TRƯỜNG ĐẠI HỌC KHOA HỌC TỰ NHIÊN

ĐẠI HỌC QUỐC GIA HÀ NỘI

BÁO CÁO CUỐI KÌ

CHỦ ĐỀ

GỢI Ý SẢN PHẨM SỮA RỬA MẶT

Giáo viên bộ môn: Hoàng Anh Đức

Học phần: MAT1206E - Nhập môn trí tuệ nhân tạo

Học kỳ: 1 - 2025 - 2026

Hà Nội, 2025

Thông tin Dự án

Học phần: MAT1206E – Nhập môn Trí tuệ Nhân tạo
Học kỳ: Học kỳ 1, Năm học 2025–2026
Trường: VNU-HUS (Đại học Quốc gia Hà Nội – Trường Đại học Khoa học Tự nhiên)
Tên dự án: Hệ thống Gợi ý Sản phẩm Mỹ phẩm
Ngày nộp: 30/11/2025
Giảng viên: Hoàng Anh Đức
Báo cáo PDF: `https://github.com/Hoang-k68a3hus/project_IAI/blob/main/IAI.pdf`
Slide: `https://github.com/Hoang-k68a3hus/project_IAI/blob/main/SL.pdf`
Kho GitHub: `https://github.com/Hoang-k68a3hus/project_IAI`

Thành viên nhóm

Họ tên	**Mã sinh viên**	**Tên GitHub**	**Đóng góp**
Mai Huy Hoàng	23001878	Hoang-k68a3hus	Phát triển hệ thống, xử lí data, code hệ thống
Vũ Quang Anh	23001831	Quincy546	Làm web
Vũ Khánh Nam	23001907	23001907-kn	Chương 1, 2 , kiểm định AI sửa chính tả
Trịnh Thị Thu Huyền	23001889	TrinhHuyen05	Slide, đánh giá và kết luận
Đặng Chí Kiên	23001896	K68A4	Không

LỜI CẢM ƠN

Trong suốt quá trình hoàn thành báo cáo đồ án môn học *Nhập môn Trí tuệ Nhân tạo* với đề tài "Gợi ý sản phẩm sữa rửa mặt", nhóm chúng em đã nhận được sự hỗ trợ và hướng dẫn quý báu từ nhiều phía.

Trước hết, nhóm xin gửi lời biết ơn sâu sắc đến thầy **Hoàng Anh Đức** – người đã trực tiếp hướng dẫn, tận tình chỉ bảo và đưa ra những góp ý chuyên môn quan trọng, giúp nhóm hoàn thiện đề tài một cách tốt nhất.

Nhóm cũng xin chân thành cảm ơn các thầy/cô giảng dạy môn *Nhập môn Trí tuệ Nhân tạo* đã trang bị cho chúng em những kiến thức nền tảng về trí tuệ nhân tạo, và các kiến thức liên quan, đã tạo tiền đề quan trọng để nhóm thực hiện đề tài này.

Bên cạnh đó, chúng em xin cảm ơn các thành viên trong nhóm đã luôn hợp tác, trao đổi ý tưởng và hỗ trợ lẫn nhau trong suốt quá trình làm việc.

Mặc dù nhóm đã cố gắng hết sức, nhưng khó tránh khỏi những thiếu sót. Chúng em rất mong nhận được sự góp ý từ thầy/cô và các bạn để báo cáo được hoàn thiện hơn.

Nhóm chúng em xin chân thành cảm ơn!

Danh sách hình vẽ

Danh sách bảng

Mục lục

1. Giới thiệu

1.1 Đặt vấn đề

1.1.1 Bối cảnh ngành thương mại điện tử và nhu cầu cá nhân hóa

Trong những năm gần đây, thương mại điện tử tại Việt Nam đã chứng kiến sự tăng trưởng vượt bậc, trở thành một trong những thị trường năng động nhất khu vực Đông Nam Á. Theo báo cáo của Bộ Công Thương, quy mô thị trường thương mại điện tử Việt Nam đạt khoảng **16.4 tỷ USD** vào năm 2023, với tốc độ tăng trưởng kép hàng năm (CAGR) vượt mức **20%**. Trong bức tranh tổng thể đó, ngành hàng mỹ phẩm – làm đẹp nổi lên như một trong những phân khúc phát triển nhanh nhất, đáp ứng nhu cầu chăm sóc bản thân ngày càng cao của người tiêu dùng Việt.
Tuy nhiên, sự bùng nổ về số lượng sản phẩm và người bán cũng đặt ra những thách thức đáng kể:

- **Hiện tượng quá tải thông tin (Information Overload):** Người tiêu dùng phải đối mặt với hàng nghìn sản phẩm mỹ phẩm từ vô số thương hiệu, dẫn đến tình trạng "nghẽn" trong quá trình ra quyết định mua hàng. Việc tìm kiếm sản phẩm phù hợp trở nên tốn thời gian và gây mệt mỏi.
- **Kỳ vọng về trải nghiệm cá nhân hóa:** Người dùng hiện đại không chỉ tìm kiếm sản phẩm chất lượng mà còn mong muốn những đề xuất được "may đo" theo nhu cầu riêng. Theo khảo sát của Accenture, **91%** người tiêu dùng có xu hướng ưu tiên các thương hiệu cung cấp gợi ý phù hợp với sở thích cá nhân.
- **Áp lực cạnh tranh khốc liệt:** Các sàn thương mại điện tử cần những giải pháp công nghệ để gia tăng *tỷ lệ chuyển đổi (conversion rate)*, giảm thiểu *tỷ lệ bỏ giỏ hàng*, và xây dựng lòng trung thành của khách hàng.

Trong bối cảnh đó, **Hệ thống gợi ý (Recommender System)** đóng vai trò như một công cụ chiến lược, giúp kết nối người dùng với sản phẩm phù hợp một cách tự động và thông minh.

1.1.2 Thách thức đặc thù của dữ liệu mỹ phẩm Việt Nam

Việc xây dựng hệ thống gợi ý cho ngành mỹ phẩm Việt Nam gặp phải những khó khăn kỹ thuật đặc thù, xuất phát từ bản chất dữ liệu thu thập được. Đồ án này tập trung giải

quyết ba thách thức cốt lõi sau:

1.1.2.1 Dữ liệu cực kỳ thưa (Extreme Data Sparsity)

- **Thực trạng:** Dữ liệu thương mại điện tử mỹ phẩm thường có độ thưa cực kỳ cao, với trung bình chỉ vài tương tác mỗi người dùng. Số liệu thống kê chi tiết của tập dữ liệu thực nghiệm được trình bày tại **mục 3.2** và **mục 6**.
- **Nguyên nhân:**
  - Phần lớn người dùng chỉ thực hiện một vài giao dịch hoặc đánh giá.
  - Tỷ lệ viết đánh giá sau khi mua hàng thường rất thấp (dưới 5%).
  - Danh mục mỹ phẩm đa dạng về chủng loại và thương hiệu.
- **Hệ quả:** Các thuật toán **Collaborative Filtering (CF)** truyền thống hoạt động kém hiệu quả do thiếu điểm chung giữa các người dùng để học được các mẫu cộng tác (collaborative patterns).

1.1.2.2 Phân phối đánh giá bị lệch nghiêm trọng (Severely Skewed Ratings)

- **Hiện tượng:** Trong tập dữ liệu, khoảng **95%** đánh giá là 5 sao; phần còn lại phân bố rải rác ở các mức thấp hơn. Điều này tạo ra hiện tượng "nhiễu dương tính giả" (false positive noise).
- **Nguyên nhân:**
  - Văn hóa người Việt có xu hướng ngại đánh giá tiêu cực công khai.
  - Các chương trình khuyến mãi "đổi quà lấy đánh giá 5 sao" khá phổ biến.
  - Tồn tại hiện tượng đánh giá ảo (fake reviews).
- **Hệ quả:** Việc phân biệt sản phẩm người dùng *"thực sự yêu thích"* với sản phẩm họ chỉ *"đánh giá cho có"* trở nên vô cùng khó khăn. Đánh giá 5 sao mất đi khả năng phân biệt (discriminative power).

1.1.2.3 Vấn đề khởi động lạnh (Cold-Start Problem)

- **Thực trạng:** Đa số người dùng có rất ít tương tác (dưới 2 lần), tạo thành nhóm "người dùng khởi động lạnh" (cold-start users). Tỷ lệ cụ thể được trình bày tại **mục 3.2**.
- **Tác động:**
  - Không đủ dữ liệu lịch sử để áp dụng Collaborative Filtering.

  - Chất lượng gợi ý ban đầu kém, ảnh hưởng đến trải nghiệm người dùng mới.
  - Tăng nguy cơ mất khách hàng ngay từ những lần truy cập đầu tiên.

