoặc chuyển các mẫu không đạt yêu cầu sang bước đánh giá thủ công. Từ góc độ trí tuệ nhân tạo, việc làm sạch dữ liệu giúp cải thiện tốc độ hội tụ của mô hình, hạn chế hiện tượng overfitting do nhiễu và tăng khả năng khái quát hoá.

Trong khi đó, dữ liệu ảnh thường được lưu trữ theo thư mục với khối lượng lớn và đến từ nhiều nguồn khác nhau, vì vậy một số tác vụ quan trọng trong quá trình làm sạch có thể kể tới bao gồm:

- **Phát hiện trùng lặp**: Loại bỏ các bản sao chính xác hoặc gần giống trong kho ảnh làm tập huấn luyện để tránh làm lệch phân phối dữ liệu và gây lặp mẫu trong quá trình huấn luyện.
- **Đánh giá chất lượng ảnh**: Vì tồn tại các ảnh tiêu chuẩn không phù hợp làm khả năng khái quát của mô hình nếu vẫn được đưa vào huấn luyện. Do đó, cần xây dựng các phương thức đánh giá và đo lường định lượng ảnh để xác định ảnh không đạt chuẩn, từ đó loại bỏ hoặc đánh dấu.

Một số tác vụ trong quy trình làm sạch dữ liệu văn bản

Đối với dữ liệu văn bản, quy trình làm sạch bao gồm các bước chuyển đổi, chuẩn hoá và lọc nhằm biến dữ liệu thô thành tập văn bản có cấu trúc, nhất quán và phù hợp cho các tác vụ NLP như tiền xử lý, tạo embedding, huấn luyện hoặc phân tích. Mục tiêu không chỉ là loại bỏ nhiễu về ký tự hay định dạng mà còn đảm bảo giữ được ý nghĩa ngữ nghĩa, giảm dung lượng dư thừa và bảo vệ thông tin nhạy cảm trước khi lưu trữ hoặc chia sẻ. Một hệ thống hiệu quả cần hỗ trợ xử lý streaming, ghi nhận nguồn gốc dữ liệu (provenance) và tích hợp các quality gates để tự động sàng lọc. [16] [19]

Trong thực tế, dữ liệu văn bản thường đến từ nhiều nguồn như file TXT/JSON, API, OCR hoặc thu thập từ web, dẫn đến các vấn đề phổ biến như trùng lặp nội dung, dư thừa thẻ HTML/markup, rò rỉ thông tin như URL/email/số điện thoại, ký tự đặc biệt hoặc khoảng trắng không cần thiết, không nhất quán về encoding hoặc chữ hoa/chữ thường, cũng như các văn bản quá ngắn hoặc quá dài không phù hợp cho việc xử lý.

Dưới đây là các tác vụ làm sạch thường có trong các quy trình huấn luyện mô hình TTNT:

Tác vụ	**Yêu cầu**
Lọc trùng lặp	Nguồn dữ liệu thường chứa nhiều bản sao giống hoặc gần giống bản gốc, dẫn đến làm méo phân phối, lãng phí tài nguyên huấn luyện. Do đó các bản trùng cần bị loại khỏi luồng chính hoặc được gắn flag với tham chiếu đến bản gốc
Loại bỏ thẻ HTML/markup	Văn bản thu từ web thường kèm thẻ HTML, script hoặc comment gây nhiễu. Có thể loại bỏ các thẻ html đó để

	văn bản thuần (plain text) không còn tag; giữ lại nội dung có ý nghĩa.
Loại bỏ URL, email, số điện thoại	URL/email/phone có thể chứa các thông tin nhạy cảm, gây nhiễu khi mô hình sinh nội dung. Có thể xử lý chuỗi sau khi xóa hoặc thay thế bằng token chuẩn.
Loại bỏ ký tự đặc biệt và khoảng trắng thừa	Các ký tự control, biểu tượng lạ, hoặc nhiều khoảng trắng sẽ làm ảnh hưởng tokenizer và tính nhất quán. Chuỗi chỉ nên chứa ký tự cho phép (letters, digits, punctuation theo policy) và spacing chuẩn hoá (single space, trim); metadata có thể ghi số ký tự đã thay đổi.
Chuyển về chữ thường	Sự khác biệt chữ hoa/chữ thường ảnh hưởng matching/token counts; đôi khi case giữ thông tin (ví dụ tên riêng) nên phải cấu hình
Kiểm tra độ dài	Văn bản quá ngắn thiếu ngữ cảnh; quá dài gây quá tải cho tokenizer hoặc không phù hợp với giới hạn model.

Bảng 2.8: Một số tác vụ làm sạch dữ liệu văn bản

2.2.2. Quá trình chuyển đổi dữ liệu

Chuyển đổi dữ liệu là tập hợp các bước xử lý nhằm biến dữ liệu thô thành dạng có cấu trúc, nhất quán và sẵn sàng cho các giai đoạn tiếp theo trong vòng đời dữ liệu, như xây dựng đặc trưng (feature engineering), huấn luyện mô hình hoặc phân tích. Các thao tác phổ biến trong quá trình này bao gồm chuẩn hóa kiểu dữ liệu và đơn vị đo, làm sạch dữ liệu nhiễu hoặc bị lỗi, chuẩn hóa định dạng (ví dụ: ngày tháng, mã hóa ký tự), phân đoạn văn bản, tách từ (tokenization), mã hóa các biến phân loại và trích xuất các đặc trưng cơ bản.

Mục tiêu của chuyển đổi dữ liệu không chỉ dừng lại ở việc xử lý lỗi mà còn đảm bảo tính xác định (deterministic), khả năng tái lập (reproducibility) và lưu vết nguồn gốc dữ liệu (provenance), từ đó hỗ trợ hiệu quả cho việc kiểm toán và gỡ lỗi khi cần thiết.

2.2.3. Quá trình gán nhãn dữ liệu

Gắn nhãn dữ liệu (data labeling) là tập hợp các hoạt động nhằm gán thông tin mục tiêu — như nhãn, thẻ hoặc chú thích — lên các bản ghi để phục vụ huấn luyện, đánh giá và giám sát mô hình học máy. Quá trình này bao gồm cả thao tác thủ công do con người thực hiện (human annotation), các quy tắc tự động hóa (rule-based labeling) và nhãn yếu hoặc nhãn sinh bởi mô hình (weak supervision / model-based labeling). Một hệ thống gắn nhãn chất lượng không chỉ tạo ra nhãn chính xác mà còn

đảm bảo ghi lại provenance (nguồn nhãn, thời điểm, confidence), duy trì sự nhất quán giữa các annotator và cho phép kiểm tra, sửa lỗi hoặc rollback khi cần.

2.2.4. Dữ liệu đặc trưng (feature data)

Dữ liệu đặc trưng (feature data) là các biểu diễn trung gian đã được trích xuất hoặc biến đổi từ dữ liệu thô, tối ưu cho việc huấn luyện, inference hoặc phục vụ hệ thống production. Một bản ghi đặc trưng thường là vector số (embedding), các cột tổng hợp (aggregates, rolling stats), các chỉ số phân loại đã mã hóa (one-hot, categorical id), hoặc các descriptors (HOG, color histogram). Mục tiêu chính của feature data là cung cấp đầu vào có kích thước và định dạng ổn định cho mô hình hoặc pipeline serving, tối ưu cho throughput, truy vấn nhanh và tái dùng giữa nhiều mô hình.

Dữ liệu đặc trưng khác với dữ liệu tinh chỉnh ở chỗ nó đã mất (hoặc giảm) một phần thông tin thô để đổi lấy tính hiệu quả và tính thống nhất. Do đó feature store/feature dataset cần lưu trữ không chỉ giá trị feature mà còn metadata mô tả cách sinh feature (feature_code_version, scaler params, window_size, encoder vocabulary) để đảm bảo reproducibility và khả năng tái tạo khi retrain hoặc khi phát hiện drift.

Dữ liệu tinh chỉnh thường giữ nhiều đặc điểm của nguồn gốc: kích thước ảnh gốc hoặc crop hợp lệ, nguyên câu văn bản (có hoặc không tokenized tùy yêu cầu mô hình), hoặc bản ghi bảng có các cột gốc chưa bị mất thông tin quan trọng. Về storage và định dạng, dữ liệu tinh chỉnh thường lưu ở dạng thích hợp cho huấn luyện: files và manifests (image files + COCO/PASCAL annotations, JSONL cho text prompt/response, Parquet/CSV cho bảng kèm label column. Việc lưu giữ bản gốc (raw) cùng bản đã chuẩn hóa giúp debug và cho phép tái tạo lại các bước xử lý nếu cần rollback.

Một số tác vụ lưu dữ liệu đặc trưng

Dưới đây mô tả các tác vụ vận hành chính liên quan đến feature data:

Tác vụ	Mô tả
Chuẩn hóa và lưu feature theo định dạng tối ưu	Lưu trữ dữ liệu đặc trưng theo định dạng tối ưu phù hợp cho truy vấn và xử lý hiệu quả như (Parquet, Feather, HDF5, binary shards). Quá trình thực hiện bằng cách serialize id + vector hoặc cột đặc trưng, chọn compression phù hợp và áp dụng nén để tối ưu và kích thước lưu trữ.

Xây dựng chỉ mục và tích hợp kho vector	Thiết lập chỉ mục cho embeddings bằng các cách (FAISS, Annoy, HNSW) hoặc sử dụng vào cơ sở dữ liệu vector để hỗ trợ truy vấn nearest-neighbor lookup nhanh, đồng thời lưu mapping id↔origin_id và duy trì chỉ mục theo chính sách cập nhật phù hợp.
Phân vùng, chia shard và tối ưu hoá truy cập	Partition và shard feature dataset theo key (time/label/feature_group) để tối ưu throughput đọc cho training hoặc serving, đồng thời tạo manifest và policy replication để đọc song song hiệu quả.
Kiểm tra tính nhất quán và xác thực pipeline feature	Kiểm tra tính hợp lệ của feature (no NaN/Inf, trong range, distribution checks) và phát cảnh báo khi drift; thực hiện test tự động để bảo đảm feature phù hợp schema kỳ vọng.
Lưu trữ các aggregate dẫn xuất và feature	Lưu trữ các feature tổng hợp (rolling counts, last_seen, aggregates) trong key-value store hoặc online feature store với TTL và idempotent update để phục vụ low-latency serving.

Bảng 2.9: Tổng quan một số tác vụ trong quy trình lưu trữ dữ liệu đặc trưng

Nhìn chung, lưu trữ dữ liệu đặc trưng là lớp hạ tầng then chốt để đảm bảo tính hiệu quả, nhất quán và khả năng mở rộng cho pipeline ML. Thiết kế feature store/feature artifact phải cân bằng giữa tiết kiệm lưu trữ (compression/quantization), tốc độ truy cập (partitioning/indexing) và tính reproducibility (versioning/lineage). Việc chuẩn hoá metadata, kiểm tra tính nhất quán và cung cấp cơ chế rollback/versioned rollout sẽ giảm rủi ro training-serving skew và giúp vận hành mô hình tin cậy trong môi trường production.

2.3. Kết luận chương

Trong chương này, luận văn đã trình bày tổng quan về các nền tảng mã nguồn mở phục vụ cho quá trình phát triển mô hình học máy, tập trung vào hai phân hệ cốt lõi là Data Ingestion và Data Augmentation. Thông qua việc phân tích các nguồn dữ liệu đầu vào đa dạng như dữ liệu dạng bảng, dữ liệu tuần tự, streaming, dữ liệu sự kiện và dữ liệu dạng tệp, chương đã làm rõ vai trò quan trọng của việc thu thập và tổ chức dữ liệu trong toàn bộ vòng đời mô hình.

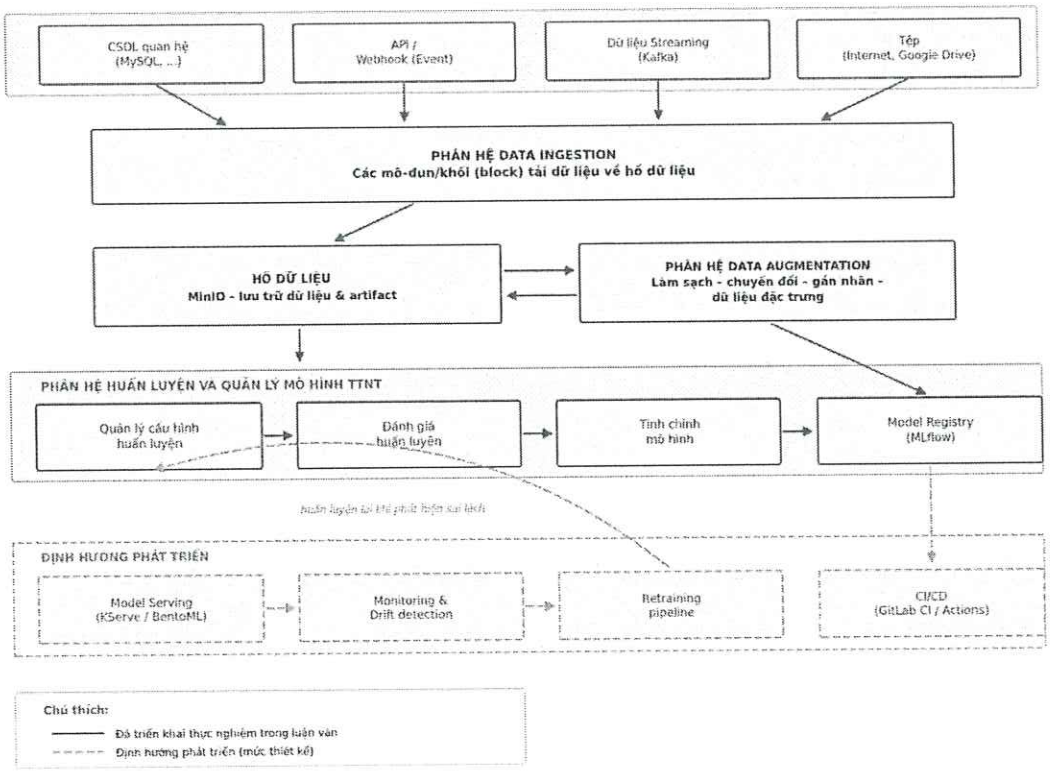
Bên cạnh đó, các quy trình xử lý dữ liệu như làm sạch, chuyển đổi và gán nhãn cũng đã được trình bày chi tiết, nhằm đảm bảo dữ liệu đạt chất lượng cao, nhất quán và sẵn sàng cho các bước tiếp theo như huấn luyện và đánh giá mô hình.

CHƯƠNG 3: THIẾT KẾ VÀ XÂY DỰNG HỆ THỐNG MLOPS

Chương 3 sẽ trình bày quá trình thiết kế và xây dựng hệ thống MLOPS trên cơ sở tích hợp các nền tảng mã nguồn mở đã được phân tích ở Chương 2. Nội dung chương gồm: kiến trúc tổng thể của hệ thống và mối quan hệ giữa các thành phần

3.1. Kiến trúc tổng thể hệ thống MLOPS

Hệ thống MLOPS trong luận văn được thiết kế theo kiến trúc mô-đun (modular architecture), trong đó mỗi phân hệ đảm nhận một giai đoạn trong vòng đời phát triển mô hình học máy và trao đổi dữ liệu với nhau thông qua hồ dữ liệu tập trung. Cách tổ chức này cho phép các phân hệ được phát triển, thay thế và mở rộng độc lập, phù hợp với đặc điểm của việc tích hợp nhiều nền tảng mã nguồn mở khác nhau



Hình 3.1: Kiến trúc tổng thể hệ thống MLOPS

Hệ thống gồm các thành phần chính sau:

– **Phân hệ Data Ingestion:** tiếp nhận dữ liệu từ nhiều loại nguồn khác nhau, gồm cơ sở dữ liệu quan hệ (MySQL), dữ liệu sự kiện qua API/Webhook, dữ liệu streaming qua Apache Kafka và dữ liệu dạng tệp từ Internet hoặc Google Drive. Mỗi loại nguồn được xử lý bởi một mô-đun tải dữ liệu riêng; trong backend, mỗi mô-đun được cài đặt dưới dạng một khối (block) kế thừa từ các

lớp cơ sở chung, bảo đảm tính thống nhất về giao diện và khả năng tái sử dụng.

- **Hồ dữ liệu xây dựng trên nền tảng MinIO:** đóng vai trò kho lưu trữ tập trung cho dữ liệu thô, dữ liệu đã xử lý và các artifact sinh ra trong quá trình huấn luyện. Đây là điểm kết nối trung tâm giữa các phân hệ: dữ liệu do phân hệ Data Ingestion thu thập được ghi vào hồ dữ liệu, sau đó được các phân hệ phía sau đọc ra để xử lý và huấn luyện.
- **Phân hệ Data Augmentation:** thực hiện làm sạch, chuyển đổi, gán nhãn dữ liệu và tạo dữ liệu đặc trưng theo các quy trình đã phân tích, nhằm nâng cao chất lượng dữ liệu trước khi đưa vào huấn luyện.
- **Phân hệ huấn luyện và quản lý mô hình TTNT:** gồm chuỗi các mô-đun quản lý cấu hình huấn luyện, đánh giá huấn luyện, tinh chỉnh mô hình và quản lý danh mục mô hình thông qua MLflow Model Registry. MLflow phối hợp với MinIO theo cơ chế: siêu dữ liệu (tham số, chỉ số đánh giá, phiên bản) được MLflow quản lý, còn artifact của mô hình được lưu trên MinIO.

Về mối quan hệ giữa các thành phần, luồng dữ liệu trong hệ thống đi theo một chiều thống nhất: từ các nguồn dữ liệu, qua phân hệ Data Ingestion vào hồ dữ liệu; từ hồ dữ liệu, dữ liệu được phân hệ Data Augmentation xử lý rồi cung cấp cho phân hệ huấn luyện; kết quả huấn luyện (mô hình, chỉ số, checkpoint) được đăng ký vào Model Registry và lưu trữ trên MinIO. Nhờ lấy hồ dữ liệu làm trung tâm, các phân hệ không phụ thuộc trực tiếp vào nhau mà chỉ phụ thuộc vào định dạng dữ liệu trao đổi, giúp hệ thống dễ mở rộng.

Để bảo đảm tính nhất quán về thuật ngữ, "mô-đun" được dùng để chỉ một đơn vị chức năng của hệ thống ở mức thiết kế và giao diện người dùng; "khối" được dùng để chỉ lớp cài đặt tương ứng của mô-đun trong mã nguồn backend, trong khi "hồ dữ liệu" được dùng thống nhất để chỉ kho lưu trữ dữ liệu tập trung xây dựng trên nền tảng MinIO.

3.2. Kiến trúc và cài đặt các mô-đun phục vụ quản lý nguồn dữ liệu

3.2.1. Một số mô-đun thực thi lệnh tải dữ liệu dạng bảng về hồ dữ liệu

Dưới đây là mẫu cấu trúc mô tả chi tiết cho từng mô-đun tiếp nhận dữ liệu dạng bảng trong hệ thống:

Mô-đun kéo dữ liệu từ hệ quản trị cơ sở dữ liệu MySQL

Mô-đun thực hiện kết nối và truy vấn dữ liệu từ hệ quản trị cơ sở dữ liệu MySQL.