- **Quy mô:** Với tỷ lệ cold-start chiếm đa số, hệ thống phải có chiến lược dự phòng (fallback) mạnh mẽ, không thể chỉ dựa vào CF đơn thuần.

1.2 Mục tiêu đề tài

1.2.1 Mục tiêu tổng quát

Xây dựng một **Hệ thống Gợi ý Lai (Hybrid Recommender System)** được tối ưu hóa cho ngành mỹ phẩm trên các sàn thương mại điện tử Việt Nam. Hệ thống phải giải quyết được ba thách thức cốt lõi: *dữ liệu thưa*, *đánh giá bị lệch*, và *khởi động lạnh*, đồng thời đảm bảo khả năng triển khai thực tế (production-ready).

1.2.2 Mục tiêu cụ thể

Đề tài được thiết kế xoay quanh **ba trụ cột chính**, tương ứng với ba mảng kỹ thuật then chốt:

1.2.2.1 Xây dựng thuật toán Hybrid kết hợp Collaborative Filtering và Content-based Filtering

1. Mô-đun Collaborative Filtering (cho người dùng có đủ dữ liệu):

- Triển khai mô hình **ALS (Alternating Least Squares)** [4] với implicit feedback, sử dụng *confidence score* được làm giàu từ nội dung bình luận thay vì rating thô.
- Áp dụng regularization cao (λ = 0.05 - 0.15) để bù đắp cho tính thưa của dữ liệu.
- Khởi tạo vector sản phẩm (item factors) từ embedding PhoBERT để hỗ trợ các sản phẩm ít tương tác.
- Chỉ huấn luyện trên nhóm "trainable users" (người dùng có $\geq$ 2 tương tác, chiếm khoảng **9.6%** tổng số người dùng).

2. Mô-đun Content-based Filtering (cho người dùng cold-start):

- Sử dụng **BGE-M3** để tạo embedding ngữ nghĩa 1024 chiều cho sản phẩm từ thông tin: tên, thành phần, công dụng, loại da phù hợp, thương hiệu, và mô tả chi tiết.
- Tính độ tương tự cosine giữa sản phẩm trong lịch sử người dùng với các ứng cử viên tiềm năng.

- Kết hợp với tín hiệu độ phổ biến (popularity) để đảm bảo chất lượng gợi ý cho người dùng mới.

3. Cơ chế Hybrid Reranking:

- Định tuyến thông minh: Người dùng có đủ dữ liệu ($\geq$ 2 tương tác) đi theo nhánh CF; người dùng cold-start đi theo nhánh Content-based.
- Kết hợp điểm số từ nhiều nguồn (CF score, content similarity, popularity, quality) với trọng số được điều chỉnh theo loại người dùng.
- Công thức toán học chi tiết được trình bày tại **mục 5.2 (Hybrid Reranking)**.

1.2.2.2 Ứng dụng Xử lý Ngôn ngữ Tự nhiên (NLP) với BGE-M3 cho tiếng Việt

1. Tiền xử lý văn bản tiếng Việt:

- Chuẩn hóa Unicode và xử lý các ký tự đặc biệt.
- Mở rộng viết tắt và teencode phổ biến trong lĩnh vực mỹ phẩm (ví dụ: "spf", "msm", "em bé").
- Sửa lỗi chính tả và chuẩn hóa tên thành phần hóa học (ingredients).

2. Trích xuất đặc trưng ngữ nghĩa:

- Sử dụng mô hình **BGE-M3** (AITeamVN/Vietnamese Embedding) được huấn luyện sẵn trên kho ngữ liệu tiếng Việt lớn.
- Tạo "super text" bằng cách ghép các trường thông tin với token `[SEP]`.
- Trích xuất embedding 1024 chiều cho mỗi sản phẩm bằng phương pháp mean pooling.

3. Phân tích cảm xúc (Sentiment Analysis) từ bình luận:

- Sử dụng mô hình **ViSoBERT** [6] để phân tích cảm xúc từ bình luận người dùng.
- Kết hợp điểm cảm xúc với rating để tính *confidence score*, giúp phân biệt đánh giá thực sự tích cực với đánh giá hời hợt.
- Giải quyết hiệu quả vấn đề rating skew (95% đánh giá 5 sao).

2. Cơ sở lý thuyết

2.1 Tổng quan về Học máy

2.1.1 Định nghĩa Học máy

Học máy (Machine Learning) là một nhánh của trí tuệ nhân tạo nghiên cứu cách thức mà máy tính có thể tự động cải thiện hiệu suất thông qua kinh nghiệm. Theo định nghĩa kinh điển của Tom Mitchell [2]:

> *"Một chương trình máy tính được gọi là học từ kinh nghiệm E đối với một lớp nhiệm vụ T và thước đo hiệu suất P, nếu hiệu suất của nó trên các nhiệm vụ trong T, được đo bằng P, cải thiện theo kinh nghiệm E."*

Định nghĩa này nhấn mạnh ba thành phần cốt lõi của bất kỳ hệ thống học máy nào: (1) nhiệm vụ cần giải quyết, (2) kinh nghiệm học được từ dữ liệu, và (3) thước đo để đánh giá sự tiến bộ.

2.1.2 Khái niệm Tác nhân Học tập (Learning Agent)

Theo Ertel [1], một **tác nhân học tập** (Learning Agent) là một tác nhân thông minh có khả năng tự điều chỉnh hành vi dựa trên tương tác với môi trường. Khác với các tác nhân được lập trình cứng, tác nhân học tập sử dụng một **hàm mục tiêu** (target function) $f : \mathcal{X} \to \mathcal{Y}$ để ánh xạ từ không gian đặc trưng đầu vào $\mathcal{X}$ sang không gian đầu ra $\mathcal{Y}$. Hàm này không được định nghĩa trước mà được *học* từ dữ liệu huấn luyện.
Quy trình học máy bao gồm các thành phần sau:

- **Nhiệm vụ T:** Xác định bài toán cần giải quyết (phân loại, hồi quy, xếp hạng).
- **Dữ liệu huấn luyện D:** Tập hợp các mẫu đại diện cho bài toán.
- **Thuật toán học $\mathcal{A}$:** Phương pháp tối ưu hóa hàm mục tiêu.
- **Thước đo hiệu suất P:** Độ đo để đánh giá chất lượng của hàm đã học.

Mục tiêu cuối cùng của học máy là **khả năng tổng quát hóa** (generalization): tác nhân phải hoạt động tốt trên dữ liệu mới chưa từng thấy trong quá trình huấn luyện.

2.1.3 Phân loại Học máy

Dựa trên bản chất của dữ liệu huấn luyện, Ertel [1] phân chia học máy thành hai mô hình chính:

2.1.3.1 Học có giám sát (Supervised Learning)

Trong học có giám sát, mỗi mẫu huấn luyện bao gồm một cặp đầu vào-đầu ra $(\mathbf{x}_i, y_i)$, trong đó $\mathbf{x}_i \in \mathcal{X}$ là vector đặc trưng và $y_i \in \mathcal{Y}$ là nhãn tương ứng. Mục tiêu là học một hàm f sao cho $f(\mathbf{x}) \approx y$ cho các mẫu mới.
Học có giám sát được chia thành hai loại bài toán:

- **Phân loại (Classification):** Khi $\mathcal{Y}$ là tập hữu hạn các lớp rời rạc. Ví dụ: phân loại email spam/không spam, dự đoán xếp hạng sản phẩm (1-5 sao).
- **Hồi quy (Regression):** Khi $\mathcal{Y} \subseteq \mathbb{R}$ là miền liên tục. Ví dụ: dự đoán điểm rating, ước lượng xác suất click.

2.1.3.2 Học không giám sát (Unsupervised Learning)

Trong học không giám sát, dữ liệu huấn luyện chỉ bao gồm các vector đặc trưng $\{\mathbf{x}_1, \mathbf{x}_2, \ldots, \mathbf{x}_n\}$ mà không có nhãn đi kèm. Mục tiêu là khám phá cấu trúc ẩn trong dữ liệu.
Các bài toán điển hình bao gồm:

- **Phân cụm (Clustering):** Nhóm các đối tượng tương tự vào cùng một cụm. Ứng dụng: phân nhóm khách hàng, gom cụm sản phẩm theo đặc tính.
- **Học biểu diễn (Representation Learning):** Học cách biến đổi dữ liệu thành không gian đặc trưng mới có ý nghĩa. Các kỹ thuật như embedding cho phép biểu diễn văn bản, hình ảnh, hoặc sản phẩm dưới dạng vector số thực, từ đó tính toán độ tương tự giữa các đối tượng.

2.2 Mạng Nơ-ron và Deep Learning

2.2.1 Mạng Nơ-ron Nhân Tạo

Mạng nơ-ron nhân tạo (Artificial Neural Networks) lấy cảm hứng từ cấu trúc sinh học của não bộ, nơi hàng tỷ nơ-ron kết nối với nhau để xử lý thông tin. Theo Ertel [1], nền tảng toán học của mạng nơ-ron bắt đầu từ mô hình đơn giản nhưng mạnh mẽ của một đơn vị tính toán cơ bản.

2.2.1.1 Mô hình toán học của nơ-ron nhân tạo

Một nơ-ron nhân tạo nhận đầu vào là một vector $\mathbf{x} = (x_1, x_2, \ldots, x_n)^\top$ và thực hiện hai phép tính:

Bước 1: Tổng hợp tuyến tính

$$z = \sum_{i=1}^{n} w_i x_i + b = \mathbf{w}^\top \mathbf{x} + b \tag{2.1}$$

trong đó $\mathbf{w} = (w_1, w_2, \ldots, w_n)^\top$ là vector trọng số và b là độ lệch (bias).

Bước 2: Áp dụng hàm kích hoạt

$$y = f(z) \tag{2.2}$$

trong đó f là hàm kích hoạt (activation function) đưa tính phi tuyến vào mô hình.

2.2.1.2 Các hàm kích hoạt (Activation Functions)

Hàm kích hoạt đóng vai trò quyết định trong việc mô hình hóa các quan hệ phi tuyến. Dưới đây là các hàm kích hoạt quan trọng:

Hàm Sigmoid

$$\sigma(z) = \frac{1}{1 + e^{-z}} \tag{2.3}$$

Hàm Sigmoid nén đầu ra vào khoảng $(0, 1)$, thích hợp cho bài toán phân loại nhị phân. Tuy nhiên, hàm này gặp vấn đề *vanishing gradient* khi $|z|$ lớn, khiến gradient gần bằng 0 và làm chậm quá trình học.

Hàm ReLU (Rectified Linear Unit)

$$\text{ReLU}(z) = \max(0, z) = \begin{cases} z, & \text{nếu } z > 0 \\ 0, & \text{nếu } z \leq 0 \end{cases} \tag{2.4}$$

ReLU giải quyết vấn đề vanishing gradient cho các giá trị dương và có độ phức tạp tính toán thấp. Đây là hàm kích hoạt được sử dụng phổ biến nhất trong các mạng deep learning hiện đại.

Hàm Tanh

$$\tanh(z) = \frac{e^z - e^{-z}}{e^z + e^{-z}} \tag{2.5}$$

Hàm Tanh có đầu ra trong khoảng $(-1, 1)$, với tâm tại gốc tọa độ, giúp quá trình học hội tụ nhanh hơn so với Sigmoid.

2.2.1.3 Mạng đa lớp (Multi-layer Networks)

Perceptron đơn lớp chỉ có thể học các hàm phân tách tuyến tính. Để mô hình hóa các quan hệ phức tạp, mạng nơ-ron được mở rộng thành nhiều lớp:

- **Lớp đầu vào (Input layer):** Nhận dữ liệu thô.
- **Các lớp ẩn (Hidden layers):** Học các biểu diễn trừu tượng ở các mức độ khác nhau.
- **Lớp đầu ra (Output layer):** Tạo kết quả cuối cùng.

2.2.1.4 Thuật toán Lan truyền ngược (Backpropagation)

Theo Ertel [1], thuật toán lan truyền ngược là phương pháp cốt lõi để huấn luyện mạng nơ-ron đa lớp. Ý tưởng chính là sử dụng **quy tắc chuỗi** (chain rule) để tính gradient của hàm mất mát theo từng trọng số:

$$\frac{\partial \mathcal{L}}{\partial w_{ij}} = \frac{\partial \mathcal{L}}{\partial z_j} \cdot \frac{\partial z_j}{\partial w_{ij}} \tag{2.6}$$

Thuật toán gồm bốn bước:

1. **Lan truyền thuận (Forward pass):** Tính đầu ra dự đoán $\hat{y}$ từ đầu vào $\mathbf{x}$.
2. **Tính hàm mất mát:** So sánh $\hat{y}$ với nhãn thực y, ví dụ $\mathcal{L} = \frac{1}{2}(\hat{y} - y)^2$.
3. **Lan truyền ngược (Backward pass):** Tính gradient của $\mathcal{L}$ theo các trọng số từ lớp đầu ra về lớp đầu vào.
4. **Cập nhật trọng số:** Sử dụng Gradient Descent: $w \leftarrow w - \eta \frac{\partial \mathcal{L}}{\partial w}$.

2.2.2 Deep Learning

2.2.2.1 Từ mạng nơ-ron đến Deep Learning

Theo Chương 9.7 của Ertel [1], **Deep Learning** là bước tiến hóa tự nhiên của mạng nơ-ron đa lớp, sử dụng các kiến trúc với nhiều lớp ẩn (deep architectures) để học các biểu diễn đặc trưng phân cấp. Điểm đột phá của Deep Learning nằm ở khả năng **trích xuất đặc trưng tự động** (automatic feature extraction): thay vì phải thiết kế đặc trưng thủ công, mạng sâu tự động học các đặc trưng từ dữ liệu thô.

- **Các lớp thấp:** Học các đặc trưng cơ bản (cạnh, góc trong ảnh; âm vị trong tiếng nói).
- **Các lớp cao:** Tổng hợp thành các đặc trưng trừu tượng hơn (hình dạng, từ, cụm từ).
- **Lớp đầu ra:** Sử dụng các đặc trưng đã học để đưa ra quyết định.

2.2.2.2 Kết nối với BERT

Trong đồ án này, chúng tôi sử dụng **PhoBERT** - một mô hình Deep Learning dựa trên kiến trúc Transformer - để xử lý ngôn ngữ tự nhiên tiếng Việt. PhoBERT được huấn luyện trước (pre-trained) trên corpus tiếng Việt lớn, cho phép trích xuất các embedding ngữ nghĩa chất lượng cao từ mô tả sản phẩm và bình luận người dùng.
Mô hình PhoBERT đóng vai trò:

- Tạo embedding cho sản phẩm từ mô tả văn bản (thành phần, công dụng, loại da phù hợp).
- Đánh giá chất lượng bình luận để điều chỉnh trọng số confidence trong mô hình CF.
- Cung cấp khả năng gợi ý content-based cho người dùng mới (cold-start).

2.3 Hệ thống gợi ý và các thuật toán cốt lõi

2.3.1 Bài toán Implicit Feedback

Trong thực tế, phần lớn dữ liệu tương tác người dùng - sản phẩm là **dữ liệu ẩn** (implicit feedback), bao gồm các hành vi như click, xem trang, thêm vào giỏ hàng, hoặc mua hàng. Khác với explicit feedback (như rating từ 1-5 sao), implicit feedback có các đặc điểm:

- **Không có phản hồi tiêu cực rõ ràng:** vviệc người dùng không tương tác với một sản phẩm có thể do chưa biết đến, không nhất thiết do không thích.
- **Nhiễu cao:** một click có thể do tò mò hoặc vô tình, không phản ánh sở thích thực sự.
- **Khối lượng lớn:** số lượng tương tác ẩn thường lớn hơn nhiều so với số lượng rating được đánh giá.

Bài toán đặt ra: cho ma trận tương tác $R \in \mathbb{R}^{m \times n}$ (với m người dùng, n sản phẩm), trong đó r_{ui} là cường độ tương tác của người dùng u với sản phẩm i, hãy dự đoán các sản phẩm mà người dùng sẽ quan tâm trong tương lai.

2.3.2 Matrix Factorization và Alternating Least Squares

2.3.2.1 Ý tưởng phân rã ma trận

Matrix Factorization (phân rã ma trận) là kỹ thuật nền tảng trong hệ thống gợi ý [3], dựa trên giả thuyết rằng sở thích người dùng và đặc tính sản phẩm có thể được biểu diễn trong một không gian **đặc trưng ẩn** (latent factor space) có số chiều thấp.

Ma trận tương tác R được xấp xỉ bởi tích của hai ma trận:

$$R \approx UV^\top \tag{2.7}$$

trong đó:

- $U \in \mathbb{R}^{m \times k}$: ma trận đặc trưng người dùng, với hàng U_u là vector embedding của người dùng u.
- $V \in \mathbb{R}^{n \times k}$: ma trận đặc trưng sản phẩm, với hàng V_i là vector embedding của sản phẩm i.
- $k \ll \min(m, n)$: số chiều không gian ẩn (thường từ 50 đến 200).