Hình 3.2: Giao diện cấu hình mô-đun kéo dữ liệu từ một hệ quản trị cơ sở dữ liệu

Cài đặt trong backend: Trong backend của hệ thống, mô-đun này được triển khai dưới dạng một lớp Python có tên `MySQLSourceBlock`, kế thừa từ hai lớp cha `TaskGeneralBlock` và `MinioConnectionBlock`. Việc kế thừa này cho phép khối vừa đảm nhận vai trò một tác vụ trong pipeline ingest, vừa có khả năng tương tác với hệ thống lưu trữ MinIO để ghi dữ liệu đầu ra.

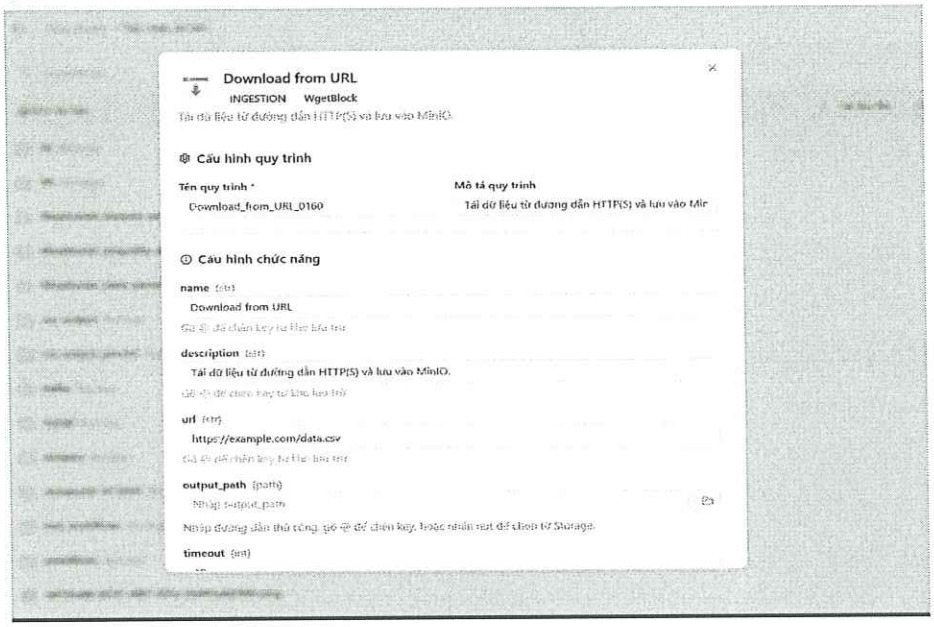
Mục tiêu chính của khối là kết nối tới cơ sở dữ liệu MySQL, thực thi truy vấn SQL do người dùng cấu hình, và lưu kết quả dưới dạng bảng vào kho lưu trữ. Do đó cấu trúc lớp được tổ chức rõ ràng với các thuộc tính metadata như `category`, `description`, và `img_url` để phục vụ giao diện người dùng và phân loại chức năng. Các phương thức chính bao gồm:

- `from_config(cls, config: Dict[str, Any])`: là một class method kế thừa từ TaskGeneratlBlock dùng để khởi tạo đối tượng từ một cấu hình đầu vào dạng dictionary. Đây là điểm tiếp nhận cấu hình từ giao diện người dùng hoặc từ file pipeline.
- `show_config(self)`: phương thức chính thực kế thừa từ TaskGeneratlBlock, trả về cấu hình hiện tại của khối dưới dạng dictionary, phục vụ cho việc hiển thị lại thông tin hoặc ghi log.
- `create_connection(self)`: thực hiện kết nối tới cơ sở dữ liệu MySQL bằng cách sử dụng thư viện `mysql.connector`. Hàm này xử lý thông tin xác thực và kiểm tra kết nối.
- `_resolve_username(self)`: là hàm nội bộ dùng để xác định tên người dùng kết nối, có thể được lấy từ cấu hình hoặc từ hệ thống quản lý bí mật.
- `execute(self, data: Any = None)`: là phương thức chính thực kế thừa từ TaskGeneratlBlock thi truy vấn SQL đã cấu hình, trả về kết quả dưới dạng

`pandas.DataFrame`. Kết quả này sau đó có thể được ghi vào MinIO hoặc truyền sang khối tiếp theo trong pipeline.

Mô-đun thu nhận file bảng từ URL (HTTP/FTP)

Khối thực hiện tải dữ liệu bảng từ địa chỉ URL (HTTP/HTTPS) và lưu trữ vào hệ thống tạm hoặc kho lưu trữ như MinIO.



Hình 3.3: Giao diện cấu hình mô-đun kéo một tệp dạng bảng từ Internet

Cài đặt trong backend: Trong backend, tương tự khối MySQL, khối này được định nghĩa dưới dạng lớp, kế thừa từ hai lớp cha `TaskGeneralBlock` và `MinioConnectionBlock`. Mục tiêu chính của khối là thực hiện tải file từ một địa chỉ URL (file CSV, Excel,...), lưu nội dung vào MinIO, và trả về thông tin metadata dưới dạng bảng để sử dụng trong các bước xử lý tiếp theo.

Cấu trúc lớp được tổ chức rõ ràng với các thuộc tính mô tả như `description` và `icon_url`, phục vụ cho việc hiển thị trong giao diện người dùng. Phương thức `task_generator()` xác định loại tác vụ là `INGESTION`, giúp hệ thống phân loại khối trong pipeline. Việc khởi tạo khối từ cấu hình được thực hiện qua hàm `from_config(config)`, trong khi `show_config()` trả về cấu hình hiện tại để phục vụ hiển thị hoặc ghi log.

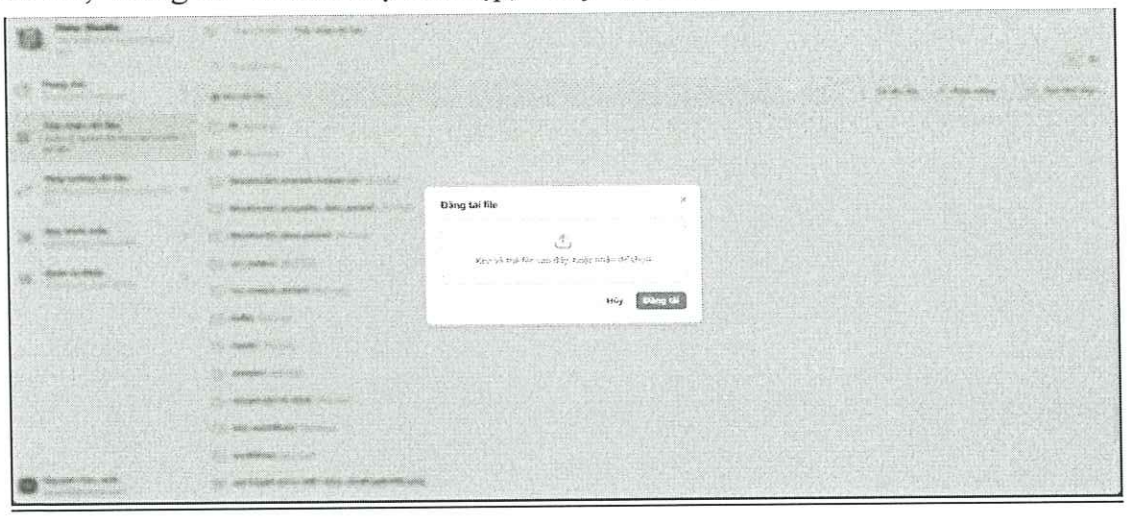
Các hàm xử lý chính bao gồm `_download(url, content_type)`, thực hiện HTTP GET tới địa chỉ URL và trả về nội dung file dưới dạng `requests.Response`. Hàm `_download_url(url)` trích xuất nội dung file dưới dạng `bytes`, phục vụ cho việc ghi vào stream. Trong phương thức `execute(data)`, hệ thống xác định tên file (từ URL hoặc cấu hình), tải nội dung về dưới dạng stream, và gọi `_upload(payload)` để lưu vào MinIO. Sau đó, hàm `_record(...)` ghi lại metadata gồm URL, tên file,

người dùng, vị trí lưu trữ và loại nội dung. Kết quả cuối cùng được trả về dưới dạng một `DataFrame` chứa một dòng mô tả file đã tải, giúp pipeline có thể tiếp tục xử lý hoặc kiểm tra dữ liệu.

Mô-đun tải tệp dữ liệu dạng bảng lên trực tiếp lên kho lưu trữ của người dùng

Khối cho phép người dùng tải tệp dữ liệu bảng (CSV, Excel, Parquet…) từ máy cá nhân lên hệ thống lưu trữ nội bộ hoặc MinIO, phục vụ cho các bước xử lý tiếp theo trong pipeline.

Giao diện cấu hình: Giao diện khối được thiết kế đơn giản và thân thiện, bao gồm: Khu vực kéo-thả hoặc chọn tệp; Hiển thị tên tệp, dung lượng, định dạng sau khi tải lên; "Đăng tải" để xác nhận lưu tệp, "Hủy" để thoát.



Hình 3.4: Mô-đun tải tệp lên kho lưu trữ của người dùng

Khác với hai mô-đun trên, mô-đun này tương tác trực tiếp với hệ lưu trữ mà không cần thông qua bước trung gian như truy vấn cơ sở dữ liệu hay tải dữ liệu từ Internet. Khi người dùng thực hiện thao tác kéo-thả hoặc chọn tệp để đăng tải, hệ thống sẽ tiếp nhận nội dung tệp dưới dạng stream và ghi trực tiếp vào kho lưu trữ theo đường dẫn được cấu hình hoặc sinh tự động. Việc ghi dữ liệu diễn ra ngay trong quá trình thực thi khối, không phụ thuộc vào nguồn bên ngoài hay kết nối mạng, giúp đảm bảo độ ổn định và tốc độ xử lý cao.

Điểm đặc biệt của mô-đun này là khả năng ghi nhận đầy đủ metadata liên quan đến tệp: tên gốc, định dạng, người tải lên, thời gian, kích thước và vị trí lưu trữ. Những thông tin này được lưu dưới dạng bảng và có thể sử dụng ngay trong các bước xử lý tiếp theo như phân tích định dạng, kiểm tra chất lượng dữ liệu, hoặc ánh xạ schema. Cách tiếp cận này giúp đơn giản hóa quy trình ingest dữ liệu thủ công, đồng thời tạo điều kiện thuận lợi để tích hợp các khối xử lý tự động phía sau. Trong các phiên bản mở rộng, mô-đun có thể hỗ trợ upload nhiều tệp cùng lúc, gắn nhãn dữ liệu, hoặc tích hợp với các dịch vụ lưu trữ đám mây để phục vụ các trường hợp ingest phức tạp hơn.

Ví dụ: Kết nối truy vấn dữ liệu từ MySQL

Thử nghiệm khối MySQL để kết nối tới cơ sở dữ liệu quan hệ và truy xuất bảng dữ liệu người dùng.

Cấu hình thực hiện:

- Host: `127.0.0.1`
- Database: `vds`
- User: `root`
- Query: `SELECT * FROM user_flows LIMIT 100`
- Output path: mysql/users.csv`

Hình 3.5: Cấu hình thực hiện kết nối

Luồng thực hiện:

1. Người dùng kéo khối MySQL Source vào giao diện và cấu hình thông tin kết nối.
2. Hệ thống thực hiện kết nối và chạy truy vấn SQL.
3. Kết quả được trả về dưới dạng bảng, lưu vào MinIO và hiển thị preview.

Kết quả:

- Trạng thái: Thành công
- Số bản ghi: 100
- Thời gian thực thi: ~2.3 giây
- Log: Không có lỗi kết nối hoặc truy vấn

Hình 3.6: Kết quả thực hiện

Mô-đun quản lý hồ sơ của nguồn dữ liệu vào dạng bảng

Trong hệ thống, mỗi luồng dữ liệu sau khi được thực thi đều được ghi nhận và lưu trữ đầy đủ thông tin vào cơ sở dữ liệu quản lý tác vụ. Việc này không chỉ phục vụ mục tiêu giám sát và truy vết, mà còn giúp người dùng dễ dàng kiểm tra lịch sử hoạt động, phân tích lỗi, và tối ưu hóa quy trình ingest. Mỗi bản ghi luồng bao gồm các thông tin như: tên luồng, số lần chạy, thời điểm chạy gần nhất, trạng thái thực thi (thành công/thất bại), và các thông số cấu hình tại thời điểm đó.

Hình 3.7: Giao diện mô-đun quản lý hồ sơ của nguồn dữ liệu vào dạng bảng

Hệ thống cho phép người dùng tra cứu các luồng đã chạy thông qua giao diện bảng điều khiển, với các bộ lọc theo trạng thái, thời gian, hoặc tên luồng. Các luồng thất bại được đánh dấu rõ ràng, kèm theo log chi tiết để hỗ trợ phân tích nguyên nhân. Với các luồng thành công, người dùng có thể xem lại kết quả đầu ra, cấu hình đã sử dụng, và thậm chí sao chép để tạo luồng mới tương tự. Cơ chế lưu trữ này giúp đảm

bảo tính lặp lại (reproducibility) và minh bạch trong toàn bộ quá trình ingest và xử lý dữ liệu.

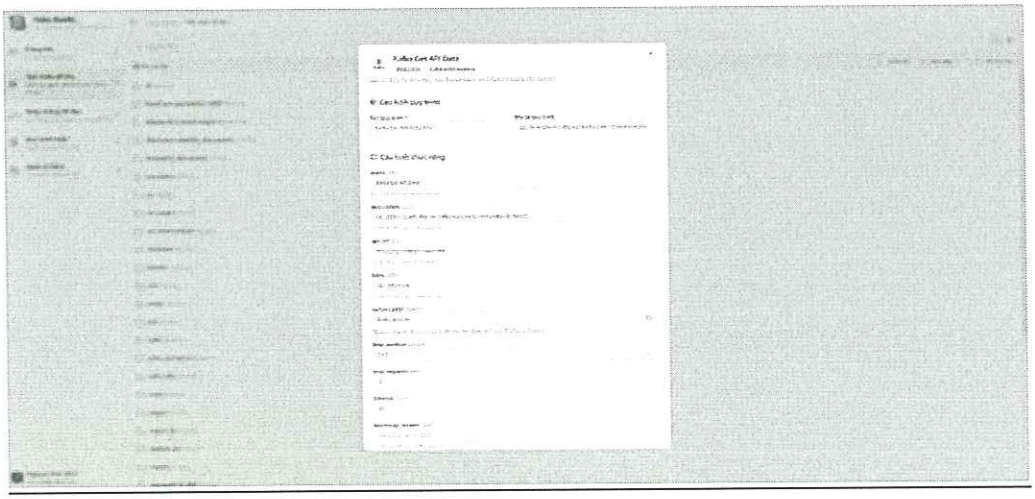
3.2.2. Một số mô-đun thực thi lệnh tải dữ liệu dạng tuần tự về hồ dữ liệu

Mô-đun cho phép người dùng sử dụng Apache Kafka để thu thập dữ liệu tuần tự từ API dưới dạng theo lô hoặc trực tiếp (Stream/Batch)

Mô-đun này trình bày chi tiết về một giải pháp chuyên dụng sử dụng Apache Kafka làm nền tảng trung tâm để thu thập dữ liệu tuần tự từ các nguồn API. Nó cung cấp khả năng cấu hình linh hoạt, cho phép người dùng lựa chọn giữa hai phương thức thu thập chính:

- Stream (Trực tiếp): Thu thập và xử lý dữ liệu theo thời gian thực, ngay khi dữ liệu được tạo ra từ API.
- Batch (Theo lô): Thu thập dữ liệu theo các khoảng thời gian định kỳ hoặc khi đạt đến một khối lượng nhất định.

Giải pháp này đảm bảo việc trích xuất dữ liệu từ API được thực hiện một cách hiệu quả, đáng tin cậy và có khả năng mở rộng.



Hình 3.8: Giao diện cấu hình mô-đun kéo dữ liệu về hồ dữ liệu sử dụng Kafka

Cài đặt trong backend: Trong backend của hệ thống, mô-đun này được triển khai dưới dạng một lớp Python có tên 'KafkaGetAPIData', kế thừa từ hai lớp cha 'TaskGeneralBlock' và 'MinioConnectionBlock'. Việc kế thừa này cho phép khối vừa đảm nhận được vai trò một tác vụ trong ETL/ELT pipeline, vừa cung cấp khả năng tương tác với hệ thống lưu trữ đã được cài đặt cùng với hệ thống là MinIO để thực hiện việc ghi dữ liệu đầu ra.

Lớp KafkaGetAPIDataBlock được thiết kế như một khối chức năng (task block) chuyên biệt cho việc **thu thập** dữ liệu. Mục tiêu cốt lõi của nó là bắc một cây cầu tự động giữa một nguồn dữ liệu API bên ngoài và một hệ thống lưu trữ/phân tích dữ liệu thời gian thực (Apache Kafka).

Nói một cách đơn giản, khối này thực hiện ba nhiệm vụ chính:

1. **Gọi API:** Tự động gửi yêu cầu (request) đến một URL API được chỉ định để lấy dữ liệu.
2. **Đẩy vào Kafka:** Lấy dữ liệu nhận được và đẩy (produce) vào một chủ đề (topic) Kafka cụ thể.
3. **Báo cáo:** Trả về một bản tóm tắt về kết quả công việc (ví dụ: đã đẩy bao nhiêu bản ghi).

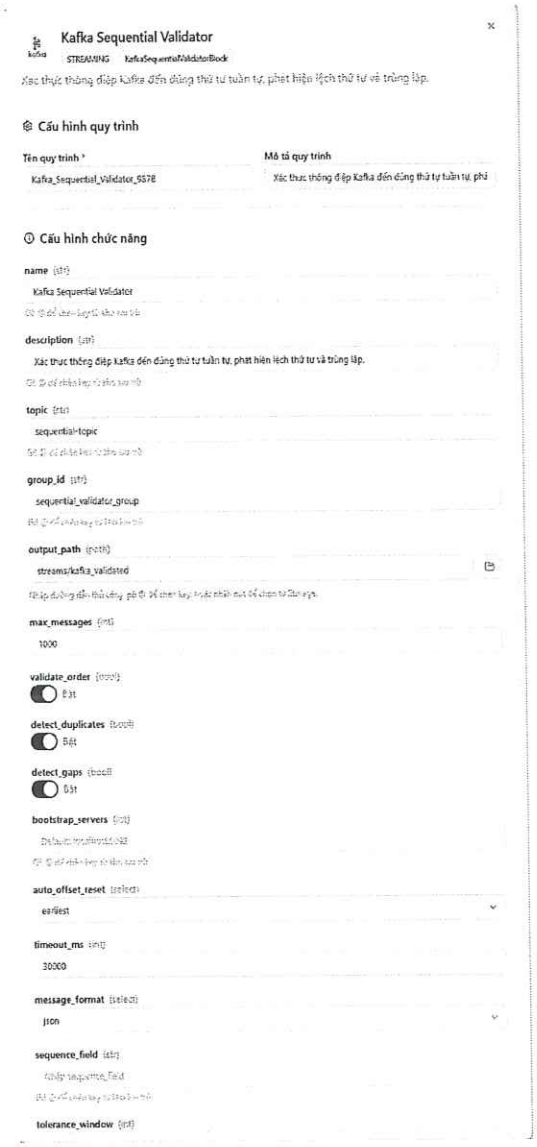
So với các phương pháp truyền thống như viết script Python (sử dụng requests và kafka-python) thủ công và cấu hình cron job để lập lịch thu thập dữ liệu, mô-đun KafkaGetAPIData trên nền tảng này mang lại nhiều ưu điểm vượt trội về tính trực quan, độ tin cậy và khả năng tái sử dụng.