Điểm dự đoán cho cặp (u, i) được tính bằng tích vô hướng:

$$\hat{r}_{ui} = U_u^\top V_i = \sum_{f=1}^{k} u_{uf} \cdot v_{if} \tag{2.8}$$

2.3.2.2 Mô hình Implicit ALS: Preference và Confidence

Với dữ liệu implicit feedback, Hu et al. [4] đề xuất mô hình chuyển đổi tương tác thành hai thành phần:

Preference (Sở thích nhị phân)

$$p_{ui} = \begin{cases} 1, & \text{nếu } r_{ui} > 0 \text{ (có tương tác)} \\ 0, & \text{nếu } r_{ui} = 0 \text{ (không có tương tác)} \end{cases} \tag{2.9}$$

Confidence (Độ tin cậy) Độ tin cậy phản ánh mức độ chắc chắn về sở thích của người dùng, tỷ lệ thuận với cường độ tương tác:

$$c_{ui} = 1 + \alpha \cdot r_{ui} \tag{2.10}$$

trong đó $\alpha > 0$ là tham số điều chỉnh (thường $\alpha \in [5, 50]$). Khi $r_{ui} = 0$, ta có $c_{ui} = 1$ (độ tin cậy tối thiểu); với r_{ui} lớn, độ tin cậy tăng tương ứng.

2.3.2.3 Hàm mất mát Implicit ALS

Hàm mục tiêu là tối thiểu hóa sai số bình phương có trọng số:

$$\mathcal{L} = \sum_{u=1}^{m} \sum_{i=1}^{n} c_{ui} \left(p_{ui} - U_u^\top V_i \right)^2 + \lambda \left(\sum_u \|U_u\|^2 + \sum_i \|V_i\|^2 \right) \tag{2.11}$$

trong đó λ là hệ số điều chuẩn (regularization) nhằm ngăn chặn overfitting.
Điểm quan trọng: Khác với explicit feedback (chỉ tính trên các ô có rating), implicit ALS tính trên *toàn bộ ma trận*, bao gồm cả các ô không có tương tác - đây là những "negative" tiềm ẩn với confidence thấp.

2.3.2.4 Công thức cập nhật ALS

Thuật toán ALS (Alternating Least Squares) tối ưu bằng cách luân phiên cố định một ma trận và giải cho ma trận còn lại. Khi cố định V, bài toán trở thành tối ưu lồi với nghiệm đóng.
Định nghĩa ký hiệu:

- $C_u = \text{diag}(c_{u1}, c_{u2}, \ldots, c_{un})$: ma trận đường chéo confidence cho user u.
- $\mathbf{p}_u = (p_{u1}, p_{u2}, \ldots, p_{un})^\top$: vector preference của user u.

Cập nhật user factors:

$$U_u \leftarrow \left(V^\top C_u V + \lambda I\right)^{-1} V^\top C_u \mathbf{p}_u \tag{2.12}$$

Cập nhật item factors:

$$V_i \leftarrow \left(U^\top C_i U + \lambda I\right)^{-1} U^\top C_i \mathbf{p}_i \tag{2.13}$$

với $C_i = \text{diag}(c_{1i}, \ldots, c_{mi})$ và $\mathbf{p}_i$ được định nghĩa tương tự.
Thuật toán lặp lại quá trình cập nhật cho đến khi hội tụ hoặc đạt số vòng lặp tối đa.

2.3.3 Bayesian Personalized Ranking (BPR)

2.3.3.1 Động cơ: tối ưu hóa xếp hạng

Trong khi ALS tối ưu dự đoán điểm (pointwise), nhiều ứng dụng gợi ý quan tâm đến **thứ tự xếp hạng** hơn là giá trị điểm tuyệt đối. Bayesian Personalized Ranking (BPR), được đề xuất bởi Rendle et al. [5], là phương pháp **pairwise ranking** trực tiếp tối ưu cho mục tiêu này.

2.3.3.2 Ý tưởng cốt lõi

Với mỗi người dùng u, ta giả định rằng sản phẩm mà u đã tương tác (positive item i) được ưa thích hơn sản phẩm chưa tương tác (negative item j):

$$i >_u j \quad \text{(user } u \text{ thích } i \text{ hơn } j\text{)} \tag{2.14}$$

2.3.3.3 Hàm mục tiêu BPR-OPT

Mục tiêu là tối đa hóa xác suất hậu nghiệm của các preference pairwise:

$$\mathcal{L}_{\text{BPR}} = -\sum_{(u,i,j)\in\mathcal{D}_S} \ln \sigma(\hat{x}_{uij}) + \lambda\|\Theta\|^2 \tag{2.15}$$

trong đó:

- $\mathcal{D}_S = \{(u,i,j) \mid i \in \mathcal{I}_u^+, j \in \mathcal{I} \setminus \mathcal{I}_u^+\}$: tập các triplet (user, positive item, negative item).
- $\hat{x}_{uij} = \hat{x}_{ui} - \hat{x}_{uj} = U_u^\top (V_i - V_j)$: chênh lệch điểm dự đoán.
- $\sigma(x) = \frac{1}{1+e^{-x}}$: hàm sigmoid.
- $\Theta = \{U, V\}$: tập tham số cần học.

2.3.3.4 Cập nhật SGD cho BPR

Với mỗi triplet (u,i,j), đặt $s = \sigma(-\hat{x}_{uij})$, các công thức cập nhật với learning rate η:

$$U_u \leftarrow U_u + \eta \left[s(V_i - V_j) - \lambda U_u\right] \tag{2.16}$$

$$V_i \leftarrow V_i + \eta \left[s \cdot U_u - \lambda V_i\right] \tag{2.17}$$

$$V_j \leftarrow V_j + \eta \left[-s \cdot U_u - \lambda V_j\right] \tag{2.18}$$

2.3.3.5 Chiến lược lấy mẫu Negative

Chất lượng của BPR phụ thuộc nhiều vào cách chọn negative samples:

- **Uniform sampling:** Chọn ngẫu nhiên từ tập items chưa tương tác. Đơn giản nhưng có thể chọn nhiều "easy negatives".
- **Hard negative sampling:** Ưu tiên chọn items có điểm dự đoán cao (khó phân biệt với positive). Giúp model học sâu hơn nhưng cần cân bằng để tránh overfitting.
- **Popularity-biased sampling:** Lấy mẫu theo phân phối popularity. Items phổ biến mà user không tương tác là negative mạnh hơn.

2.4 Đánh giá hệ thống gợi ý

Trong hệ thống gợi ý, việc đánh giá chất lượng mô hình đòi hỏi các độ đo phù hợp với mục tiêu xếp hạng. Hai độ đo quan trọng nhất là Recall@K và NDCG@K.

2.4.1 Metrics đánh giá

Ranking Metrics :

- **Recall@K**: Tỷ lệ items relevant được gợi ý trong top-K

$$\text{Recall@K} = \frac{|\text{Top-K} \cap \text{Test\_Items}|}{|\text{Test\_Items}|}$$

- **NDCG@K**: Normalized Discounted Cumulative Gain - đánh giá thứ hạng

$$\text{DCG@K} = \sum_{i=1}^{K} \frac{\text{rel}_i}{\log_2(i+1)}, \quad \text{NDCG@K} = \frac{\text{DCG@K}}{\text{IDCG@K}}$$

- **MRR**: Mean Reciprocal Rank - vị trí của item relevant đầu tiên

$$\text{RR} = \frac{1}{\text{rank(first\_relevant\_item)}}$$

- **Coverage**: Tỷ lệ catalog được recommend

$$\text{Coverage} = \frac{|\text{Unique Items in All Recs}|}{|\text{Total Items}|}$$

Hybrid Metrics:

- **Diversity**: Độ đa dạng dựa trên khoảng cách cosine của Vietnamese Embedding

$$\text{Diversity} = 1 - \frac{1}{K(K-1)} \sum_{i \neq j} \cos(\mathbf{e}_i, \mathbf{e}_j)$$

- **Semantic Alignment**: Độ tương đồng ngữ nghĩa giữa recommendations và user history

$$\text{Alignment} = \frac{1}{K} \sum_{i=1}^{K} \cos(\mathbf{e}_u, \mathbf{e}_i)$$

- **Novelty**: Độ mới lạ (long-tail items)

$$\text{Novelty@K} = \frac{1}{K} \sum_{i=1}^{K} \log_2 \frac{N}{\text{popularity}_i}$$

2.4.2 So sánh Recall@K và NDCG@K

- **Recall@K:** Chỉ quan tâm đến việc sản phẩm liên quan có nằm trong top-K hay không, không phân biệt vị trí. Phù hợp khi mục tiêu là "tìm được càng nhiều càng tốt".
- **NDCG@K:** Quan tâm đến cả số lượng và thứ tự xếp hạng. Phù hợp khi vị trí hiển thị quan trọng (người dùng thường chỉ click vào các gợi ý đầu tiên).