Việc cấu hình toàn bộ quy trình—từ điểm cuối API, phương thức HTTP, tham số, headers, cho đến thông tin máy chủ Kafka và tên topic—đều được thực hiện hoàn toàn qua giao diện người dùng. Điều này giúp giảm thiểu sai sót kỹ thuật (như quên xử lý lỗi kết nối, quên đóng producer) và rút ngắn đáng kể thời gian triển khai.

Người dùng không cần phải ghi nhớ cú pháp phức tạp của thư viện Kafka hay viết logic boilerplate để khởi tạo Producer, AdminClient, xử lý tuần tự hóa (serialization) dữ liệu sang JSON, hay quản lý xác thực. Ngoài ra, hệ thống hỗ trợ sẵn các tính năng nâng cao như tự động tạo topic, cấu hình timeout và retries cho kết nối, và hỗ trợ các cơ chế xác thực phức tạp (cả API key cho API lẫn SASL cho Kafka). Khả năng thu thập lặp lại (polling) với poll_interval và max_requests, cùng khả năng xử lý linh hoạt các cấu trúc JSON trả về, giúp kiểm soát tốt hơn khi làm việc với các nguồn API không đồng nhất hoặc yêu cầu thu thập dữ liệu tuần tự.

Mô-đun cho phép người dùng kiểm tra tính hợp lệ của luồng xử lý dữ liệu tuần tự

Khối chức năng này cho phép người dùng kiểm tra tính tuần tự của luồng tin nhắn Kafka theo một trường "sequence" (hoặc fallback theo offset), phát hiện lệch thứ tự, trùng lặp, và khoảng trống (gaps), sau đó ghi kết quả chi tiết ra lưu trữ đối tượng (MinIO) và trả về một bản tóm tắt dạng DataFrame.



Hình 3.9: Giao diện cấu hình cho khối kiểm tra tính hợp lệ của luồng xử lý dữ liệu dạng tuần tự

Cài đặt trong Backend:

Lớp KafkaSequentialValidatorBlock là một khối chức năng (task block) xử lý luồng (streaming) chuyên dụng, được thiết kế để hoạt động như một thành phần giám sát và xác thực chất lượng dữ liệu (Data Quality) trong một pipeline Kafka.

Lớp này kế thừa từ hai (2) lớp cơ sở chính:

TaskGeneralBlock (Lớp Tác vụ Cơ sở):

- **Mục đích:** Việc kế thừa TaskGeneralBlock tích hợp lớp này vào lõi của hệ thống (flow core). Điều này cung cấp các phương thức và thuộc tính cần thiết để khối có thể được nhận diện, cấu hình và thực thi như một

mắt xích trong một luồng (flow) xử lý dữ liệu. Nó bắt buộc triển khai các phương thức như execute() (để thực thi) và show_config() (để định nghĩa giao diện).

MinioConnectionBlock (Lớp Kết nối MinIO):

- **Mục đích:** Việc kế thừa MinioConnectionBlock cung cấp cho lớp này các khả năng tích hợp sẵn để tương tác với một hệ thống lưu trữ đối tượng (object storage) tương thích S3 (như MinIO). Chức năng chính được sử dụng từ lớp này là phương thức self.save_file(), cho phép khối dễ dàng lưu trữ các tệp tin báo cáo (ví dụ: file JSON kết quả xác thực) lên MinIO.

Mục tiêu chính của KafkaSequentialValidatorBlock là đảm bảo tính toàn vẹn, tính đầy đủ và tính đúng thứ tự của các thông điệp (messages) được tiêu thụ từ một topic Kafka.

Nó hoạt động như một **Kafka Consumer** (bên tiêu thụ), kết nối vào một topic và thực hiện các nhiệm vụ xác thực theo thời gian thực (hoặc theo lô) trên một số lượng thông điệp được chỉ định (max_messages).

Các nhiệm vụ xử lý chính bao gồm:

- **Phát hiện trùng lặp (detect_duplicates):** Xác định và báo cáo các thông điệp có cùng một số thứ tự (sequence number) đã xuất hiện trước đó.
- **Phát hiện thông điệp lệch thứ tự (validate_order):** Xác định các thông điệp đến muộn, có số thứ tự nhỏ hơn số thứ tự đang được kỳ vọng.
- **Phát hiện lỗ hổng dữ liệu (detect_gaps):** Xác định các khoảng trống trong chuỗi tuần tự, khi một thông điệp đến có số thứ tự lớn hơn đáng kể so với số kỳ vọng (vượt quá tolerance_window).

Cấu trúc của lớp được bóc tách thành các phương thức chính với các vai trò rõ rệt:

Phương thức Khởi tạo và Cấu hình

- **__init__(self, ...) (Hàm khởi tạo):**
  - Đây là hàm khởi tạo của lớp. Nó tiếp nhận toàn bộ các tham số cấu hình từ người dùng (ví dụ: bootstrap_servers, topic, group_id, sequence_field, tolerance_window).

- Nó chịu trách nhiệm lưu trữ các tham số này vào từ điển self.parameters để các phương thức khác có thể truy cập.
- Nó cũng thực hiện chuẩn hóa đường dẫn output_path để lưu kết quả.

- **show_config(cls) (Định nghĩa lược đồ cấu hình):**
  - Là một phương thức của lớp (@classmethod), nó định nghĩa "lược đồ" (schema) cho giao diện người dùng (UI).
  - Phương thức này quy định các trường cấu hình, kiểu dữ liệu (ví dụ: str, int, bool, list, password, select), và giá trị mặc định cho chúng (như trong hình ảnh bạn đã cung cấp).

Phương thức Thực thi Chính (Core Execution)

execute(self, data: Any = None):

- Đây là phương thức điều phối chính, được gọi khi khối được thực thi. Toàn bộ logic xác thực nằm ở đây, được bọc trong một khối try...finally để đảm bảo consumer luôn được đóng (consumer.close()).
- **Luồng hoạt động:**
  1. Gọi _create_consumer() để thiết lập kết nối đến Kafka.
  2. Khởi tạo các biến quản lý trạng thái:
     - seen_sequences (kiểu set): Để theo dõi các số thứ tự đã thấy (phát hiện trùng lặp).
     - expected_sequence (kiểu int): Số thứ tự tiếp theo được kỳ vọng (phát hiện lệch thứ tự và lỗ hổng).
     - validation_results (kiểu dict): Để tổng hợp số liệu thống kê lỗi.
  3. Bắt đầu vòng lặp (for message in consumer:), đọc từng thông điệp cho đến khi đạt max_messages hoặc hết timeout_ms.
  4. **Bên trong vòng lặp:** a. Gọi _extract_sequence(message_data) để lấy số thứ tự. b. Thực hiện logic so sánh seq với seen_sequences và expected_sequence để cập nhật validation_results (đếm duplicates, out_of_order, gaps). c. Thêm thông điệp và trạng thái xác thực (is_valid) vào một danh sách messages tạm thời.

5. **Sau vòng lặp:** a. Chuyển đổi toàn bộ kết quả (validation_results và messages) thành một chuỗi JSON. b. Sử dụng phương thức self.save_file() (kế thừa từ MinioConnectionBlock) để lưu tệp JSON kết quả này vào output_path trên MinIO. c. Trả về một pd.DataFrame tóm tắt chứa đường dẫn đến tệp kết quả và các số liệu thống kê chính.

Phương thức Trợ giúp Kỹ thuật (Private Helpers)

_create_consumer(self):

- Đây là phương thức kỹ thuật chịu trách nhiệm khởi tạo đối tượng KafkaConsumer.
- Nó xây dựng dict cấu hình kết nối, bao gồm bootstrap_servers, group_id, auto_offset_reset.
- **Tính năng quan trọng:** Nó tự động cấu hình value_deserializer (bộ giải tuần tự) dựa trên message_format. Nếu là json, nó sẽ tự động dùng json.loads để chuyển đổi thông điệp byte thành đối tượng dict của Python.

extract_sequence(self, message: Dict[str, Any]):

- Đây là hàm chứa logic nghiệp vụ cốt lõi để xác định "thứ tự" của một thông điệp.
- **Logic ưu tiên:** Nó sẽ cố gắng trích xuất số thứ tự từ trường (sequence_field) mà người dùng đã chỉ định trong nội dung value của thông điệp (ví dụ: message['value']['id']).
- **Logic dự phòng (Fallback):** Nếu sequence_field không được cung cấp hoặc không tìm thấy, nó sẽ tự động sử dụng message.get("offset") (số offset do Kafka quản lý) làm số thứ tự.

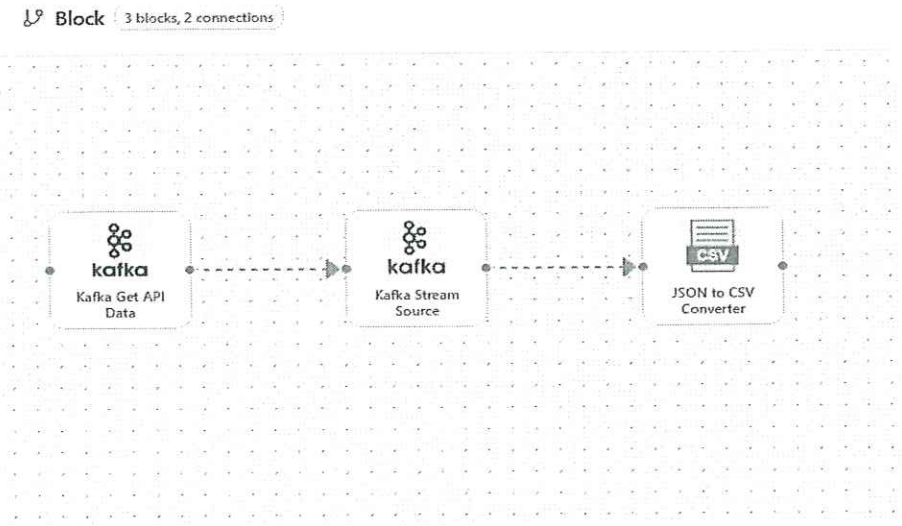
```python
from __future__ import annotations

import uuid
import os
import json
from io import BytesIO
from typing import Any, Dict, Optional, List
import pandas as pd

# Kafka imports with compatibility handling
try:
    from kafka import KafkaConsumer
    from kafka.errors import KafkaError
    KAFKA_AVAILABLE = True
except ImportError as e:
    print(f"Kafka library not available: {e}")
    KAFKA_AVAILABLE = False
    class KafkaConsumer:
        def __init__(self, *args, **kwargs):
            raise Exception("Kafka library not properly installed")
    class KafkaError(Exception):
        pass

from CoreModules.flowcore.baseBlockClass import TaskGeneralBlock, TaskCategory
from CoreModules.flowcore.baseMinIO import MinioConnectionBlock


class KafkaSequentialValidatorBlock(TaskGeneralBlock, MinioConnectionBlock):
    """
    Validate that Kafka messages arrive in correct sequential order.
    Detects out-of-order messages, gaps, and duplicates.
    """

    category = TaskCategory.STREAMING
    description = "Validate Kafka messages arrive in correct sequential order"
    img_url = "/static/icons/ingestion/kafka.png"

    def __init__(…
    ) -> None: …

    @classmethod
    def from_config(cls, config: Dict[str, Any]) -> "KafkaSequentialValidatorBlock": …

    @classmethod
    def show_config(cls) -> Dict[str, Any]: …

    def _create_consumer(self) -> KafkaConsumer: …

    def _extract_sequence(self, message: Dict[str, Any]) -> Optional[int]: …

    def _resolve_username(self, data: Any) -> str: …

    def execute(self, data: Any = None) -> pd.DataFrame: …
```

Hình 3.10: Cài đặt khối kiểm tra tính tuần tự trong luồng xử lý dữ liệu

Ví dụ: Lấy dữ liệu từ 1 API chuyển đổi thành tệp CSV

Mục tiêu: Thử nghiệm khả năng lấy dữ liệu theo lô bằng Kafka, đầu ra tạo thành file .CSV có thể sử dụng để biến đổi thành DataFrame, phù hợp cho việc truy vấn, phân tích dữ liệu

Tổng thể của luồng xử lý:



Hình 3.11: Tổng thể luồng thiết kế

Cấu hình các khối

- **KafkaGetAPIData Block:**
  - api_uirl : https://jsonplaceholder.typicode.com/comments
  - topic: api-data-topic
  - bootstrap_server: 112.137.129.245:30002
  - http_method: GET
  - max_request: 1
  - timeout: 30
- **KafkaStreamSource**
  - topic: api-data-topic
  - bootstrap_server: 112.137.129.245:30002
  - message_format: json
  - timeout_ms: 30000
  - output_path: streams/kafka
- **JSONtoCSVConverter**
  - default params

Luồng thực hiện:

1. Người dùng kéo thả 3 khối KafkaGetAPI, KafkaStreamSource, JSONtoCSVConverter
2. Nối 3 mô-đun này với nhau thực hiện kích hoạt chạy thử luồng xử lý
3. Hệ thống nhận biết tạo producer bằng API đã được cung cấp

4. Hệ thống đẩy messages do producer vào các broker,broker kiểm tra xem topic đã tồn tại hay chưa, nếu có rồi thì truyền vào topic, nếu chưa thì tạo mới
5. Mô-đun KafkaStreamSource tạo consumer lấy messages từ topic và lưu vào đường dẫn đã tạo trong mô-đun
6. Mô-đun JSONtoCSV Converter sẽ chuyển đổi output JSON thành định dạng .csv

Kết quả:

- Trạng thái: Thành công
- số bản ghi: 500
- Thời gian thực thi: ~ 37 giây
- Log: Không có lỗi, luồng được thực thi thành công

3.2.3. Một số mô-đun thực thi lênh tải dữ liệu dạng sự kiện về hồ dữ liệu

Mô-đun cho phép người dùng sử dụng engine của Apache Kafka để tạo một pipeline streaming data từ API

Mô-đun này trình bày chi tiết về một giải pháp chuyên dụng sử dụng Apache Kafka làm nền tảng trung tâm để thu thập dữ liệu dạng Streaming từ các API URL. Nó cung cấp khả năng cấu hình linh hoạt phục vụ được cả cho 2 nhu cầu Streaming và Event. Giải pháp này đảm bảo được việc trích xuất dữ liệu từ API được thực hiện một cách hiệu quả, đáng tin cậy và có khả năng mở rộng

Hình 3.12: Giao diện cấu hình mô-đun kéo dữ liệu về hồ dữ liệu sử dụng Kafka

Cài đặt trong backend:

Class KafkaGetAPIDataBlock là một component quan trọng trong hệ thống Data Pipeline, được thiết kế để thực hiện vai trò kép trong quá trình thu thập và phân phối dữ liệu. Class này hoạt động như một cầu nối giữa các REST API endpoints và Apache Kafka message broker, cho phép tự động hóa quy trình ingestion dữ liệu từ các nguồn bên ngoài vào hệ thống streaming.

Class được xây dựng dựa trên kiến trúc đa kế thừa (multiple inheritance), kế thừa từ hai class cha:

- **TaskGeneralBlock**: Cung cấp framework cơ bản cho các task trong workflow, bao gồm quản lý metadata, lifecycle và execution context
- **MinioConnectionBlock**: Cung cấp khả năng kết nối và tương tác với MinIO object storage để lưu trữ metadata và artifacts

Mô-đun được phân loại thuộc TaskCategory.INGESTION, nghĩa là nó thuộc nhóm các components chịu trách nhiệm thu thập dữ liệu từ nguồn bên ngoài vào hệ thống.

Class hỗ trợ đầy đủ các tham số để tương tác với REST API:

Thông tin kết nối cơ bản:

- api_url (str): URL endpoint của API cần fetch dữ liệu
- http_method (str): Phương thức HTTP, hỗ trợ GET và POST
- timeout (int): Thời gian timeout cho mỗi request, mặc định 30 giây

Xác thực và bảo mật:

- api_key_header (Optional[str]): Tên header chứa API key
- api_key_value (Optional[str]): Giá trị API key để authentication
- headers (Dict[str, str]): Custom headers cho request

Dữ liệu request:

- query_params (Dict[str, str]): Query parameters cho URL
- body (Dict[str, Any]): Request body cho POST method

Cấu hình Kafka Infrastructure

Thông tin broker và topic:

- bootstrap_servers (List[str]): Danh sách địa chỉ Kafka brokers

- topic (str): Tên topic đích để produce messages

Cấu hình topic:

- num_partitions (int): Số lượng partitions cho topic, mặc định 1
- replication_factor (int): Hệ số nhân bản cho độ tin cậy, mặc định 1

Xác thực SASL:

- username (Optional[str]): Username cho SASL authentication
- password (Optional[str]): Password cho SASL authentication

Tham số điều khiển luồng xử lý

Kiểm soát polling:

- poll_interval (int): Thời gian chờ (giây) giữa các lần gọi API, mặc định 0
- max_requests (int): Số lần gọi API tối đa trong một lần execute, mặc định 1

Batch processing:

- batch_size (int): Kích thước batch cho việc xử lý, mặc định 100

Cấu hình output

Đường dẫn lưu trữ:

- output_path (str): Đường dẫn để lưu metadata trong MinIO

Flags điều khiển:

- produce_to_kafka (bool): Bật/tắt chức năng produce message vào Kafka, mặc định True
- save_data_to_minio (bool): Bật/tắt chức năng lưu dữ liệu vào MinIO, mặc định False

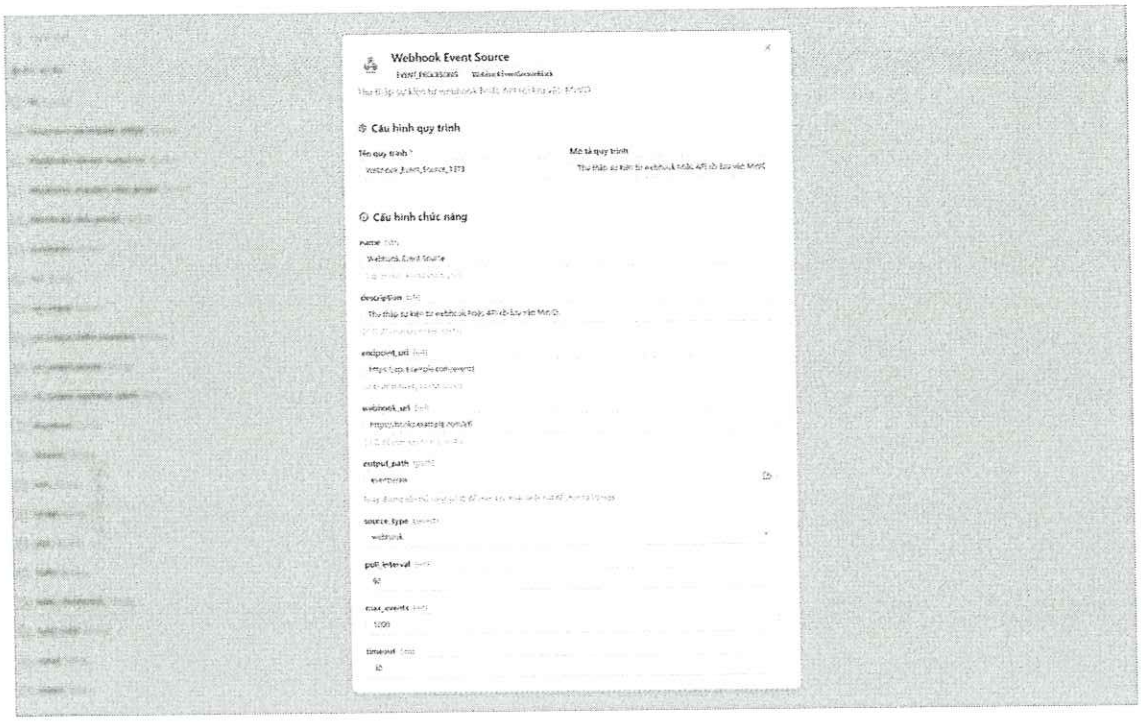
```
class KafkaGetAPIDataBlock(TaskGeneralBlock, MinioConnectionBlock):
    """Fetch data from API and produce to Kafka topic."""

    category = TaskCategory.INGESTION
    description = "Fetch data from API, produce to Kafka topic, and save metadata to MinIO"
    img_url = "/static/icons/ingestion/kafkaapi.png"

    def __init__(...
    ) -> None: ...

    @classmethod
    def from_config(cls, config: Dict[str, Any]) -> "KafkaGetAPIDataBlock": ...

    @classmethod
    def show_config(cls) -> Dict[str, Any]: ...

    def _create_or_get_topic(self, admin_client: KafkaAdminClient) -> bool: ...

    def _create_producer(self) -> KafkaProducer:
        """Create Kafka producer."""
        if not KAFKA_AVAILABLE: ...

        try: ...

        except KafkaError as e: ...

    def _create_admin_client(self) -> KafkaAdminClient: ...

    def _fetch_data_from_api(self) -> List[Dict[str, Any]]: ...

    def _resolve_username(self, data: Any) -> str: ...

    def execute(self, data: Any = None) -> pd.DataFrame: ...
```

Hình 3.13: Cấu trúc lớp xử lý dữ liệu tuần tự sử dụng Kafka

Mô-đun cho phép người dùng sử dụng Mô-đun Webhook Event Source để thu thập event từ REST API, URL và lưu kết quả output

WebhookEventSourceBlock là một khối chức năng trong hệ thống xử lý dữ liệu event-driven, được thiết kế để thu thập các sự kiện từ các nguồn webhook hoặc HTTP endpoint. Khối này thuộc nhóm EVENT_PROCESSING, đóng vai trò là điểm khởi đầu trong các workflow xử lý sự kiện, nơi các event độc lập với nhau và không yêu cầu tính tuần tự xử lý.

Khối kế thừa từ hai lớp cơ sở là TaskGeneralBlock và MinioConnectionBlock, cho phép tích hợp đầy đủ với workflow engine và hệ thống lưu trữ MinIO. Các sự kiện được thu thập sẽ được lưu trữ dưới định dạng JSON để phục vụ cho các bước xử lý tiếp theo trong luồng dữ liệu.



Hình 3.14: Giao diện cấu hình cho khối WebhookEventSource

Cài đặt trong Backend:

WebhookEventSourceBlock hỗ trợ ba phương thức thu thập sự kiện chính:

HTTP Polling: Khối thực hiện polling định kỳ đến một HTTP endpoint để lấy dữ liệu. Phương thức này phù hợp với các nguồn cung cấp API REST, nơi dữ liệu được cập nhật liên tục và cần được thu thập theo chu kỳ. Hệ thống tự động xử lý nhiều định dạng response khác nhau bao gồm array, object với key "events" hoặc "data", và single object.

Webhook Mode: Phương thức này được thiết kế để nhận các sự kiện được đẩy từ nguồn thông qua webhook. Hiện tại đang ở giai đoạn mô phỏng với các event mẫu, và sẽ được phát triển thành webhook server thực tế trong các phiên bản tiếp theo.

REST API: Tương tự HTTP polling nhưng được tối ưu hóa cho các API endpoint cụ thể với các tham số và header được cấu hình sẵn.

Khi khối được thực thi, quy trình xử lý diễn ra theo các bước sau:

1. Đầu tiên, khối thiết lập các HTTP headers cần thiết, bao gồm authentication token nếu có. Các header mặc định như Content-Type và User-Agent cũng được thêm vào để đảm bảo tương thích với các API endpoint.

2. Tiếp theo, khối thực hiện thu thập events từ nguồn đã cấu hình. Đối với HTTP polling, một GET request được gửi đến endpoint URL. Response được parse và xử lý để trích xuất danh sách events, bất kể định dạng response là gì.
3. Sau khi thu thập, mỗi event sẽ được tự động bổ sung metadata quan trọng bao gồm timestamp (thời điểm thu thập), source (nguồn gốc của event), sequence (số thứ tự), endpoint URL, và ID duy nhất. Các thông tin này giúp truy vết và quản lý events trong suốt quá trình xử lý.
4. Nếu được cấu hình event filter, khối sẽ áp dụng các điều kiện lọc để chỉ giữ lại những events thỏa mãn tất cả các tiêu chí. Cơ chế lọc hoạt động dựa trên nguyên tắc exact match cho từng cặp key-value trong filter.
5. Cuối cùng, danh sách events sau khi được xử lý và lọc sẽ được chuyển đổi thành pandas DataFrame, sau đó được serialize thành JSON và lưu trữ vào MinIO. Tên file được tạo tự động theo format events_{timestamp}_{uuid}.json để đảm bảo tính duy nhất.

WebhookEventSourceBlock được cài đặt tại BE/CoreModules/tasks/ingestion/webhook_source.py, kế thừa từ hai lớp cơ sở:

- **TaskGeneralBlock**: Cung cấp khung xương cơ bản của một mô-đun trong workflow
- **MinioConnectionBlock**: Cung cấp khả năng kết nối và lưu trữ dữ liệu vào MinIO

Mô-đun thuộc category TaskCategory.EVENT_PROCESSING với icon /static/icons/ingestion/webhook.png.

Các phương thức chính và cấu hình của lớp

Phương thức khởi tạo (__init__)

Nhận 11 tham số cấu hình bao gồm source_type, endpoint_url, poll_interval,max_events,timeout, headers, auth_token, output_path, event_filter và minIOusername. Phương thức này khởi tạo các lớp cha, xử lý output_path (loại bỏ dấu "/" đầu tiên, tự động tạo path dạng events/{uuid} nếu rỗng), và lưu trữ tất cả parameters vào self.parameters.

Phương thức thu thập events

- **_poll_events()**: Thu thập events từ HTTP endpoint bằng GET request. Phương thức này kiểm tra endpoint_url, setup headers, thực hiện HTTP request, parse JSON response (hỗ trợ nhiều format: array, object với

key "events"/"data", single object), thêm metadata (_timestamp, _source, _sequence, _endpoint) vào mỗi event, và giới hạn số lượng theo max_events.

- **_simulate_webhook_events()**: Tạo các events mẫu để mô phỏng webhook (user_signup, order_placed, payment_received). Mỗi event được thêm metadata bao gồm _timestamp, _source, _sequence và _id duy nhất.

Phương thức xử lý và hỗ trợ

- **_setup_request_headers()**: Chuẩn bị HTTP headers bằng cách sao chép custom headers từ config, thêm Authorization header nếu có auth_token, và thiết lập các headers mặc định (Content-Type, User-Agent).
- **_apply_event_filter()**: Lọc events dựa trên event_filter. Duyệt qua từng event và chỉ giữ lại những events có tất cả các key-value khớp với điều kiện trong filter.

Phương thức thực thi chính (execute)

Đây là phương thức core thực hiện toàn bộ quy trình xử lý:

1. Thu thập events dựa trên source_type (gọi _poll_events() hoặc _simulate_webhook_events())
2. Áp dụng filter bằng _apply_event_filter()
3. Chuyển đổi danh sách events sang pandas DataFrame
4. Serialize DataFrame thành JSON và tạo buffer
5. Upload lên MinIO với tên file events_{timestamp}_{uuid}.json
6. Tạo và trả về DataFrame chứa metadata (path, storage_url, events_collected, collection_timestamp, file_size_bytes, event_types)

Phương thức factory pattern

- **from_config()**: Class method tạo instance từ dictionary config, sử dụng khi load workflow từ JSON.
- **show_config()**: Class method trả về schema định nghĩa các parameters với type, options, default values và descriptions, phục vụ cho việc render UI form cấu hình.

CƠ CHẾ XỬ LÝ LỖI

Tất cả các phương thức đều được bọc trong try-except blocks. Các lỗi HTTP request được bắt qua requests.exceptions.RequestException, lỗi thiếu tham số được raise dưới dạng ValueError, và các lỗi khác được wrap trong Exception với message mô tả rõ ràng. Phương thức execute() đảm bảo trả về DataFrame ngay cả khi không có events (với status "no_events").

BE › CoreModules › tasks › ingestion › webhook_source.py

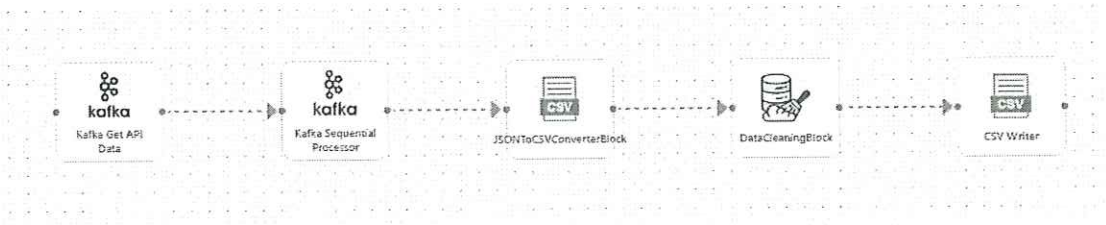
```python
from __future__ import annotations

import uuid
import os
import json
import time
from io import BytesIO
from typing import Any, Dict, Optional, List
import pandas as pd
import requests
from datetime import datetime, timedelta

from CoreModules.flowcore.baseBlockClass import TaskGeneralBlock, TaskCategory
from CoreModules.flowcore.baseMinIO import MinioConnectionBlock


class WebhookEventSourceBlock(TaskGeneralBlock, MinioConnectionBlock):
    """Listen for webhook events or poll HTTP endpoints for events."""

    category = TaskCategory.EVENT_PROCESSING
    description = "Collect events from webhooks or HTTP endpoints and save to MinIO"
    img_url = "/static/icons/ingestion/webhook.png"

    def __init__(...
    ) -> None: ...

    @classmethod
    def from_config(cls, config: Dict[str, Any]) -> "WebhookEventSourceBlock": ...

    @classmethod
    def show_config(cls) -> Dict[str, Any]: ...

    def _setup_request_headers(self) -> Dict[str, str]: ...

    def _poll_events(self) -> List[Dict[str, Any]]: ...

    def _simulate_webhook_events(self) -> List[Dict[str, Any]]: ...

    def _apply_event_filter(self, events: List[Dict[str, Any]]) -> List[Dict[str, Any]]: ...


    def execute(self, data: Any = None) -> pd.DataFrame: ...
```

Hình 3.15: Cài đặt phần backend của lớp WebhookEventSourceBlock

Ví dụ: Lấy dữ liệu từ API theo thời gian thực sử dụng mô-đun KafkaGetAPIData để lấy dữ liệu dạng Streaming lưu vào hồ dữ liệu

Mục tiêu: Thử nghiệm khả năng fetch dữ liệu từ API realtime bằng Kafka, đầu ra tạo thành file CSV đã được làm sạch, chuẩn hoá, đủ tiêu chuẩn để sử dụng làm DataFrame thích hợp cho việc phân tích, truy vấn

Tổng thể của luồng xử lý:



Hình 3.16: Tổng thể luồng xử lý dữ liệu Streaming

Cấu hình các khối:

1. API-Source-1 (KafkaGetAPIDataBlock)

Khối này có nhiệm vụ gọi API định kỳ và đẩy dữ liệu vào Kafka.

- API URL: https://jsonplaceholder.typicode.com/comments (Phương thức: GET)
- Máy chủ Kafka: 112.137.129.245:30002
- Topic Kafka: streaming-api-data
- Tần suất Polling: 30 giây
- Đẩy vào Kafka: true (Kích hoạt)
- Lưu trực tiếp vào MinIO: false (Tắt)
- Đường dẫn (MinIO): streaming/raw (Dùng để lưu metadata)
- Người dùng MinIO: uet

2. Processor-1 (KafkaSequentialProcessorBlock)

Khối này đọc dữ liệu từ Kafka, xử lý tuần tự và duy trì trạng thái.

- Máy chủ Kafka: 112.137.129.245:30002
- Topic (đọc từ): streaming-api-data
- Group ID: streaming_processor_group
- Offset Reset: earliest (Đọc từ thông điệp cũ nhất)
- Lưu trạng thái: true (Kích hoạt)

- Đường dẫn output (Dữ liệu): streaming/processed
- Đường dẫn output (Trạng thái):T streaming/state
- Người dùng MinIO: uet

3. JSON-Converter-1 (JSONToCSVConverterBlock)

Khối này nhận dữ liệu (dạng JSON) từ khối trước và chuyển đổi sang định dạng DataFrame (để chuẩn bị ghi CSV).

- Key chứa JSON: value (Lấy dữ liệu từ trường value của thông điệp)
- Key chứa dữ liệu bên trong: data
- Làm phẳng (Normalize) JSON lồng nhau: true (Kích hoạt)
- Đường dẫn output (MinIO): streaming/converted
- Người dùng MinIO: uet

4. Cleaning-1 (DataCleaningBlock)

Khối này thực hiện các tác vụ làm sạch trên DataFrame.

- Xử lý giá trị Null: true
- Chiến lược xử lý Null: drop (Xóa các hàng chứa giá trị null)
- Xóa trùng lặp: true
- Giữ lại (khi trùng): first (Giữ lại bản ghi đầu tiên phát hiện)
- Xử lý ngoại lai: false (Tắt)
- Chuẩn hóa Text: false (Tắt)
- Người dùng MinIO: uet

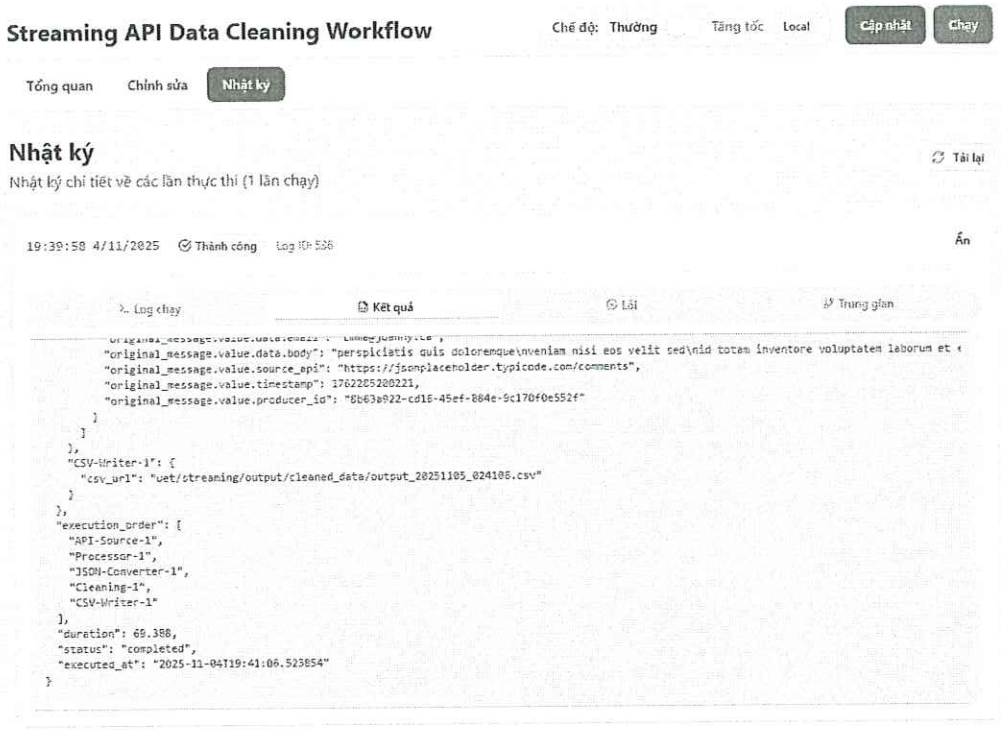
5. CSV-Writer-1 (CSVWriterBlock)

Khối cuối cùng ghi DataFrame đã làm sạch ra file CSV trong MinIO.

- Đường dẫn file (MinIO): streaming/output/cleaned_data
- Dấu phân cách: ,
- Ghi kèm Index: false (Không ghi cột index của DataFrame)
- Encoding: utf-8
- Người dùng MinIO: uet

Kết quả:

- **Trạng thái:** Thành công
- **Số bản ghi:** 500
- **Refresh mỗi sau:** 30 giây
- **Log:** Chạy thành công trong vòng 70s



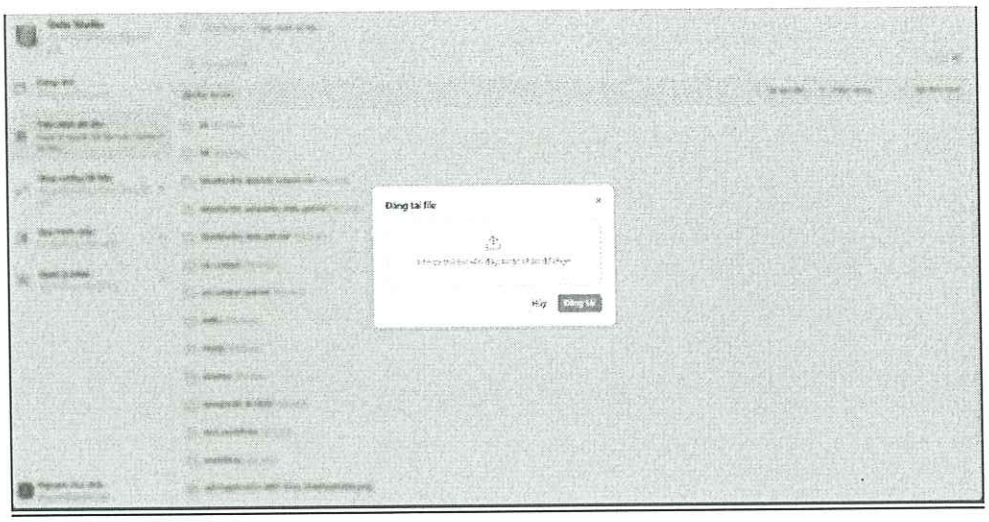
Hình 3.17: Kết quả của luồng xử lý dữ liệu Streaming

3.2.4. Một số mô-đun thực thi lệnh tải dữ liệu dạng tệp về hồ dữ liệu

Dưới đây là mẫu cấu trúc mô tả chi tiết cho từng mô-đun tiếp nhận dữ liệu DẠNG TỆP trong hệ thống:

Mô-đun tải tệp dữ liệu DẠNG TỆP lên trực tiếp lên kho lưu trữ của người dùng

Khối cho phép người dùng tải tệp dữ liệu tệp hình ảnh, văn bản từ máy cá nhân lên hệ thống lưu trữ nội bộ hoặc MinIO, phục vụ cho các bước xử lý tiếp theo trong luồng.**Giao diện cấu hình:** Giao diện khối được thiết kế đơn giản và thân thiện, bao gồm: Khu vực kéo-thả hoặc chọn tệp; Hiển thị tên tệp, dung lượng, định dạng sau khi tải lên; "Đăng tải" để xác nhận lưu tệp, "Hủy" để thoát.