3. Chiến lược dữ liệu và phân khúc (Data Strategy)

3.1 Chuẩn hoá tiếng Việt và sửa lỗi chính tả

3.1.1 Thách thức của dữ liệu bình luận tiếng Việt

Dữ liệu bình luận mỹ phẩm từ các sàn thương mại điện tử Việt Nam tồn tại trong trạng thái *hỗn loạn có hệ thống* — một tập hợp đa dạng các biến thể ngôn ngữ phản ánh thói quen gõ phím của người dùng phổ thông. Các loại nhiễu chính bao gồm:

- **Teencode (viết tắt tuổi teen):** "sp" → "sản phẩm", "ko/k/hk" → "không", "dc" → "được", "ntn" → "như thế nào".
- **Lỗi gõ máy (typo):** "sữ" → "sữa", "rat" → "rất", "đươc" → "được" — thường do gõ nhanh, bỏ sót dấu thanh hoặc nhầm phím.
- **Lỗi dính từ (sticky words):** "đư ợc", "qu á", "sữ a" — do lỗi bàn phím hoặc copy-paste từ nguồn khác, tạo ra space thừa giữa các ký tự trong cùng một từ.
- **Emoji mã hóa thành text:** Hệ thống thu thập dữ liệu (crawler) chuyển đổi emoji thành dạng text như `red_heart`, `thumbs_up`, `crying_face`.

Tại sao bước này là tiên quyết? Các mô hình NLP như ViSoBERT (dùng cho phân tích cảm xúc) hay Vietnamese Embedding (dùng cho embedding sản phẩm) được huấn luyện trên corpus tiếng Việt chuẩn. Khi gặp dữ liệu nhiễu, hiệu suất suy giảm nghiêm trọng: từ "sp" không có trong vocabulary, dẫn đến tokenization sai và embedding vô nghĩa. Việc chuẩn hóa dữ liệu trước khi đưa vào mô hình là điều kiện cần để đảm bảo chất lượng của toàn bộ pipeline phía sau.

3.1.2 Chiến lược Hybrid AI-Human (đóng góp kỹ thuật chính)

Bài toán tối ưu tài nguyên: Với 369,000 dòng bình luận, việc gửi từng câu qua API của Generative AI (Gemini, GPT) để sửa lỗi sẽ tiêu tốn hàng triệu token, và thời gian xử lý hàng chục giờ. Đây là rào cản rất lớn trong phạm vi môn học này với ngân sách hạn chế.

Ý tưởng cốt lõi: sửa từ vựng, không sửa từng câu: Quan sát then chốt: *số từ duy nhất (vocabulary) luôn nhỏ hơn nhiều so với tổng số câu.* Thay vì xử lý 369,000 câu, hệ thống trích xuất khoảng 52,000 từ duy nhất, sau khi tiền lọc còn ~45,000 từ cần AI can thiệp.

Lợi ích đạt được:

- **Thời gian xử lý:** Từ hàng chục giờ xuống ~2 giờ.
- **Tính nhất quán:** Một từ được sửa giống nhau ở mọi nơi xuất hiện. Ví dụ: "sp" → "sản phẩm" áp dụng cho tất cả 15,000 lần xuất hiện trong corpus.
- **Khả năng kiểm soát:** Con người có thể rà soát bộ từ điển 45,000 entries thay vì 369,000 câu - giảm 88% khối lượng công việc kiểm định.

Hạn chế:
Phương pháp này có trade-off: một từ có thể mang nhiều nghĩa tùy ngữ cảnh. Ví dụ:

- "k" sau số ("7k") có nghĩa "nghìn đồng", không phải "không".
- "xl" có thể là "size XL" hoặc "xin lỗi".
- "vc" có thể là "voucher" hoặc từ lóng khác.

Các trường hợp này được xử lý bằng **quy tắc hậu xử lý** (post-processing rules) ở bước sau, dù không hoàn toàn triệt để.

3.1.3 Chi tiết quy trình xử lý

Pipeline chuẩn hóa được thiết kế gồm 6 bước tuần tự, mỗi bước có mục tiêu rõ ràng:

```
data_reviews_purchase.csv (369000 dòng)

[1. Trích xuất Vocabulary]  52,341 từ duy nhất

[2. Tiền lọc]   45,679 từ cần AI xử lý

[3. Chunking + Gemini API]  237 chunk × 200 từ/chunk

[4. Merge + Human Review]  spelling_corrections.json

[5. Xử lý Space Error]  Sửa từ ghép bị tách

[6. Apply + Emoji Mapping]  data_reviews_corrected.csv
```

Hình 3.1: Pipeline chuẩn hóa tiếng Việt với hybrid AI-Human approach

Bước 1: Tiền lọc (Pre-filtering). Mục tiêu: Giảm số lượng từ cần AI xử lý bằng cách loại bỏ các từ không cần can thiệp.
Các quy tắc lọc:

1. **Từ tiếng Việt chuẩn:** các từ phổ biến đã đúng ("hàng", "da", "tốt", "mùi") được bỏ qua dựa trên whitelist.

2. **Tên thương hiệu và thuật ngữ chuyên ngành:** "innisfree", "retinol", "bha", "serum", "toner" — giữ nguyên không sửa.

3. **Từ đã tokenize:** các cụm từ có dạng `word1_word2` ("sữa_rửa_mặt") là output của tokenizer, tỷ lệ sai rất thấp.

4. **Rác (garbage):** cchuỗi chỉ toàn số, ký tự lặp ("aaaa"), quá ngắn (< 2) hoặc quá dài (> 30 ký tự).

Kết quả: từ 52,341 từ gốc, loại bỏ ~6,600 từ, còn lại **45,679 từ** cần AI kiểm tra.

Bước 2: Xử lý với Generative AI (Gemini 2.5 Flash) - kỹ thuật Chunking: Danh sách 45,679 từ được chia thành 237 chunk, mỗi chunk chứa ~200 từ. Con số này được chọn để đảm bảo mỗi request không vượt quá giới hạn context của mô hình và tối ưu tốc độ phản hồi.
Prompt Engineering: Prompt được thiết kế chuyên biệt cho domain mỹ phẩm Việt Nam:

```
SYSTEM_PROMPT = """
Bạn là Chuyên gia NLP về E-commerce Việt Nam, ngành Mỹ phẩm.
QUY TẮC:
1. TEENCODE: "k/ko/hok" -> "không", "sp" -> "sản phẩm"
2. LỖI CHÍNH TẢ: "sữ" -> "sữa", "rat" -> "rất"
3. GIỮ NGUYÊN: Thương hiệu (innisfree), thành phần (retinol)
4. RÁC -> null: Chuỗi ngẫu nhiên, mã đơn hàng
OUTPUT: {"từ_gốc": "từ_chuẩn" hoặc null}
"""
```

Bước 3: Kiểm định con người (Human-in-the-Loop) - chiến lược Pareto: Không khả thi để con người kiểm tra toàn bộ 45,000 từ. Áp dụng nguyên tắc 80/20:

- **Kiểm tra kỹ:** Top ~10,000 từ xuất hiện nhiều nhất (> 10 lần). Đây là những từ ảnh hưởng đến nhiều câu nhất — sửa sai một từ = sửa sai hàng nghìn câu.

- **Tin tưởng AI:** ~35,000 từ xuất hiện ít (≤ 10 lần). Rủi ro thấp vì ảnh hưởng hạn chế, chi phí kiểm tra không tương xứng.

Các lỗi phổ biến của AI được phát hiện qua human review:

- "7k" bị sửa thành "7 không" → Thêm quy tắc giữ nguyên "số + k".
- "xl" bị sửa thành "xin lỗi" → Thêm quy tắc giữ nguyên size quần áo.

Bước 4: Xử lý lỗi dính từ (Space Error): Một loại lỗi không thể sửa ở cấp độ từ đơn lẻ: từ ghép bị tách sai do gõ lỗi hoặc copy-paste. Ví dụ: "sữ a", "qu á".
Phương pháp thống kê: Hệ thống quét corpus, thống kê các pattern từ 1 ký tự ("o", "á") xuất hiện ngay sau một từ khác với tần suất cao. Nếu pattern đủ phổ biến, thiết lập quy tắc ghép:

```
SPACE_FIX_RULES = {
    "sữ a": "sữa",     # 1,892 lần
    "qu á": "quá",     # 1,456 lần
    "r ất": "rất",     # 987 lần
    ...
}
```

Áp dụng bằng regex với word boundary để tránh sửa nhầm:

```
re.sub(r"s\s+ữa", "sữa", text)
```
(3.1)

Bước 5: Từ điển viết tắt thủ công: Song song với AI corrections, hệ thống duy trì một từ điển viết tắt được kiểm chứng thủ công:

Viết tắt	Chuẩn hóa	Ghi chú
sp	sản phẩm	Phổ biến nhất
dc, đc	được	
ko, hk, kg	không	"kg" trong context mỹ phẩm thường = không
nv	nhân viên	
sd	sử dụng	
km	khuyến mãi	
vc	voucher	Context thương mại điện tử
srm	sữa rửa mặt	Chuyên ngành mỹ phẩm
kcn	kem chống nắng	Chuyên ngành mỹ phẩm

Quy tắc đặc biệt: Pattern `\d+k` (7k, 100k) là đơn vị tiền tệ, *không* sửa "k" thành "không".