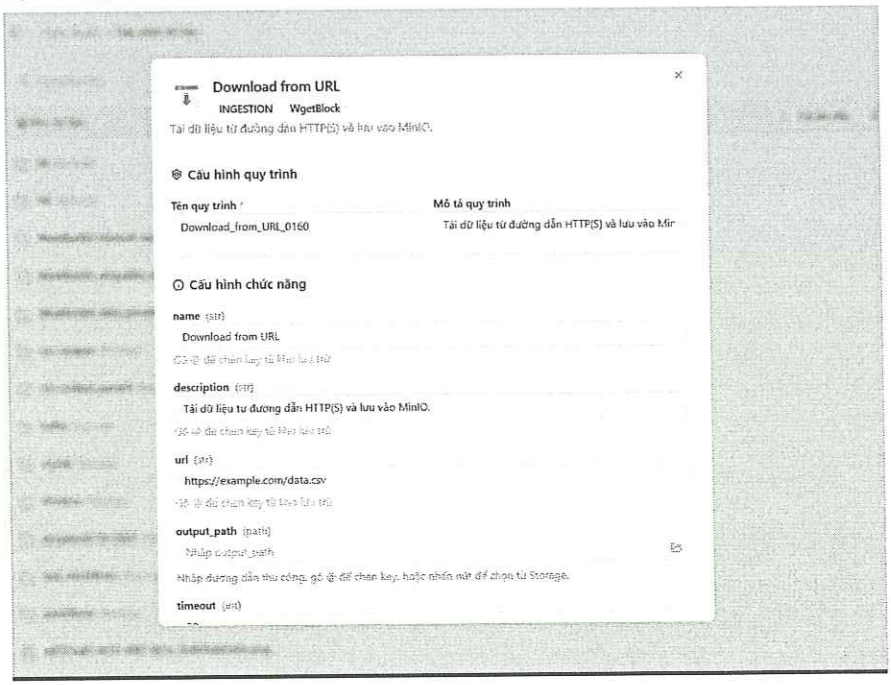


Hình 3.18: Mô-đun tải tệp lên kho lưu trữ của người dùng

Khi người dùng thực hiện thao tác kéo-thả hoặc chọn tệp để đăng tải, hệ thống sẽ tiếp nhận nội dung tệp dưới dạng stream và ghi trực tiếp vào kho lưu trữ theo đường dẫn được cấu hình hoặc sinh tự động. Việc ghi dữ liệu diễn ra ngay trong quá trình thực thi khối, không phụ thuộc vào nguồn bên ngoài hay kết nối mạng, giúp đảm bảo độ ổn định và tốc độ xử lý cao.

Mô-đun thu nhận file bảng từ URL (HTTP/FTP)

Khối thực hiện tải dữ liệu bảng từ địa chỉ URL (HTTP/HTTPS) và lưu trữ vào hệ thống tạm hoặc kho lưu trữ như MinIO.



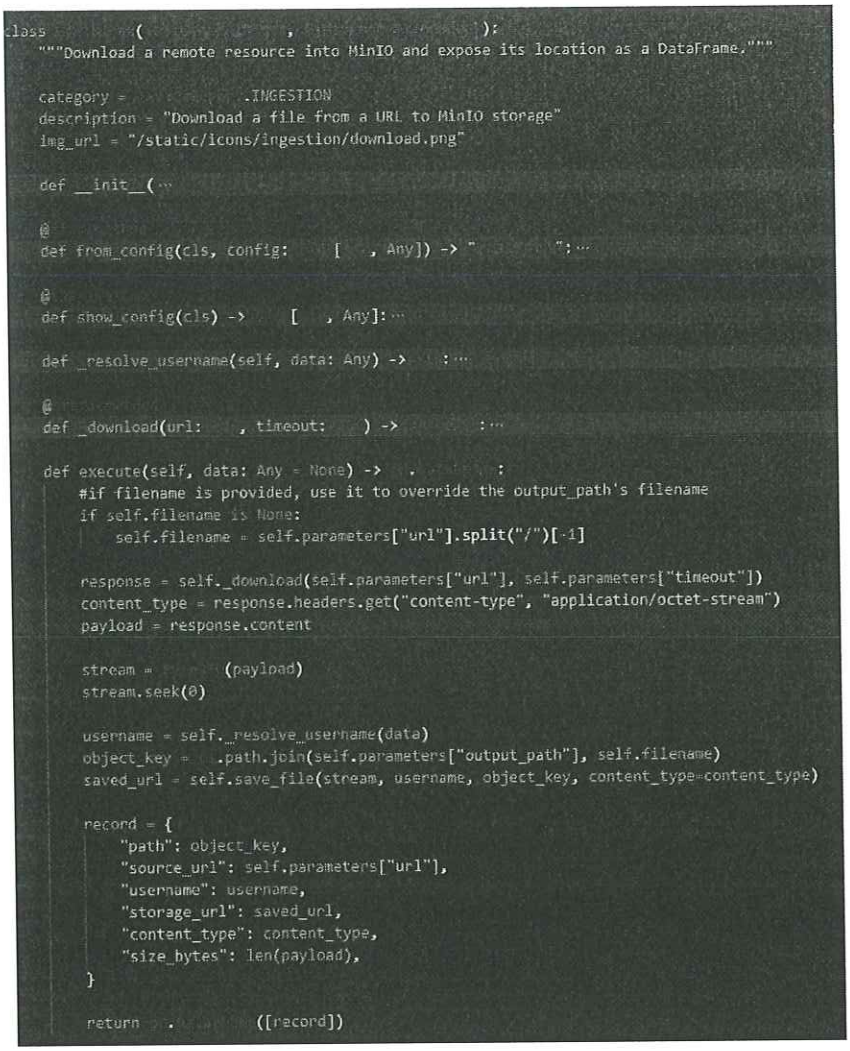
Hình 3.19: Giao diện cấu hình mô-đun kéo một tệp DẠNG TỆP từ Internet

Cài đặt trong backend: Trong backend, tương tự khối MySQL, khối này được định nghĩa dưới dạng lớp , kế thừa từ hai lớp cha `TaskGeneralBlock` và `MinioConnectionBlock`. Mục tiêu chính của khối là thực hiện tải file từ một địa chỉ URL (thường là file CSV, Excel hoặc ZIP), lưu nội dung vào MinIO, và trả về thông tin metadata dưới dạng tệp để sử dụng trong các bước xử lý tiếp theo.

Cấu trúc lớp được tổ chức rõ ràng với các thuộc tính mô tả như `description` và `icon_url`, phục vụ cho việc hiển thị trong giao diện người dùng. Phương thức `task_generator()` xác định loại tác vụ là `INGESTION`, giúp hệ thống phân loại khối trong luồng. Việc khởi tạo khối từ cấu hình được thực hiện qua hàm `from_config(config)`, trong khi `show_config()` trả về cấu hình hiện tại để phục vụ hiển thị hoặc ghi log.

Các hàm xử lý chính bao gồm `_download(url, content_type)`, thực hiện HTTP GET tới địa chỉ URL và trả về nội dung file dưới dạng `requests.Response`. Hàm `_download_url(url)` trích xuất nội dung file dưới dạng `bytes`, phục vụ cho việc ghi vào stream. Trong phương thức `execute(data)`, hệ thống xác định tên file (từ URL hoặc cấu hình), tải nội dung về dưới dạng stream, và gọi `_upload(payload)` để lưu vào MinIO. Sau đó, hàm `_record(...)` ghi lại metadata gồm URL, tên file, người dùng, vị trí lưu trữ và loại nội dung. Kết quả cuối cùng được trả về dưới dạng một `pandas.DataFrame` chứa một dòng mô tả file đã tải, giúp luồng có thể tiếp tục xử lý hoặc kiểm tra dữ liệu.

Việc sử dụng `BytesIO` để xử lý nội dung tải về giúp tối ưu hiệu năng và giảm tải bộ nhớ, đặc biệt khi làm việc với file lớn. Cấu trúc này cũng cho phép mở rộng dễ dàng để hỗ trợ các giao thức khác như FTP, hoặc tích hợp xác thực nâng cao trong các phiên bản tiếp theo.

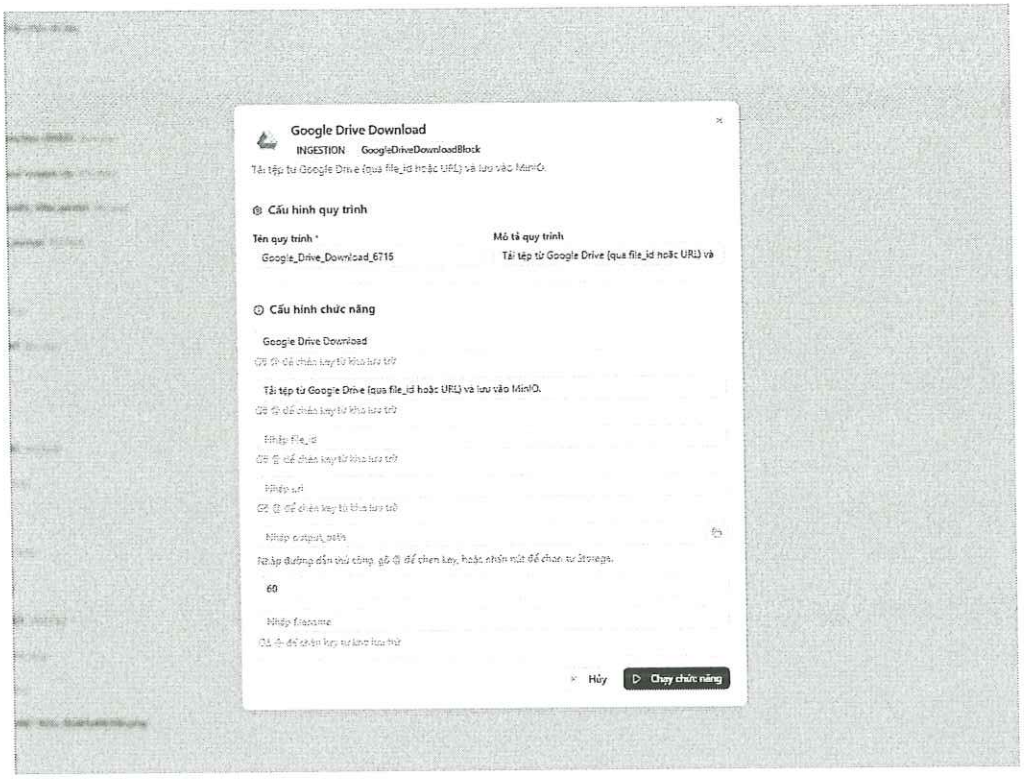


```
class          (                ,                    ):
    """Download a remote resource into MinIO and expose its location as a DataFrame."""

    category =              .INGESTION
    description = "Download a file from a URL to MinIO storage"
    img_url = "/static/icons/ingestion/download.png"

    def __init__(…

    @
    def from_config(cls, config:     [    , Any]) -> "          ";…

    @
    def show_config(cls) ->     [    , Any]:…

    def _resolve_username(self, data: Any) ->     :…

    @
    def _download(url:     , timeout:     ) ->          :…

    def execute(self, data: Any = None) ->    .          :
        #if filename is provided, use it to override the output_path's filename
        if self.filename is None:
            self.filename = self.parameters["url"].split("/")[-1]

        response = self._download(self.parameters["url"], self.parameters["timeout"])
        content_type = response.headers.get("content-type", "application/octet-stream")
        payload = response.content

        stream =          (payload)
        stream.seek(0)

        username = self._resolve_username(data)
        object_key =    .path.join(self.parameters["output_path"], self.filename)
        saved_url = self.save_file(stream, username, object_key, content_type=content_type)

        record = {
            "path": object_key,
            "source_url": self.parameters["url"],
            "username": username,
            "storage_url": saved_url,
            "content_type": content_type,
            "size_bytes": len(payload),
        }

        return    .          ([record])
```

Hình 3.20: Cài đặt khối kéo dữ liệu từ Internet trong backend

Mô-đun thu nhận file bảng từ google drive

Khối này sẽ thực hiện tải dữ liệu từ Google Drive thông qua liên kết chia sẻ (public share link) và lưu trữ vào hệ thống tạm hoặc kho lưu trữ như MinIO. Đây là một mô-đun quan trọng trong các pipeline ingest dữ liệu phi cấu trúc, đặc biệt là ảnh và văn bản phục vụ huấn luyện mô hình TTNT.



Hình 3.21: Cài đặt khối kéo dữ liệu từ google drive trong backend

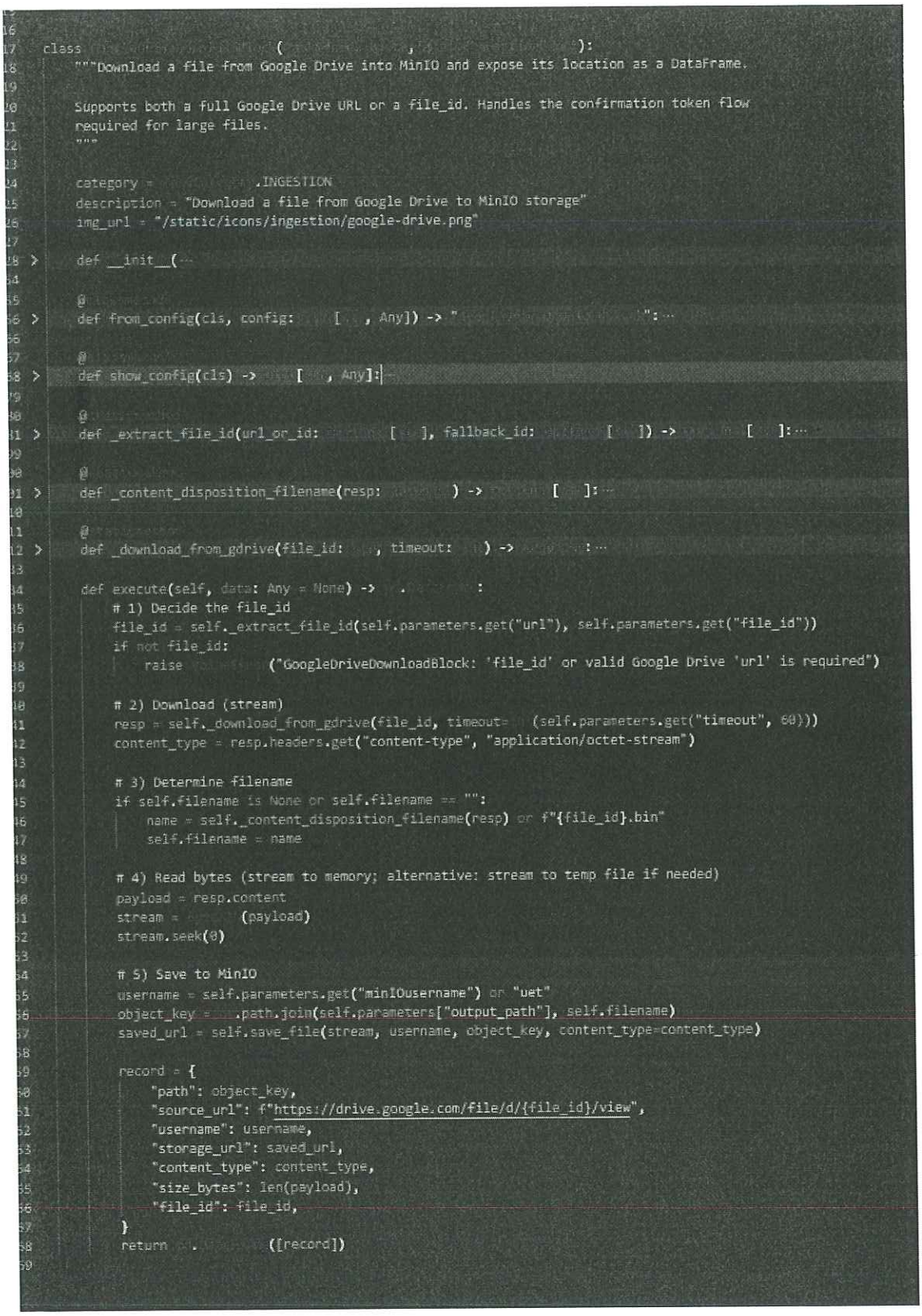
Cài đặt trong backend: Khối này được định nghĩa dưới dạng lớp kế thừa từ hai lớp cha `TaskGeneralBlock` và `MinioConnectionBlock`. Mục tiêu chính của khối là thực hiện tải file từ một địa chỉ URL (thường là file văn bản, ảnh hoặc các file nén), lưu nội dung vào MinIO, và trả về thông tin metadata để sử dụng trong các bước xử lý tiếp theo..

Mô-đun `GoogleDriveDownloadBlock` được triển khai dưới dạng một lớp xử lý backend có nhiệm vụ tải tệp từ Google Drive thông qua liên kết chia sẻ hoặc `file_id`, sau đó lưu trữ vào hệ thống MinIO và trả về thông tin metadata dưới dạng bảng (`DataFrame`). Đây là một khối ingestion quan trọng trong pipeline xử lý dữ liệu phi cấu trúc, đặc biệt với các tập tin nén chứa ảnh hoặc văn bản phục vụ huấn luyện mô hình TTNT.

Phương thức chính `execute()` thực hiện toàn bộ quy trình ingest theo các bước tuần tự. Đầu tiên, hệ thống xác định `file_id` từ tham số đầu vào, có thể là một URL chia sẻ hoặc chuỗi ID trực tiếp. Nếu không tìm thấy `file_id`, hệ thống sẽ dừng và trả về lỗi cấu hình. Tiếp theo, khối thực hiện tải nội dung tệp từ Google Drive bằng phương thức `_download_from_gdrive()`, sử dụng kết nối HTTP và timeout cấu hình (mặc định 60 giây). Sau khi nhận được phản hồi, hệ thống trích xuất `content-type` để xác định loại nội dung.

Tên tệp được xác định từ header `Content-Disposition` nếu có, hoặc mặc định là `{file_id}.bin`. Nội dung tệp được đọc vào bộ nhớ dưới dạng `bytes`, sử dụng `BytesIO` để tạo stream — cho phép xử lý linh hoạt và tối ưu hiệu năng, đặc biệt với các tệp lớn. Sau đó, hệ thống lưu stream vào MinIO thông qua phương thức `save_file()`, sử dụng `username` và `output_path` đã cấu hình để tạo `object_key`. Tệp được ghi kèm theo thông tin `content_type` để phục vụ các bước xử lý downstream.

Cuối cùng, hệ thống tạo một bản ghi metadata gồm đường dẫn lưu trữ (`path`), liên kết nguồn (`source_url`), tên người dùng (`username`), URL truy xuất từ MinIO (`storage_url`), loại nội dung (`content_type`), kích thước tệp (`size_bytes`), và `file_id`. Bản ghi này được đóng gói thành một dòng trong ` DataFrame`, giúp các khối xử lý tiếp theo có thể truy cập, kiểm tra, hoặc phân tích dữ liệu một cách dễ dàng..



```
class                    (                ,                ):
    """Download a file from Google Drive into MinIO and expose its location as a Dataframe.

    Supports both a full Google Drive URL or a file_id. Handles the confirmation token flow
    required for large files.
    """

    category =            .INGESTION
    description = "Download a file from Google Drive to MinIO storage"
    img_url = "/static/icons/ingestion/google-drive.png"

    def __init__(…

    @
    def from_config(cls, config:     [   , Any]) -> "                    ":…

    @
    def show_config(cls) ->     [   , Any]:…

    @
    def _extract_file_id(url_or_id:         [   ], fallback_id:         [   ]) ->         [   ]:…

    @
    def _content_disposition_filename(resp:         ) ->         [   ]:…

    @
    def _download_from_gdrive(file_id:    , timeout:    ) ->         :…

    def execute(self, data: Any = None) ->    .         :
        # 1) Decide the file_id
        file_id = self._extract_file_id(self.parameters.get("url"), self.parameters.get("file_id"))
        if not file_id:
            raise         ("GoogleDriveDownloadBlock: 'file_id' or valid Google Drive 'url' is required")

        # 2) Download (stream)
        resp = self._download_from_gdrive(file_id, timeout=    (self.parameters.get("timeout", 60)))
        content_type = resp.headers.get("content-type", "application/octet-stream")

        # 3) Determine filename
        if self.filename is None or self.filename == "":
            name = self._content_disposition_filename(resp) or f"{file_id}.bin"
            self.filename = name

        # 4) Read bytes (stream to memory; alternative: stream to temp file if needed)
        payload = resp.content
        stream =         (payload)
        stream.seek(0)

        # 5) Save to MinIO
        username = self.parameters.get("minIOusername") or "uet"
        object_key =    .path.join(self.parameters["output_path"], self.filename)
        saved_url = self.save_file(stream, username, object_key, content_type=content_type)

        record = {
            "path": object_key,
            "source_url": f"https://drive.google.com/file/d/{file_id}/view",
            "username": username,
            "storage_url": saved_url,
            "content_type": content_type,
            "size_bytes": len(payload),
            "file_id": file_id,
        }
        return     .         ([record])
```

Hình 3.22: Cài đặt khối kéo dữ liệu từ google drive trong backend

Khối tải dữ liệu từ Google Drive mang lại một cách tiếp cận đơn giản nhưng hiệu quả để ingest dữ liệu phi cấu trúc từ các nguồn chia sẻ phổ biến. Người dùng

không cần viết mã thủ công hay xử lý xác thực phức tạp, chỉ cần cấu hình vài trường cơ bản trong giao diện kéo-thả. Tính năng này đặc biệt hữu ích trong các trường hợp ingest dữ liệu huấn luyện từ cộng đồng, kho học thuật, hoặc các đối tác chia sẻ qua Drive.

Mô-đun giải nén tệp dạng zip

ZIP Extract là một khối phụ trợ chuyên trách giải nén các tệp nén định dạng zip đã được thu thập vào kho object storage. Khối này nhận một file nén (theo đường dẫn object hoặc stream), giải nén các file con, lưu từng file con vào thư mục đầu ra trên storage, sinh metadata cho mỗi file con và trả về một bảng (DataFrame) liệt kê các file đã giải nén cùng thuộc tính cần thiết cho các mô-đun tiếp theo.

```
class                (                ,                ):
    category =              .IO
    description = "Extract .zip files into folders in MinIO storage"
    img_url = "/static/icons/io/unzip.png"
    def __init__(self, output_folder_path:      = "", input_path:          [   ] = "", minIOusername:          [   ] = "uet",):

    def execute(self, df:          [  .          ] = None, username:      = "default_user") ->   .          :
        """
        Extract .zip files from either input_path (standalone) or DataFrame (pipeline).

        Args:
            df (pd.DataFrame): DataFrame with a column `path` containing paths to .zip files in MinIO storage.

        Returns:
            pd.DataFrame: DataFrame containing `zip_path` and `element` columns.
        """
        username = self.parameters.get("minIOusername", "uet")
        extracted_files = []
        # Sanitize input_path: treat placeholder and blanks as empty
        raw_input_path = self.parameters.get("input_path", "") or ""
        placeholder = "put empty if it follows a block"
        if isinstance(raw_input_path,     ) and raw_input_path.strip().lower() in {"", "none", "null", placeholder.lower()}:
            input_path = ""
        else:
            input_path = raw_input_path.strip()

        zip_paths:     [   ] = []
        if input_path:
            if not input_path.lower().endswith(".zip"):
                raise           (f"UnzipBlock: input_path is not a .zip file: {input_path}")
            zip_paths = [input_path]
        elif df is not None:
            col = "path" if "path" in df.columns else ("zip_path" if "zip_path" in df.columns else None)
            if col is None:
                raise           ("UnzipBlock: No input_path provided and DataFrame missing 'path' or 'zip_path' column.")
            zip_paths = [    (p) for p in df[col].dropna().tolist() if isinstance(p,     ) and p.lower().endswith(".zip")]
            if not zip_paths:
                raise           ("UnzipBlock: No .zip paths found in the provided DataFrame.")
        else:
            raise          ("UnzipBlock: No input_path provided and no DataFrame with 'path' column.")

        for zip_path in zip_paths:
            zip_filename = zip_path.split("/")[-1].split(".")[0]
            # If no explicit output folder, default to a folder named after the zip (without extension)
            output_folder = self.parameters.get("output_folder_path") or f"{zip_filename}/"
            zip_stream =          (self.get_file(username, zip_path))
            with          .          (zip_stream, "r") as zf:
                for name in zf.namelist():
                    # Normalize output path: avoid double slashes and trailing slash
                    output_folder_norm = output_folder.rstrip("/")
                    name_norm = name.lstrip("/")
                    output_path = f"{output_folder_norm}/{name_norm}"
                    file_stream =          (zf.read(name))
                    self.save_file(file_stream, username, output_path)
                    extracted_files.append({"zip_path": zip_path, "element": output_path})
        return   .          (extracted_files)
```

Hình 3.23: Cài đặt tác vụ giải nén file zip trong backend

Cài đặt trong backend: Quy trình giải nén các tệp ZIP được lưu trữ trên MinIO hỗ trợ hai dạng đầu vào: nhận trực tiếp từ cấu hình hoặc từ một nguồn dữ liệu có chứa cột đường dẫn. Hàm xử lý bắt đầu bằng việc chuẩn hóa tham số đầu vào và kiểm tra tính hợp lệ của đường dẫn tới tệp ZIP; nếu không tìm thấy đường dẫn hợp lệ, hệ thống sẽ phát sinh lỗi cấu hình. Đối với mỗi tệp ZIP, hệ thống tiến hành đọc dữ liệu nhị phân từ MinIO, sau đó nạp vào bộ nhớ và mở archive để duyệt qua từng thành phần bên trong. Mỗi mục (entry) được chuẩn hóa đường dẫn nhằm tránh các lỗi như trùng dấu "/" hoặc các rủi ro cơ bản liên quan đến path traversal. Nội dung của từng tệp con được đọc và lưu trở lại hệ thống object storage, sử dụng đường dẫn đích tương ứng (hoặc mặc định là thư mục được đặt theo tên tệp ZIP ban đầu). Trong quá trình xử lý, hệ thống đồng thời thu thập một số metadata cơ bản, chẳng hạn như tên tệp và kích thước, cho từng tệp được giải nén. Kết quả cuối cùng là một cấu trúc dữ liệu đầu ra chứa danh sách các tệp con cùng với đường dẫn lưu trữ của chúng, đóng vai trò làm đầu vào tiêu chuẩn cho các bước xử lý tiếp theo trong pipeline. Thiết kế của quy trình ưu tiên xử lý archive nhỏ theo cơ chế stream trong bộ nhớ, đảm bảo tính tương đương ở mức ghi tệp (các trường hợp ghi trùng có thể được xử lý ở tầng lưu trữ). Đồng thời, kiến trúc này cũng cho phép mở rộng dễ dàng, chẳng hạn như bổ sung kiểm tra kích thước tệp, giới hạn định dạng, tích hợp quét mã độc hoặc chuyển các tệp lỗi vào vùng quarantine.

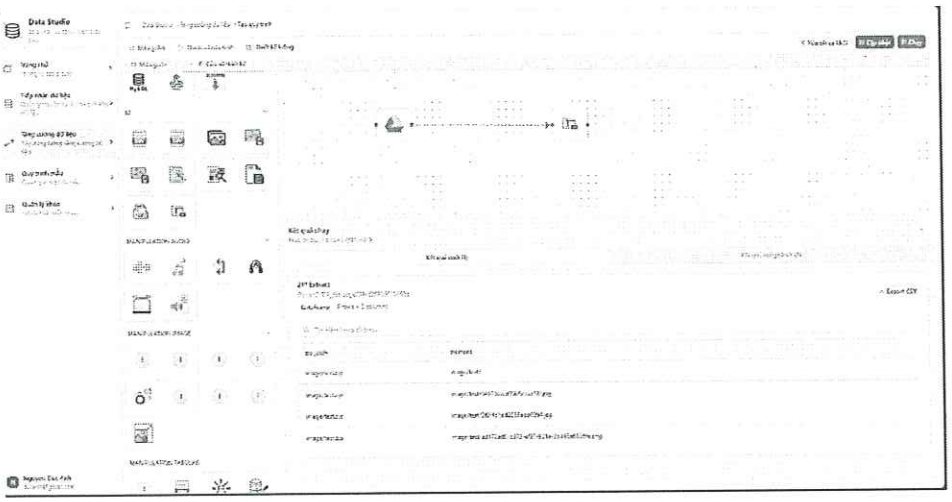
Ví dụ: Người dùng muốn tải một tệp zip chứa folder ảnh từ google drive về

Mục tiêu: Thử nghiệm khối tác vụ google drive và khối giải nén.

Cấu hình thực hiện: một file zip gồm một số ảnh được tải nên google drive

Luồng thực hiện:

1. Người dùng cài đặt khối google drive download và khối giải nén trong giao diện canvas của hệ thống.
2. Hệ thống thực hiện các tác vụ theo cài đặt của người dùng
3. Kết quả được trả về dưới dạng bảng.



Hình 3.24: Cài đặt và kết quả

Kết quả:

- Trạng thái: Thành công
- Số bản ghi: 3 file ảnh được giải nén

3.3. Kiến trúc và cài đặt các mô-đun quản lý mô hình TTNT

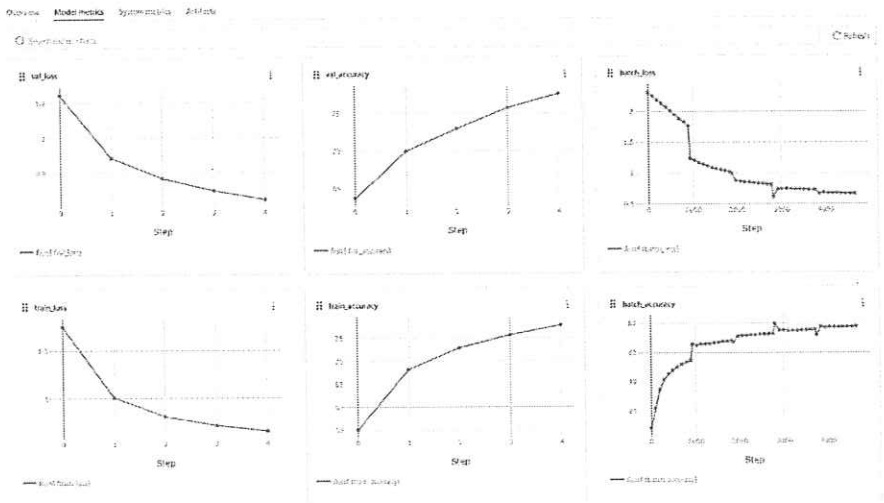
3.3.1. Xây dựng mô-đun quản lý cấu hình huấn luyện:

Cấu hình huấn luyện bao gồm các nhóm thông tin chính như: siêu tham số huấn luyện, cấu hình mô hình và các tham số liên quan đến dữ liệu. Khi một phiên huấn luyện được khởi tạo, toàn bộ cấu hình này được gắn với phiên huấn luyện tương ứng để phục vụ việc truy vết và so sánh kết quả.

Sau khi cấu hình huấn luyện được tiếp nhận, mô-đun huấn luyện chuyển sang giai đoạn chuẩn bị dữ liệu và môi trường tính toán. Đây là bước quan trọng nhằm đảm bảo rằng quá trình huấn luyện được thực hiện trong điều kiện phù hợp với cấu hình đã khai báo. [22], [23]

Song song với việc chuẩn bị dữ liệu, môi trường tính toán cũng được thiết lập phù hợp với yêu cầu của từng phiên huấn luyện. Các công việc như lựa chọn tài nguyên tính toán, cấu hình tham số môi trường và kiểm tra khả năng truy cập dữ liệu đều được thực hiện trước khi bắt đầu huấn luyện. Trong mô-đun, những thông tin liên quan đến dữ liệu và môi trường này được lưu trữ dưới dạng tệp cấu hình hoặc metadata đi kèm.

Trong bản demo thực nghiệm, mô-đun huấn luyện được sử dụng để thực hiện nhiều lần huấn luyện với các cấu hình khác nhau. Mỗi lần chạy tương ứng với một phiên huấn luyện độc lập, bao gồm toàn bộ quy trình từ khởi tạo mô hình đến khi hoàn thành vòng lặp huấn luyện. Cách tổ chức này giúp đảm bảo tính nhất quán và tránh xung đột giữa các lần thực thi.



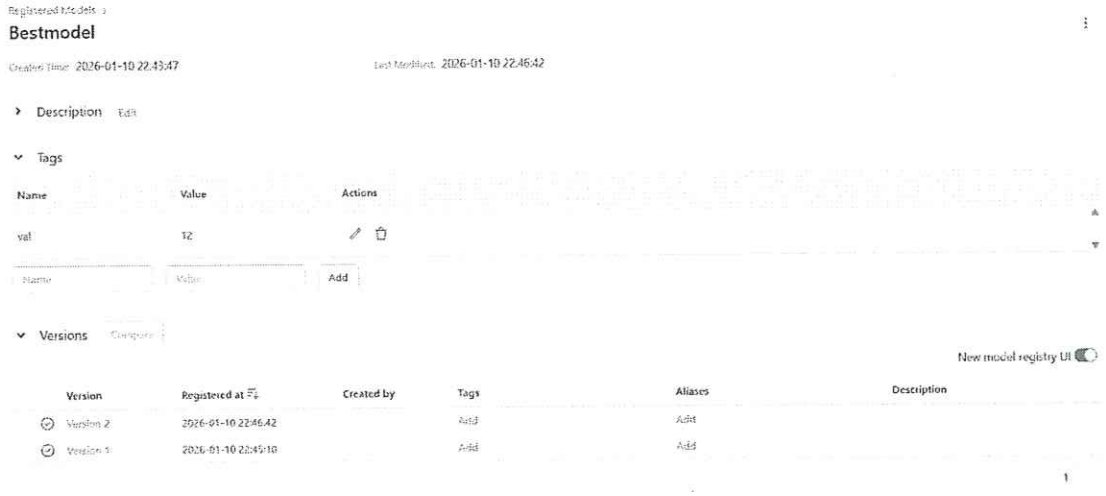
Hình 3.25: Kết quả thực thi huấn luyện mô hình

3.3.2. Xây dựng mô-đun đánh giá huấn luyện:

Mục tiêu của việc thiết kế mô-đun đánh giá không chỉ dừng lại ở việc xác định các thành phần chức năng riêng lẻ, mà còn hướng tới xây dựng một hệ thống hoàn chỉnh, có vòng đời rõ ràng và khả năng tái sử dụng cao.

Cụ thể, mô-đun đánh giá cần đáp ứng linh hoạt nhiều kịch bản sử dụng. Một mặt, nó phải cho phép đánh giá và so sánh nhiều mô hình khác nhau trên cùng một tập dữ liệu nhằm lựa chọn mô hình tối ưu. Mặt khác, mô-đun cũng cần hỗ trợ đánh giá một mô hình trên nhiều tập dữ liệu khác nhau, chẳng hạn như tập kiểm tra, tập xác thực hoặc các tập dữ liệu được thu thập theo thời gian, trong khi vẫn đảm bảo tính nhất quán của quy trình. Sự nhất quán này đóng vai trò quan trọng để đảm bảo rằng kết quả đánh giá phản ánh chính xác năng lực của mô hình, thay vì bị ảnh hưởng bởi sự thay đổi trong cách thức thực hiện. [20], [21]

Trong quá trình triển khai, cấu hình đánh giá được xây dựng như một thực thể độc lập, bao gồm thông tin về dữ liệu đánh giá, danh sách các chỉ số cần tính toán và các tham số liên quan đến quá trình thực thi. Việc tách biệt cấu hình khỏi logic đánh giá giúp mô-đun có thể dễ dàng tái sử dụng cho nhiều kịch bản khác nhau mà không cần chỉnh sửa mã nguồn.