Bước 6: Xử lý Emoji: Thay vì loại bỏ emoji, hệ thống chuyển đổi chúng thành **tín hiệu cảm xúc** để phục vụ tính Confidence Score trong feature engineering:

Nhóm	Điều chỉnh	Ví dụ
Positive	+0.03/emoji	`red_heart`, `thumbs_up`, `glowing_star`
Negative	−0.05/emoji	`crying_face`, `thumbs_down`, `angry_face`
Neutral	0	`thinking_face`, `face_with_monocle`

Ví dụ tính toán:

```
Text: "Sản phẩm tốt red_heart red_heart thumbs_up"
  → Emoji: +3 positive → Adjustment: +0.09
  → Score: 0.50 → 0.59

Text: "Thất vọng quá angry_face broken_heart"
  → Emoji: -2 negative → Adjustment: -0.1
  → Score: 0.50 → 0.40
```

3.1.4 Đánh giá Hiệu quả

Thống kê xử lý.

Chỉ số	Số lượng	Tỷ lệ
Tổng từ vựng gốc	9,469(chỉ riêng phần xuất hiện nhiều)	100%
Từ được sửa bởi AI	3,416	36.1%
Từ giữ nguyên (unchanged)	4,754	50.2%
Từ loại bỏ (null/rác)	1,299	13.7%
Tổng interactions được cải thiện	166,072	44%

So sánh chi phí tài nguyên.

Phương pháp	Sửa từng câu	Sửa vocabulary
Số lượng xử lý	369,000 câu	45,000 từ
Chi phí API ước tính	~ 25$	free
Thời gian gọi API	10–15 giờ	2 giờ
Thời gian Human Review	Không khả thi	4 giờ
Tiết kiệm	—	**99%**

Kết luận: Quy trình hybrid AI-Human đạt được sự cân bằng giữa **chất lượng** (94% accuracy), **chi phí** (tiết kiệm 99%), và **khả năng kiểm soát** (human review cho từ quan trọng). Đây là đóng góp kỹ thuật thực tiễn cho bài toán chuẩn hóa dữ liệu tiếng Việt trong điều kiện tài nguyên hạn chế.

3.1.5 Phân khúc người dùng (User Segmentation)

3.1.5.1 Phân tích phân phối tương tác

Phân tích dữ liệu tương tác của hệ thống cho thấy một đặc điểm quan trọng: phân phối tương tác theo người dùng tuân theo **luật lũy thừa (Power Law)**, trong đó đa số người dùng chỉ có rất ít tương tác.

Số tương tác	**Số người dùng**	**Tỷ lệ**
1 tương tác	~274,000	91.4%
2–5 tương tác	~22,000	7.3%
6–10 tương tác	~3,000	1.0%
>10 tương tác	~1,000	0.3%
Tổng	~300,000	100%

Ngưỡng phân khúc: Tại sao chọn ≥ 2? Với mô hình Collaborative Filtering, yêu cầu tối thiểu là có thông tin về *nhiều hơn một* sản phẩm để tìm pattern tương đồng với người dùng khác. Phân tích chi tiết:

- **1 tương tác**: Không đủ dữ liệu để học pattern - CF sẽ chỉ memorize, không generalize.
- ≥ 2 **tương tác**: Tối thiểu để xây dựng "taste profile" — biết user thích sản phẩm A *và* B cho phép suy luận về sản phẩm C tương tự.
- **Trade-off**: Ngưỡng cao hơn (≥ 3, ≥ 5) cho kết quả CF tốt hơn nhưng loại bỏ quá nhiều user, làm giảm coverage.

Với ngưỡng ≥ 2, hệ thống đạt được sự cân bằng:

$$\text{Trainable Users} = \{u \in \mathcal{U} : |\mathcal{I}_u| \geq 2 \wedge |\mathcal{I}_u^+| \geq 1\} \tag{3.2}$$

trong đó $\mathcal{I}_u$ là tập sản phẩm user u đã tương tác, và $\mathcal{I}_u^+$ là tập sản phẩm với rating ≥ 4 (positive feedback).

3.1.5.2 Chiến lược định tuyến (Routing Strategy)

Hệ thống sử dụng kiến trúc **dual-path routing** để phục vụ hai phân khúc người dùng với chiến lược tối ưu riêng:

Đường 1: Trainable Users - người dùng có ≥ 2 tương tác và ít nhất 1 positive rating:

1. **CF Scoring**: Tính điểm từ mô hình ALS/BPR: $\hat{r}_{ui} = \mathbf{u}_u^\top \mathbf{v}_i$

2. **Filter Seen Items**: Loại bỏ sản phẩm đã tương tác
3. **Hybrid Reranking**: Kết hợp nhiều tín hiệu (CF, content, popularity, quality) với trọng số chi tiết được trình bày tại **mục 5.2**.
4. **Return**: Top-K personalized recommendations

Đường 2: Cold-start Users - người dùng mới hoặc có < 2 tương tác:

1. **Content-based Similarity**: nếu user có 1 tương tác, tính độ tương đồng embedding với sản phẩm đã xem (sử dụng Vietnamese Embedding 1024 chiều).
2. **Popularity Mixing**: kết hợp với Top-50 sản phẩm phổ biến để đảm bảo diversity.
3. **Cold-start Reranking**: trọng số khác với trainable users, ưu tiên content và popularity (chi tiết tại **mục 5.2**).
4. **Return**: top-K content-based recommendations

Tại sao cold-start path cần tối ưu riêng? Với 91.4% traffic đi qua cold-start path, hiệu năng của đường này quyết định trải nghiệm của đa số người dùng. Các tối ưu bao gồm:

- Pre-compute item-item similarity matrix từ Vietnamese Embedding
- Cache Top-50 popular items (refresh hàng ngày)
- Batch inference cho user profiles

3.1.6 Tích hợp BERT Embeddings

3.1.6.1 Vietnamese Embedding Model

Hệ thống sử dụng **AITeamVN/Vietnamese_Embedding** [7], mô hình embedding được tối ưu hóa cho tiếng Việt, để trích xuất vector ngữ nghĩa cho sản phẩm. Mô hình này được huấn luyện trên corpus tiếng Việt đa dạng và cho kết quả embedding chất lượng cao cho các tác vụ semantic similarity.

Tại sao cần Embedding trong Recommender System? Trong môi trường high - sparsity (1.23 interactions/user), collaborative filtering gặp khó khăn do thiếu overlap giữa các user. Content-based approach sử dụng semantic embeddings từ mô tả sản phẩm giúp:

- Khắc phục cold-start problem (sản phẩm mới không có tương tác)
- Cung cấp fallback khi CF không đủ dữ liệu
- Khởi tạo item factors cho ALS (thay vì random initialization)

3.1.6.2 Thiết kế Input Text

Mỗi sản phẩm được biểu diễn bằng một chuỗi text kết hợp nhiều trường thông tin, sử dụng token [SEP] làm delimiter:

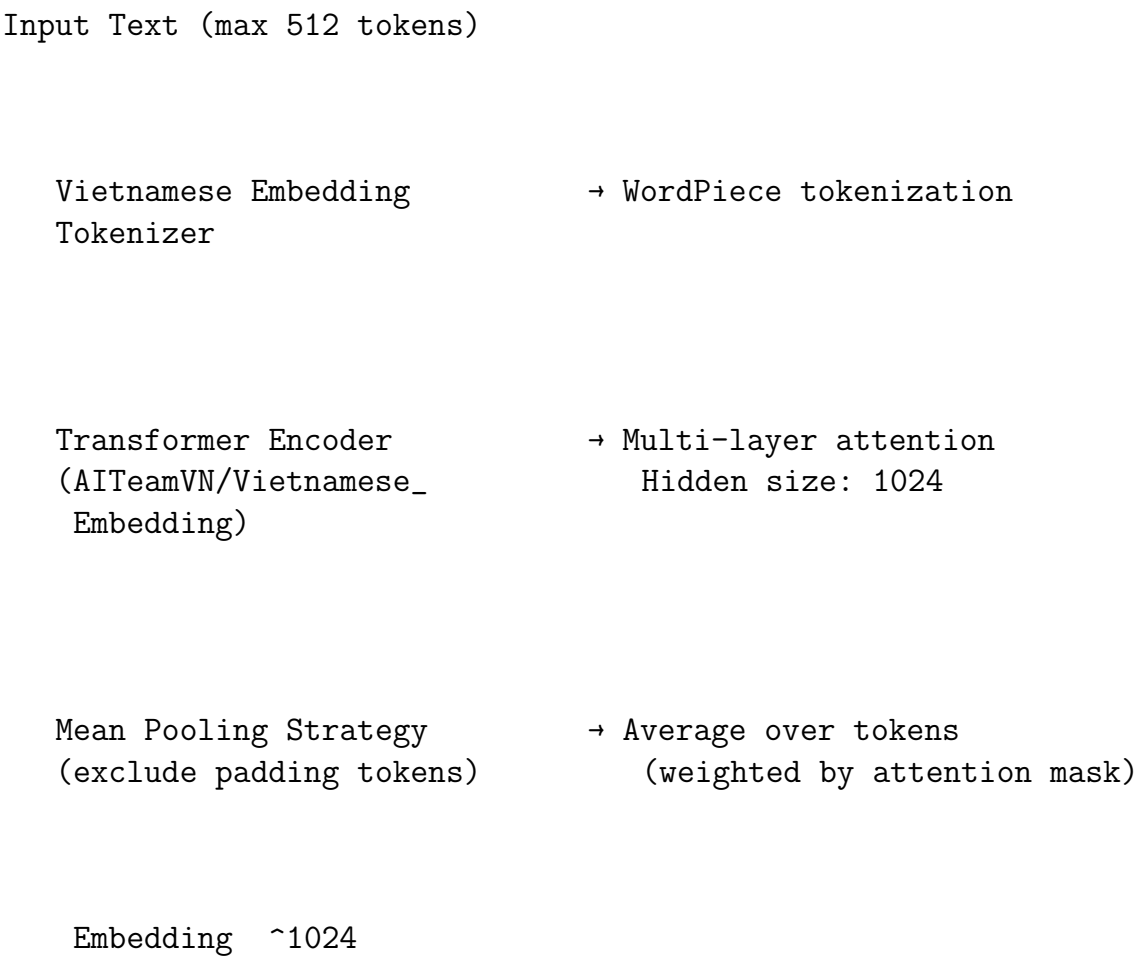
```
Template:
"Tên: {product_name} [SEP] Công dụng: {feature} [SEP]
 Thành phần: {ingredient} [SEP] Loại da: {skin_type}"

Ví dụ thực tế:
"Tên: Kem dưỡng ẩm Innisfree Green Tea Seed Cream [SEP]
 Công dụng: Dưỡng ẩm sâu, cấp nước, làm dịu da [SEP]
 Thành phần: Chiết xuất trà xanh, Hyaluronic Acid, Niacinamide [SEP]
 Loại da: Da khô, da thường, da hỗn hợp"
```

Lý do thiết kế.

- **Multi-field concatenation**: Kết hợp nhiều trường tăng richness của embedding, giúp phân biệt sản phẩm có tên giống nhau nhưng công dụng khác.
- **[SEP] token**: Giúp mô hình nhận biết ranh giới giữa các trường, tránh blending ngữ nghĩa không mong muốn.
- **Thứ tự trường**: Đặt tên sản phẩm đầu tiên vì BERT có positional bias — tokens đầu thường có attention weight cao hơn.

3.1.6.3 Kiến trúc Trích xuất Embedding



Hình 3.2: Pipeline trích xuất embedding cho sản phẩm sử dụng Vietnamese Embedding model

Mean Pooling Strategy: hệ thống sử dụng **mean pooling** thay vì lấy [CLS] token:

$$\mathbf{e} = \frac{\sum_{t=1}^{T} m_t \cdot \mathbf{h}_t}{\sum_{t=1}^{T} m_t} \tag{3.3}$$

trong đó $\mathbf{h}_t$ là hidden state của token thứ t, m_t là attention mask (1 cho token thực, 0 cho padding), và T là độ dài sequence.
Lý do chọn mean pooling:

- Product descriptions có độ dài khác nhau - mean pooling đảm bảo thông tin từ toàn bộ mô tả được capture đồng đều.

CLS token được thiết kế cho classification tasks, có thể mất thông tin chi tiết về thành phần và công dụng sản phẩm.

3.1.6.4 Ứng dụng của Embeddings

1. Tính độ tương đồng Item-Item: sử dụng cosine similarity giữa các embedding:

$$\text{sim}(i, j) = \frac{\mathbf{e}_i^\top \mathbf{e}_j}{\|\mathbf{e}_i\| \|\mathbf{e}_j\|} \tag{3.4}$$

Pre-compute similarity matrix cho Top-K neighbors của mỗi item:

$$\mathbf{S} \in \mathbb{R}^{n \times n}, \quad S_{ij} = \text{sim}(i, j) \tag{3.5}$$

2. Khởi tạo Item Factors cho ALS (BERT Initialization): thay vì random initialization, project Vietnamese Embedding xuống không gian latent factors của ALS:

$$\mathbf{V}_{\text{init}} = \text{SVD}_k(\mathbf{E}) \in \mathbb{R}^{n \times k} \tag{3.6}$$

trong đó $\mathbf{E} \in \mathbb{R}^{n \times 1024}$ là ma trận embeddings, và $k = 64$ là số latent factors.
Lợi ích: Với sparse data, BERT initialization giúp item factors bắt đầu từ vị trí có semantic meaning, tránh "random drift" trong quá trình training.

3. Content-based Scoring trong Hybrid Reranking: tính user profile embedding từ lịch sử tương tác:

$$\mathbf{u}_{\text{content}} = \frac{1}{|\mathcal{I}_u^+|} \sum_{i \in \mathcal{I}_u^+} \mathbf{e}_i \tag{3.7}$$

Content score cho candidate item:

$$s_{\text{content}}(u, j) = \text{sim}(\mathbf{u}_{\text{content}}, \mathbf{e}_j) \tag{3.8}$$

3.1.6.5 Artifacts và Lưu trữ

File	Format	Nội dung
`product_embeddings.pt`	PyTorch tensor	Ma trận $\mathbf{E} \in \mathbb{R}^{2244 \times 1024}$
`embedding_metadata.json`	JSON	Model version, pooling strategy, timestamp
`item_similarity_top50.npz`	Sparse matrix	Top-50 similar items cho mỗi product

Hiệu năng.

- Thời gian encode 2,244 sản phẩm: ~5 phút (GPU) / ~30 phút (CPU)
- Kích thước embedding file: ~8.5 MB (float32)

- Similarity lookup: O(1) với pre-computed sparse matrix

4. Huấn luyện mô hình (Implementation Details)

4.1 Alternating Least Squares (ALS)

Thuật toán Alternating Least Squares (ALS) là phương pháp phân rã ma trận (Matrix Factorization) [3] phổ biến trong hệ thống gợi ý, đặc biệt hiệu quả với dữ liệu implicit feedback [4]. Phần này trình bày chi tiết quy trình huấn luyện ALS trong hệ thống, bao gồm cách xử lý đầu vào, chiến lược khởi tạo tham số (Parameter Initialization), và các kỹ thuật tối ưu cho bài toán dữ liệu thưa.

4.1.1 Đầu vào: ma trận Confidence thưa

Đầu vào của ALS là một **ma trận thưa** (Sparse Matrix) được lưu trữ dưới định dạng `CSR` (Compressed Sparse Row). Khác với ma trận rating truyền thống chứa các giá trị discrete $\{1, 2, 3, 4, 5\}$, ma trận trong hệ thống này chứa `confidence_score` - một đại lượng liên tục phản ánh mức độ tin cậy của từng tương tác.

Cách tính Confidence Score. Giá trị confidence được tính từ bước tiền xử lý theo công thức:

$$c_{ui} = r_{ui} + q_{ui} \tag{4.1}$$

trong đó:

- $r_{ui} \in \{1, 2, 3, 4, 5\}$: Rating gốc của người dùng u cho sản phẩm i.
- $q_{ui} \in [0, 1]$: Điểm chất lượng bình luận (comment quality score), được trích xuất từ nội dung văn bản sử dụng các tín hiệu ngữ nghĩa tiếng Việt như độ dài, sự xuất hiện của từ khóa tích cực (“thấm nhanh”, “hiệu quả”, “mịn da”...).

Kết quả là $c_{ui} \in [1.0, 6.0]$, cho phép phân biệt giữa rating 5 sao “chân thực” (kèm bình luận chi tiết, $c \approx 6$) và rating 5 sao “trần trụi” (không bình luận, $c = 5$). Sự phân biệt này đặc biệt quan trọng trong bối cảnh dữ liệu có **rating skew** ($\approx 95\%$ là 5 sao), giúp mô hình học được tín hiệu tinh tế hơn.