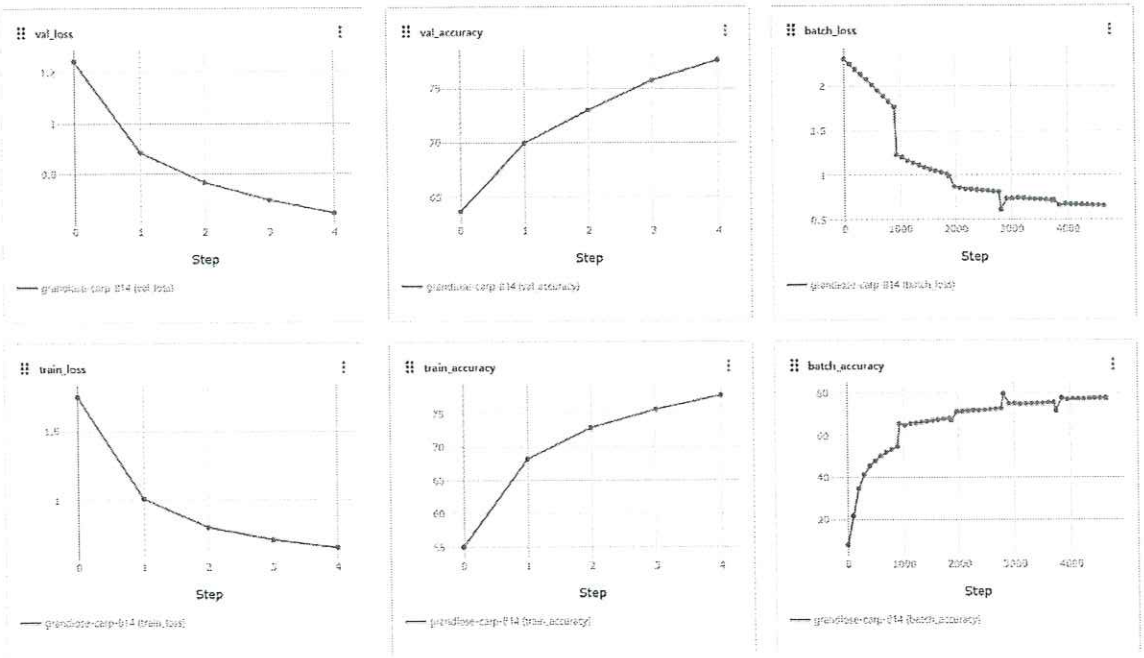
Hình 3.26: Thông tin mô hình TTNT dùng để đánh giá.

Trong suốt quá trình thực thi đánh giá, mô-đun được lập trình để ghi nhận các chỉ số đánh giá phản ánh trực tiếp hiệu năng của mô hình trên dữ liệu đánh giá. Các chỉ số này được xác định trước trong cấu hình đánh giá và có thể bao gồm các metric phổ biến hoặc các metric đặc thù của từng bài toán TTNT.

Việc theo dõi và ghi nhận kết quả đánh giá một cách có hệ thống giúp mô-đun đánh giá trở thành một thành phần quan trọng trong việc lựa chọn mô hình, kiểm soát chất lượng và hỗ trợ ra quyết định trong hệ thống TTNT.

Model metrics (6)	
val_loss	0.6477705836296082
val_accuracy	77.57
batch_loss	0.6595446956276761
train_loss	0.6583827559564159
train_accuracy	77.745
batch_accuracy	77.70359322974473

Hình 3.27: Kết quả ghi nhận kết quả đánh giá mô hình TTNT

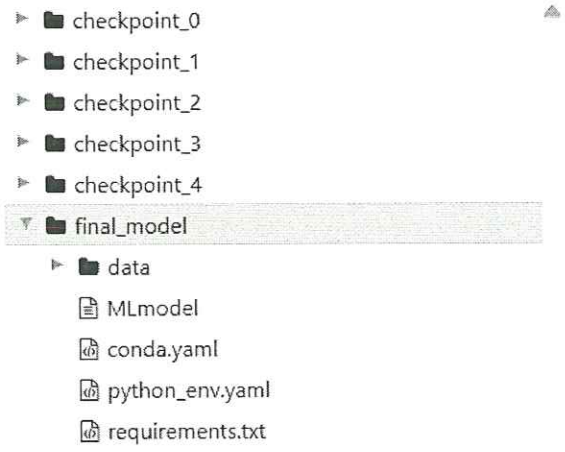


Hình 3.28: Hiển thị kết quả đánh giá mô hình TTNT

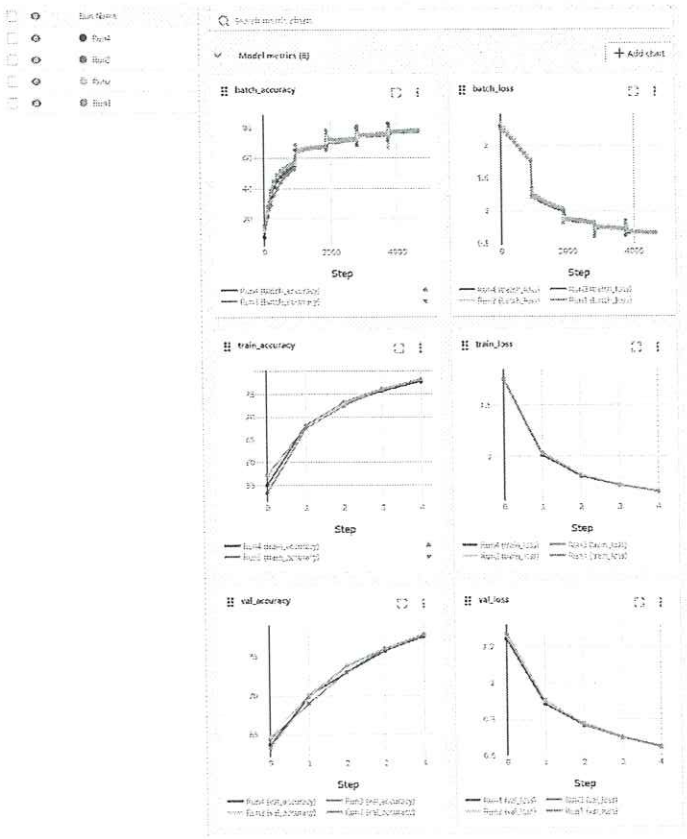
3.3.3. Xây dựng mô-đun tinh chỉnh mô hình huấn luyện :

Mô-đun tinh chỉnh được hiện thực với thành phần đầu tiên là khối định nghĩa và quản lý không gian siêu tham số. Thành phần này đóng vai trò chuẩn hóa toàn bộ tập các siêu tham số cần tối ưu, bao gồm tên tham số, kiểu dữ liệu và miền giá trị tương ứng. Việc hiện thực không gian siêu tham số như một cấu trúc dữ liệu độc lập giúp mô-đun tinh chỉnh tách biệt hoàn toàn khỏi logic huấn luyện và đánh giá mô hình. [20], [21]

Trong mô-đun tinh chỉnh, việc ghi nhận kết quả được hiện thực để phục vụ trực tiếp cho quá trình ra quyết định tối ưu hóa. Mỗi trial được ghi nhận đầy đủ siêu tham số, chỉ số đánh giá và các artifact liên quan.



Hình 3.29: Danh sách các checkpoint



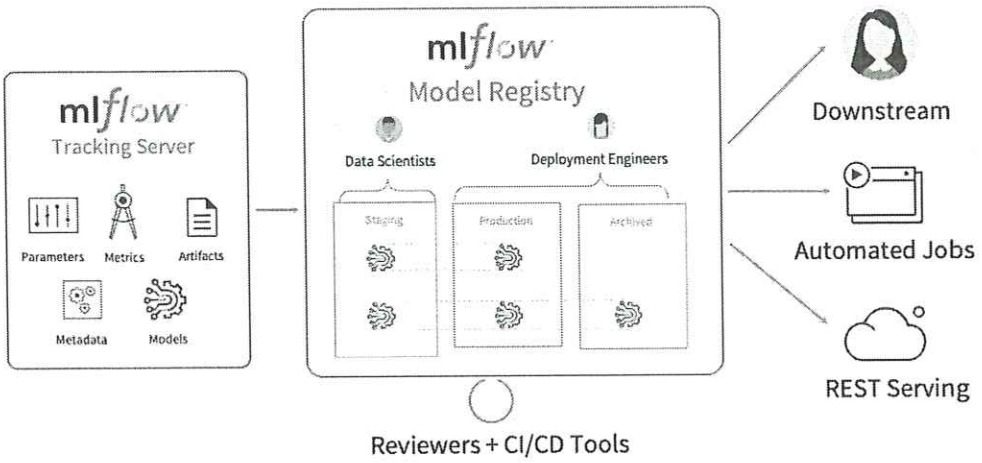
Hình 3.30: Ghi nhận kết quả để chọn bộ tham số tối ưu

3.3.4. Xây dựng mô-đun quản lý danh mục mô hình TTNT

MLFlow Registry:

MLflow Model Registry là một thành phần quan trọng của nền tảng MLflow, được thiết kế để quản lý vòng đời của mô hình Machine Learning một cách tập trung và có kiểm soát.

MLflow Model Registry là một hệ thống quản lý mô hình tập trung, cho phép lưu trữ và tổ chức các mô hình Machine Learning đã được huấn luyện và log thông qua MLflow. Thay vì quản lý mô hình dưới dạng các tệp rời rạc, Model Registry cung cấp một lớp quản lý logic, giúp các mô hình được quản lý theo tên, phiên bản và trạng thái sử dụng.



Hình 3.31: MLFlow Model Registry

Model Registry không lưu trực tiếp mô hình, mà quản lý metadata và tham chiếu đến vị trí lưu trữ mô hình (artifact store). Nhờ đó, hệ thống đảm bảo khả năng mở rộng, dễ tích hợp và thuận tiện cho việc truy vết.

Model Registry đóng vai trò như một kho lưu trữ mô hình chuẩn hóa, giúp các nhóm dữ liệu phối hợp hiệu quả trong quy trình MLOPS. [22] [23]

Trong kiến trúc MLOPS hiện đại, Model Registry giúp giải quyết các vấn đề:

- Quản lý nhiều phiên bản mô hình song song
- Kiểm soát việc chuyển mô hình từ thử nghiệm sang sản xuất
- Đảm bảo tính truy vết và khả năng tái lập
- Chuẩn hóa quy trình CI/CD cho Machine Learning

Các khái niệm chính trong MLflow Model Registry

- Registered Model: Là một tên logic đại diện cho một dòng mô hình. Ví dụ: fraud_detection_model, image_classifier.
- Model Version: Mỗi lần log một mô hình mới vào registry sẽ tạo ra một phiên bản. Phiên bản được đánh số tự động: v1, v2, v3, …
- Model Stage (Trạng thái mô hình). MLflow hỗ trợ các stage tiêu chuẩn
    - None: Mô hình mới đăng ký
    - Staging: Mô hình đang kiểm thử
    - Production: Mô hình đang phục vụ thực tế
    - Archived: Mô hình đã ngừng sử dụng

Việc chuyển stage giúp kiểm soát luồng phát hành mô hình.

MLflow Model Registry thường được tích hợp với: [22] [23]

- **MLflow Tracking**: lưu experiment, metrics, artifacts
- **MinIO / S3**: lưu trữ model artifacts
- **CI/CD (GitLab CI, GitHub Actions)**: tự động promote model
- **KServe / Seldon / BentoML**: triển khai model serving
- **Argo CD / Kubernetes**: quản lý hạ tầng triển khai

Lưu trữ mô hình với MinIO :

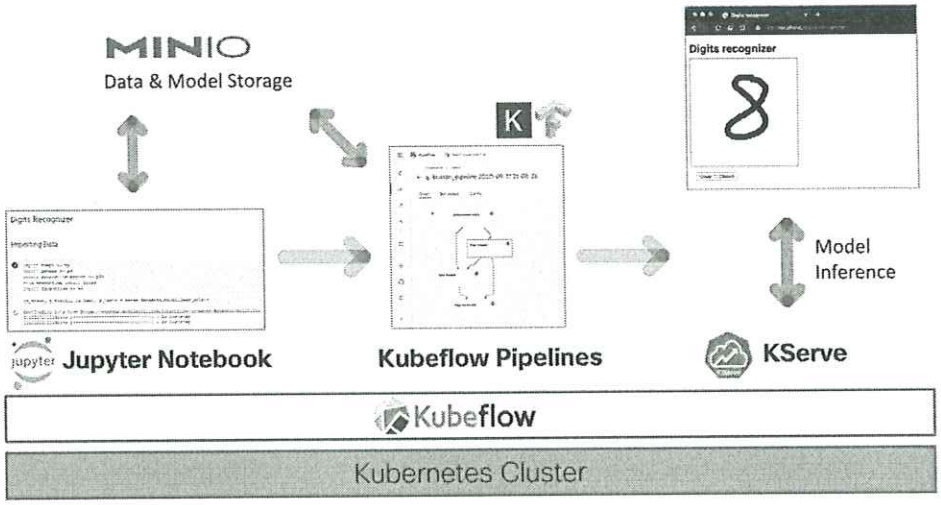
MinIO là một nền tảng lưu trữ đối tượng mã nguồn mở, được xây dựng bằng ngôn ngữ Go và hoàn toàn tương thích với giao thức Amazon S3. Hệ thống này có tính linh hoạt cao, có thể triển khai trên nhiều môi trường khác nhau như máy chủ vật lý, Docker hoặc Kubernetes. MinIO được thiết kế để xử lý và lưu trữ các loại dữ liệu phi cấu trúc với dung lượng lớn, chẳng hạn như hình ảnh, video, dữ liệu cảm biến, log, và đặc biệt là dữ liệu phục vụ huấn luyện mô hình học máy. Nhờ kiến trúc tinh gọn nhưng hiệu suất cao, MinIO đã trở thành một lựa chọn phổ biến trong các hệ thống MLOPS hiện nay. [24]

Một ưu điểm đáng chú ý của MinIO là khả năng xử lý dữ liệu với tốc độ rất cao, có thể đạt tới hàng trăm GB mỗi giây khi đọc/ghi dữ liệu lớn. Do được phát triển bằng Go và tối ưu cho xử lý song song, hệ thống vẫn đảm bảo hoạt động ổn định ngay cả khi khối lượng dữ liệu tăng mạnh. Ngoài ra, việc triển khai MinIO khá đơn giản, chỉ cần một vài lệnh cơ bản là có thể thiết lập và vận hành trên cả môi trường cục bộ lẫn hệ thống phân tán như Kubernetes. Bên cạnh đó, nhờ khả năng tương thích hoàn toàn với API của Amazon S3, các ứng dụng hiện có có thể dễ dàng chuyển đổi sang MinIO mà không cần chỉnh sửa mã nguồn, giúp tiết kiệm đáng kể thời gian và chi phí tích hợp. [24]

Về mặt bảo mật, MinIO hỗ trợ đầy đủ các cơ chế mã hóa dữ liệu cả trong quá trình truyền tải (encryption-in-transit) và khi lưu trữ (encryption-at-rest). Đồng thời, cơ chế erasure coding giúp giảm thiểu rủi ro mất dữ liệu, trong khi tính năng replication đảm bảo tính sẵn sàng cao và khả năng phục hồi khi xảy ra sự cố. Là một giải pháp mã nguồn mở, MinIO cho phép tổ chức toàn quyền kiểm soát dữ liệu, không phụ thuộc vào các nhà cung cấp dịch vụ đám mây, điều này đặc biệt quan trọng đối với các hệ thống yêu cầu bảo mật cao hoặc tuân thủ nghiêm ngặt các quy định.

Trong hệ sinh thái MLOPS, MinIO đóng vai trò như một kho lưu trữ trung tâm và nơi quản lý các tài sản xuyên suốt vòng đời của mô hình học máy. Nó có thể được sử dụng để lưu trữ dữ liệu huấn luyện, quản lý phiên bản dữ liệu thông qua DVC, lưu trữ đặc trưng trong feature store, cũng như lưu các mô hình đã huấn luyện bằng

MLflow. Ngoài ra, MinIO còn lưu trữ các kết quả trung gian, checkpoint, log và đầu ra của từng bước trong pipeline. Với khả năng mở rộng tốt và tích hợp thuận tiện, MinIO có thể kết hợp hiệu quả với các công cụ như Kubeflow, Airflow, Prefect hay MLflow, góp phần xây dựng một hạ tầng MLOPS đồng bộ, linh hoạt và hiệu quả. [26]



Hình 3.32: Nền tảng lữu trữ MinIO

Về mặt cấu trúc, Hệ thống tích hợp MLflow và MinIO thường được cấu thành từ bốn thành phần chính:

- MLflow Tracking Server: đảm nhiệm việc ghi nhận, theo dõi và quản lý các thí nghiệm, bao gồm siêu dữ liệu và kết quả thực nghiệm.
- MLflow UI: cung cấp giao diện trực quan, giúp người dùng dễ dàng theo dõi, so sánh và phân tích các thí nghiệm.
- MinIO Server: đóng vai trò lưu trữ các artifact như mô hình, file kết quả, log và các dữ liệu đầu ra khác.
- Database backend (tùy chọn): sử dụng các hệ quản trị cơ sở dữ liệu như MySQL hoặc PostgreSQL để lưu trữ siêu dữ liệu của MLflow, thay thế cho SQLite mặc định nhằm tăng khả năng mở rộng và hiệu năng.

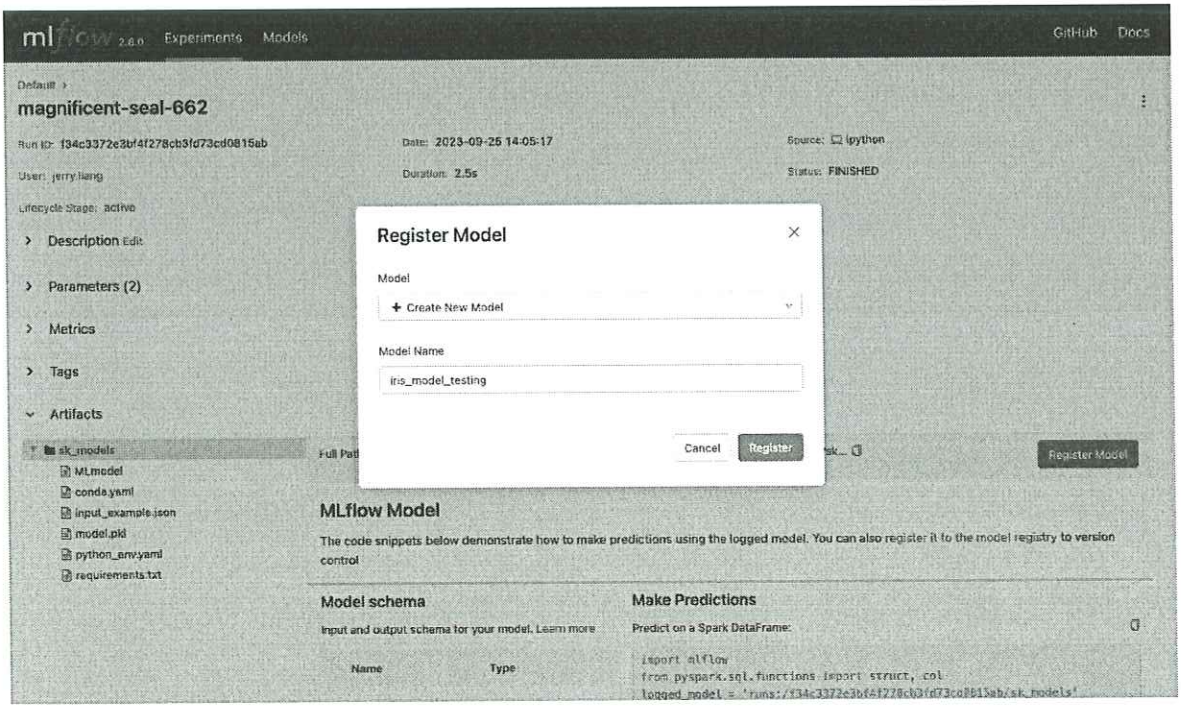
Luồng hoạt động của hệ thống có thể mô tả như sau: mã huấn luyện (training code) tương tác với MLflow thông qua Tracking API; trong đó, các artifact sẽ được lưu trữ trên MinIO, còn siêu dữ liệu sẽ được ghi vào cơ sở dữ liệu backend. [25]

Quy trình đăng ký mô hình:

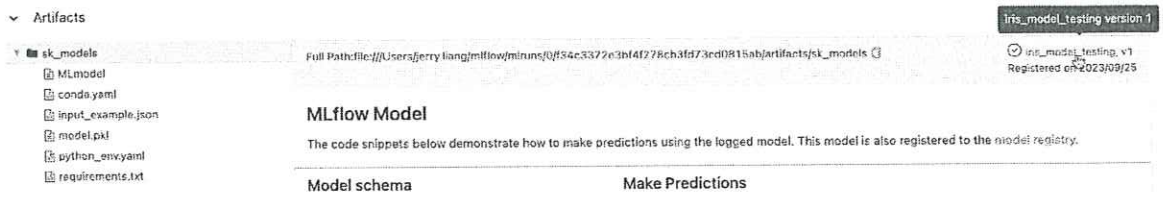
Người sử dụng lựa chọn mô hình mẫu tốt trong các thí nghiệm đã tiến hành

Hình 3.33: Giao diện chọn mô hình TTNT mẫu

Sau đó người sử dụng sẽ đăng ký tên mô hình được lựa chọn từ thí nghiệm của mình



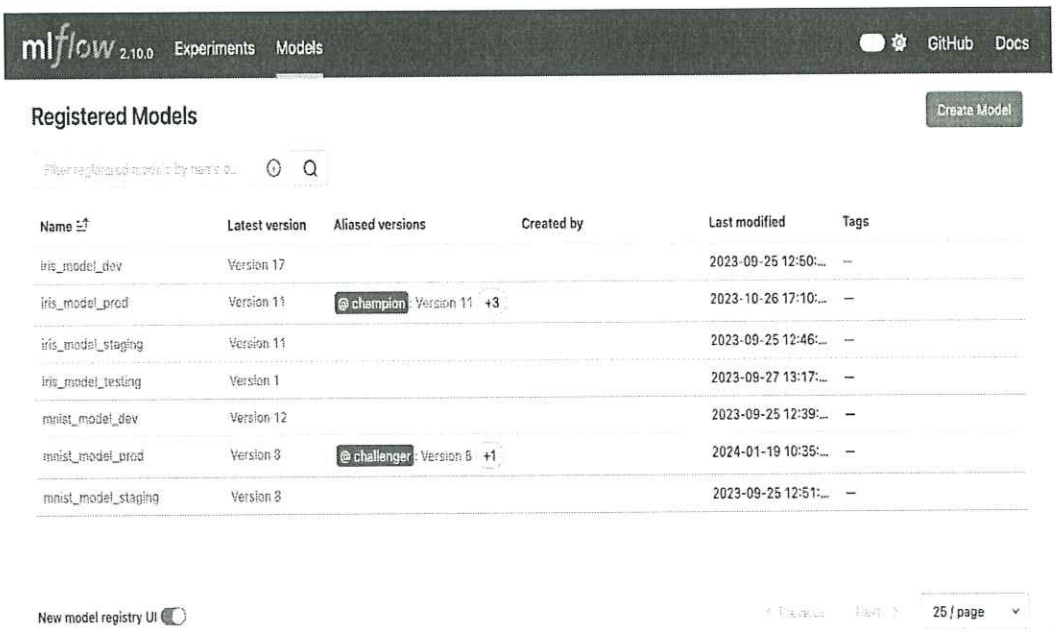
Hình 3.34: Giao diện đăng ký mô hình TTNT mẫu



Hình 3.35: Giao diện xem thông tin về mô hình TTNT

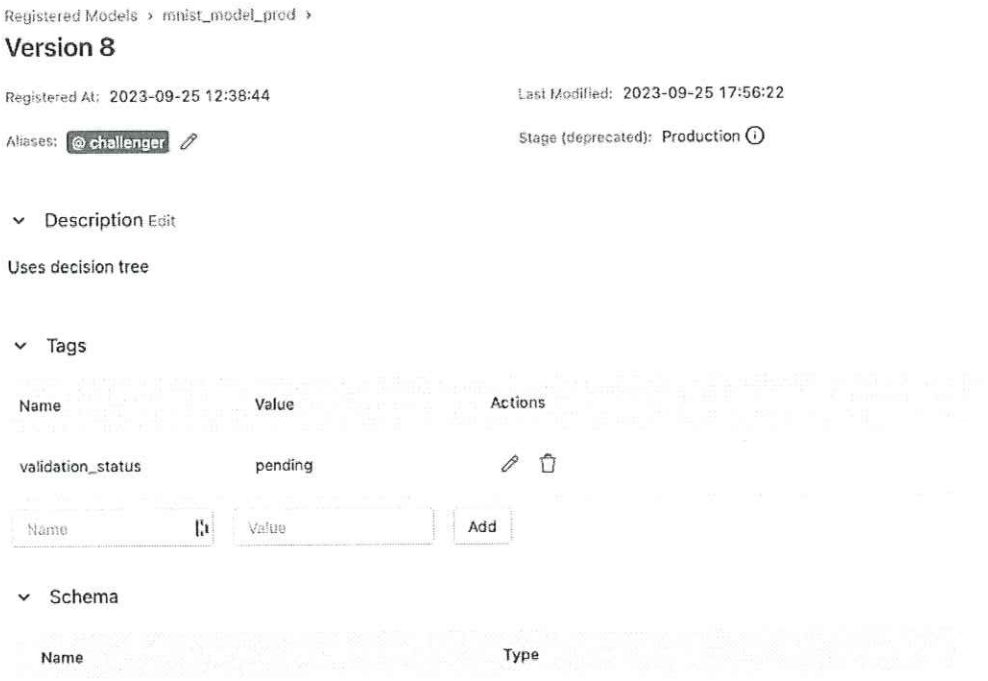
Quản lý danh sách mô hình đăng ký

Người sử dụng có thể xem danh sách các mô hình mẫu tại tab Models của MLFlow



Hình 3.36: Giao diện liệt kê các mô hình TTNT mẫu

Người sử dụng có thể xem các phiêm bản khác nhau của mô hình cũng như chuyển trạng thái xây dựng cho mô hình lưu trong model registry.



Hình 3.37: Thông tin chi tiết phiên bản TTNT mẫu

3.4. Định hướng triển khai mô hình, giám sát và pipeline huấn luyện lại

Một hệ thống MLOPS hoàn chỉnh, ngoài các phân hệ đã triển khai ở trên, còn bao gồm các giai đoạn triển khai mô hình (model serving), giám sát mô hình trong vận hành (monitoring) và huấn luyện lại (retraining).

3.4.1. Triển khai mô hình

Mô hình sau khi được chuyển sang trạng thái Production trong MLflow Model Registry sẽ được triển khai thành dịch vụ dự đoán thông qua các nền tảng mã nguồn mở như KServe, Seldon Core hoặc BentoML. Các nền tảng này đọc trực tiếp artifact mô hình từ MinIO theo tham chiếu do Model Registry cung cấp, đóng gói mô hình thành dịch vụ cung cấp giao diện REST/gRPC và được container hóa bằng Docker, triển khai trên Kubernetes để có khả năng mở rộng theo tải. Nhờ đó, việc phát hành một phiên bản mô hình mới chỉ là thao tác chuyển trạng thái trong Model Registry, không đòi hỏi thay đổi mã nguồn của dịch vụ.

3.4.2. Giám sát và phát hiện sai lệch

Sau khi mô hình được đưa vào vận hành, hệ thống cần giám sát đồng thời hai nhóm chỉ số: (i) chỉ số hạ tầng và dịch vụ như độ trễ, thông lượng, tỷ lệ lỗi, có thể thu thập và trực quan hóa bằng bộ công cụ Prometheus – Grafana; (ii) chỉ số chất lượng dữ liệu và mô hình, trong đó quan trọng nhất là phát hiện sai lệch dữ liệu (data drift) và sai lệch mô hình (model drift). Định hướng của luận văn là sử dụng thư viện mã nguồn mở Evidently AI để định kỳ so sánh phân phối của dữ liệu đầu vào thực tế với

phân phối của tập dữ liệu huấn luyện đã lưu trong hồ dữ liệu; khi mức sai lệch vượt ngưỡng cấu hình, hệ thống phát cảnh báo và kích hoạt quy trình huấn luyện lại.

3.4.3. Pipeline huấn luyện lại và tích hợp CI/CD

Pipeline huấn luyện lại được kích hoạt tự động theo hai cơ chế: theo chu kỳ định trước hoặc theo sự kiện khi phát hiện sai lệch vượt ngưỡng. Khi được kích hoạt, một công cụ điều phối luồng công việc như Apache Airflow hoặc Argo Workflows sẽ tái sử dụng chính các mô-đun đã xây dựng trong luận văn — mô-đun tải dữ liệu, mô-đun quản lý cấu hình huấn luyện, mô-đun đánh giá — để thu thập dữ liệu mới, huấn luyện lại và đánh giá mô hình. Quy trình CI/CD đảm nhận việc kiểm thử tự động và chỉ cho phép đăng ký phiên bản mô hình mới vào Model Registry khi kết quả đánh giá vượt phiên bản đang vận hành. Chuỗi khép kín "giám sát – phát hiện sai lệch – huấn luyện lại – phát hành" hoàn thiện vòng đời MLOPS như minh họa tại Hình 3.1.

Việc triển khai thực nghiệm đầy đủ các thành phần nêu trên đòi hỏi hạ tầng vận hành và dữ liệu thực tế vượt quá điều kiện của luận văn, do đó được xác định là hướng phát triển tiếp theo (trình bày tại phần Kết luận và hướng phát triển).

3.5. Kết luận chương

Chương 3 đã trình bày quá trình thiết kế, xây dựng và triển khai hệ thống MLOPS dựa trên các nền tảng mã nguồn mở nhằm hỗ trợ quản lý toàn bộ vòng đời của mô hình học máy. Hệ thống được xây dựng theo kiến trúc module, bao gồm các thành phần Data Ingestion, Data Processing, Model Training, Model Evaluation, Hyperparameter Tuning, Model Registry và lưu trữ dữ liệu bằng MinIO.

Kết quả thực nghiệm cho thấy hệ thống có khả năng tiếp nhận nhiều loại dữ liệu khác nhau, hỗ trợ tự động hóa quy trình huấn luyện, quản lý phiên bản mô hình, lưu trữ artifact và theo dõi các thí nghiệm thông qua MLflow.

Những kết quả đạt được trong chương này là cơ sở để đưa ra các kết luận chung của luận văn và định hướng cho các nghiên cứu tiếp theo nhằm hoàn thiện hệ thống theo hướng mở rộng chức năng, nâng cao khả năng tự động hóa và hỗ trợ nhiều loại mô hình TTNT hơn trong tương lai

KẾT LUẬN VÀ HƯỚNG PHÁT TRIỂN

Luận văn đã tập trung nghiên cứu và xây dựng một cách tiếp cận tích hợp các nền tảng mã nguồn mở nhằm hỗ trợ hiệu quả cho quá trình phát triển mô hình học máy trong bối cảnh MLOPS. Thông qua việc hệ thống hóa quy trình từ thu thập dữ liệu đến xử lý và tăng cường dữ liệu, luận văn đã làm rõ vai trò của từng thành phần trong pipeline dữ liệu. Đồng thời, các loại dữ liệu đầu vào như dữ liệu dạng bảng, tuần tự, streaming, sự kiện và dữ liệu dạng tệp đã được phân tích và đề xuất phương pháp xử lý phù hợp. Việc áp dụng các kỹ thuật đã góp phần nâng cao chất lượng dữ liệu, từ đó cải thiện hiệu quả của mô hình khi thực nghiệm trên các bộ dữ liệu cụ thể.

Bên cạnh những kết quả đạt được, luận văn vẫn còn một số hạn chế nhất định. Phạm vi nghiên cứu chủ yếu tập trung vào một số nền tảng mã nguồn mở phổ biến, chưa bao quát hết các công cụ đa dạng trong hệ sinh thái MLOPS hiện nay. Quy mô thực nghiệm còn hạn chế, chủ yếu dựa trên các bộ dữ liệu mẫu nên chưa phản ánh đầy đủ các bài toán dữ liệu lớn trong thực tế. Ngoài ra, một số khía cạnh quan trọng như triển khai CI/CD cho mô hình, giám sát hệ thống (monitoring), phát hiện sai lệch dữ liệu (data/model drift) và tự động huấn luyện lại (retraining) vẫn chưa được nghiên cứu và triển khai đầy đủ. Việc đánh giá hệ thống cũng mới dừng lại ở chất lượng mô hình, chưa xem xét sâu đến hiệu năng, khả năng mở rộng và chi phí vận hành.

Trong tương lai, nghiên cứu có thể được mở rộng theo nhiều hướng nhằm hoàn thiện và nâng cao giá trị thực tiễn. Cụ thể, cần tích hợp thêm các công cụ MLOPS hiện đại để xây dựng pipeline khép kín từ thu thập, xử lý, huấn luyện đến triển khai và giám sát mô hình. Việc triển khai hệ thống trên môi trường dữ liệu lớn, kết hợp với các công nghệ xử lý phân tán và streaming thời gian thực, cũng là một hướng đi quan trọng. Cuối cùng, việc mở rộng ứng dụng sang nhiều lĩnh vực khác nhau và tối ưu hóa hiệu năng triển khai trên nền tảng cloud sẽ giúp đánh giá toàn diện hơn tính khả thi và hiệu quả của giải pháp đề xuất.

TÀI LIỆU THAM KHẢO

[1]. Nogare, D.; Silveira, I.F. Experimentation, deployment and monitoring Machine Learning models: Approaches for applying MLOps. arXiv 2024, arXiv:2408.11112.

[2]. Rivas, J. M., et al. (2024). An analysis of the challenges in the adoption of MLOps. Journal of King Saud University - Computer and Information Sciences,36(10). https://www.sciencedirect.com/science/article/pii/S2444569X24001768. Ngày truy cập: 13/11/2025

[3]. Kelvins. (2024). Awesome MLOps: A Curated List of Awesome MLOps Tools. GitHub Repository. https://github.com/kelvins/awesome-mlops. Ngày truy cập: 18/11/2025

[4]. Wang, Y., et al. (2025). Transitioning from MLOps to LLMOps: Navigating the Unique Challenges of Large Language Models. Information, 16(2), 87. https://www.mdpi.com/2078-2489/16/2/87. Ngày truy cập: 04/12/2025

[5]. Google Cloud. (2024). MLOps: Continuous delivery and automation pipelines in machine learning. https://cloud.google.com/architecture/mlops-continuous-delivery-and-automation-pipelines-in-machine-learning.Ngày truy cập: 12/12/2025

[6]. VinBigData. (2021). MLOps: lời giải cho việc triển khai mô hình học máy. VinBigData.https://bdi.vinuni.edu.vn/blog/mlops-loi-giải-cho-việc-triển-khai-mo-hinh-học-may. Ngày truy cập: 13/12/2025

[7]. P.A Việt Nam. (2025). MLOps là gì? Tổng quan và vai trò trong triển khai mô hình học máy. P.A Việt Nam Knowledge Base. https://kb.pavietnam.vn/gioi-thieu-ve-mlops.html. Ngày truy cập: 18/12/2025

[8]. MLOpsVN.(2024). Tổng quan MLOps. MLOpsVN Courses. https://courses.mlops.vn/mlops-crash-course/tong-quan-he-thong/tong-quan-mlops.html. Ngày truy cập: 26/12/2025

[9]. S. Moreschini, F. Lomio, D. Hästbacka and D. Taibi, "MLOps for evolvable AI intensive software systems," 2022 IEEE International Conference on Software Analysis, Evolution and Reengineering (SANER), Honolulu, HI, USA, 2022, pp. 1293-1294, doi: 10.1109/SANER53432.2022.00155

[10]. Rahman, F. A. Bhuiyan, M. M. Hassan, H. Shahriar and F. Wu, "Towards Automation for MLOps: An Exploratory Study of Bot Usage in Deep Learning Libraries," 2022 IEEE 46th Annual Computers, Software, and Applications

Conference (COMPSAC), Los Alamitos, CA, USA, 2022, pp. 1093-1097, doi: 10.1109/COMPSAC54236.2022.00171.

[11]. M. Testi et al., "MLOps: A Taxonomy and a Methodology," in IEEE Access, vol. 10, pp. 63606-63618, 2022, doi: 10.1109/ACCESS.2022.3181730.

[12]. Schäfer, P., et al. (2021). Building A Platform for Machine Learning Operations from OpenSource Frameworks. IFAC-PapersOnLine, 54(1),133-138.

[13]. Liebau, K., et al. (2025). MLOps Landscape in 2025: Top Tools and Platforms.

[14]. Khochare, Y. Simmhan, S. Mehta and A. Agarwal, "Toward Scientific Workflows in a Serverless World," 2022 IEEE 18th International Conference on e-Science (e-Science), Salt Lake City, UT, USA, 2022, pp. 399-400, doi: 10.1109/eScience55777.2022.00057.

[15]. Tabular format - https://www.sciencedirect.com/topics/computer-science/tabular-format. Ngày truy cập: 11/01/2026

[16]. Các Nguồn Dữ Liệu Tệp (File) (2025). https://www.mastering-da.com/cac-nguon-du-lieu-file-cach-lay-tu-cac-tep-file. Ngày truy cập: 12/01/2026

[17]. Realtime Data Streaming: Kafka, Flink & Ứng Dụng Thực Tế. https://www.mcivietnam.com/blog-detail/realtime-data-streaming-kafka-flink-ung-dung-thuc-te-NJ2TID. Ngày truy cập: 13/01/2026

[18]. Structured Streaming + Kafka Integration Guide (Kafka broker version 0.10.0 or higher) - Spark 4.0.1 Documentation

[19]. Data source file format – Topic Data Source (2024). https://experienceleague.adobe.com/en/docs/analytics/import/data-sources/file-format. Ngày truy cập: 18/01/2026

[20]. A., Chen, A., Davidson, A., Ghodsi, A., Hong, S. A., Konwinski, A., Murching, S., Nykodym, T., Ogilvie, P., Parkhe, M., Xie, F., & Zumar, C. (2018). Accelerating the Machine Learning Lifecycle with MLflow. IEEE Data Eng. Bull., 41, 39–45. Ngày truy cập: 09/02/2026

[21]. Kreuzberger, D., Kühl, N., & Hirschl, S. (2023). Machine Learning Operations (MLOps): Overview, Definition, and Architecture. IEEE Access.

[22]. Databricks, MLflow Model Registry Workflow, Databricks Inc., 2025.

[23]. Databricks, MLflow Documentation, Databricks Inc., 2025.

[24]. MinIO, Inc. (2024). Accessing MinIO with Python using S3FS. [Online]. Available: https://min.io/docs/. Ngày truy cập: 26/02/2026

[25]. Braun, Practical MLOps: Operationalizing Machine Learning Models, O'Reilly Media, 2020.