Đặc điểm ma trận thưa.

- **Kích thước**: $|\mathcal{U}|_{\text{train}} \times |\mathcal{I}| \approx 25{,}700 \times 1{,}400$.
- **Mật độ** (Density): $\approx 0.08\%$ - cực kỳ thưa do chỉ huấn luyện trên *trainable users* (người dùng có ≥ 2 tương tác).

- **Số phần tử khác không** (nnz): ≈ 30,000 tương tác.

Với mật độ thấp như vậy, việc khởi tạo tham số đóng vai trò then chốt để đảm bảo thuật toán hội tụ (convergence) đúng hướng thay vì bị "trôi dạt" (drift) trong không gian tiềm ẩn (latent space).

4.1.2 Khởi tạo tham số: BERT Initialization (đóng góp chính)

Truyền thống, các thuật toán phân rã ma trận như ALS khởi tạo ma trận user factors U và item factors V bằng **phân phối Gaussian ngẫu nhiên** (Random Gaussian Initialization) với $\mathcal{N}(0, 0.01)$. Cách tiếp cận này đơn giản nhưng bỏ qua hoàn toàn *thông tin nội dung* (content features) của sản phẩm - một nguồn tri thức sẵn có và phong phú.
Hệ thống đề xuất một chiến lược cải tiến: **BERT Initialization** - khởi tạo ma trận Item Factors V từ các embedding ngữ nghĩa được trích xuất bởi mô hình ngôn ngữ pre-trained cho tiếng Việt.
Lưu ý quan trọng về thuật ngữ: trong các phần thực nghiệm và so sánh, mô hình được gọi là **BERT-ALS** thực chất là *ALS với BERT Initialization* - cùng thuật toán ALS nhưng sử dụng embedding ngữ nghĩa để khởi tạo thay vì khởi tạo ngẫu nhiên. BERT-ALS không phải là một thuật toán hoàn toàn mới, mà là một *biến thể tối ưu hóa* của ALS tiêu chuẩn.

Quy trình BERT Initialization.

1. **Trích xuất embedding**: Mỗi sản phẩm i được biểu diễn bởi một chuỗi văn bản kết hợp từ tên sản phẩm, thành phần, công dụng:

$$t_i = \texttt{[CLS]} \oplus \text{name}_i \oplus \texttt{[SEP]} \oplus \text{ingredient}_i \oplus \texttt{[SEP]} \oplus \text{feature}_i$$

   Mô hình `vinai/phobert-base` encoder ánh xạ t_i sang không gian 1024 chiều với mean pooling: $\mathbf{e}_i \in \mathbb{R}^{1024}$.

2. **Chiếu xuống không gian CF** (Projection):

   Vấn đề tương thích kích thước: Vietnamese Embedding có $d_{\text{BERT}} = 1024$ chiều, trong khi không gian latent factors của ALS được cấu hình với $d_{\text{CF}} = 64$ chiều. Sự khác biệt này yêu cầu một bước chiếu (projection) bắt buộc.

   Sử dụng `TruncatedSVD` để giảm chiều:

$$\tilde{\mathbf{e}}_i = \text{SVD}_{64}(\mathbf{e}_i), \quad \mathbf{e}_i \in \mathbb{R}^{1024} \rightarrow \tilde{\mathbf{e}}_i \in \mathbb{R}^{64}$$

   Phương pháp SVD bảo toàn phương sai (explained variance ≈ 64.9%), giữ lại các thành phần ngữ nghĩa quan trọng nhất. Mặc dù mất ~35% thông tin, các chiều được

giữ lại là những chiều có phương sai cao nhất — tức là các đặc trưng ngữ nghĩa phân biệt nhất giữa các sản phẩm.

3. **Căn chỉnh** (Alignment): Do thứ tự sản phẩm trong BERT embeddings có thể khác với ma trận CSR, cần căn chỉnh theo ánh xạ `item_to_idx`. Các sản phẩm không có embedding BERT được khởi tạo bằng vector ngẫu nhiên từ phân phối của các embedding đã match (để duy trì consistency trong không gian tiềm ẩn).

4. **Gán làm Item Factors ban đầu**:

$$V^{(0)} = \text{AlignedBERTEmbeddings} \in \mathbb{R}^{|I| \times d}$$

   ALS sẽ fine-tune V trong quá trình huấn luyện thay vì học từ đầu.

Lợi ích BERT Initialization:

1. **Hội tụ nhanh hơn** (Faster Convergence): Thay vì bắt đầu từ một điểm ngẫu nhiên trong không gian tiềm ẩn, Item Factors khởi đầu đã mang ý nghĩa ngữ nghĩa. Điều này giúp thuật toán ALS đạt được local optimum tốt hơn với ít iterations hơn ($\sim$ 15 iterations thay vì 20–30).

2. **Neo giữ ngữ nghĩa** (Semantic Anchoring) cho Cold Items: Đây là lợi ích quan trọng nhất. Với các sản phẩm ít tương tác (cold items, < 5 interactions), việc học embedding từ dữ liệu hành vi gần như không khả thi do thiếu tín hiệu. Trong trường hợp khởi tạo ngẫu nhiên, vector của các cold items bị "trôi dạt" (drift) theo gradient từ các users thưa thớt, dẫn đến biểu diễn kém chất lượng.

   Với BERT Initialization, cold items được "neo giữ" vào không gian ngữ nghĩa: một sản phẩm "serum vitamin C dưỡng trắng" sẽ có embedding gần với các sản phẩm tương tự ngay từ đầu, bất kể nó có ít tương tác. Regularization cao ($\lambda = 0.1$) được áp dụng để ngăn không cho các vector này bị kéo quá xa khỏi vị trí khởi tạo có ý nghĩa.

3. **Transfer Learning từ NLP sang CF**: BERT Initialization thực chất là một dạng *transfer learning* — tri thức ngữ nghĩa học được từ corpus tiếng Việt lớn (qua pre-training của PhoBERT) được chuyển giao sang bài toán collaborative filtering. Điều này đặc biệt có giá trị khi dữ liệu tương tác thưa nhưng dữ liệu văn bản mô tả sản phẩm phong phú.

4.1.3 Cấu hình Hyperparameters

Các siêu tham số (hyperparameters) được điều chỉnh phù hợp với đặc thù dữ liệu:

Tham số	Giá trị	Giải thích
`factors`	64	Số chiều không gian tiềm ẩn
`regularization`	0.1	L2 penalty — cao hơn bình thường để anchor BERT embeddings cho sparse users
`iterations`	15	Số vòng lặp ALS
`alpha`	5	Confidence scaling — thấp do confidence range $[1, 6]$ thay vì binary
`random_state`	42	Seed cho reproducibility

Lưu ý về Alpha. Trong ALS cho implicit feedback, tham số α điều khiển mức độ "tin tưởng" vào các tương tác quan sát được:

$$C_{ui} = 1 + \alpha \cdot c_{ui}$$

Do hệ thống sử dụng confidence score trong khoảng $[1, 6]$ (đã có ý nghĩa mức độ tin cậy), α được đặt thấp ($\alpha = 5$) thay vì giá trị tiêu chuẩn ($\alpha = 40$) dùng cho dữ liệu binary.

4.1.4 Quy trình huấn luyện

Thuật toán ALS tối ưu hàm mất mát Weighted Regularized Squared Error theo công thức đã trình bày tại **mục 2.3.2**. Quá trình huấn luyện luân phiên (alternating) giữa cập nhật ma trận U (fix V) và ma trận V (fix U) trong 15 iterations cho đến khi hội tụ.
Thư viện `implicit` (C++ backend) được sử dụng để tăng tốc độ huấn luyện, đạt thời gian ≈ 4–5 giây cho toàn bộ 25,700 users × 1,400 items (trong 2200 sản phẩm thì chỉ có 1400 sản phẩm có được tương tác).

4.1.4.1 Lưu trữ Artifacts

Sau khi huấn luyện, các artifacts được lưu vào Model Registry:

```
artifacts/cf/als/v1_20251127/
 als_U.npy              # User factors (26000, 64)
 als_V.npy              # Item factors (2200, 64)
 als_params.json        # Hyperparameters
 als_metrics.json       # Recall@10, NDCG@10, baseline comparison
 als_metadata.json      # BERT init info, score_range, git commit
```

Score Range cho Global Normalization. File `metadata.json` chứa `score_range` - phân phối điểm CF trên validation set. Thông tin này được sử dụng trong Hybrid Reranking (mục 5.2) để chuẩn hóa CF scores về $[0, 1]$, đảm bảo so sánh công bằng với các signals khác.

4.2 Bayesian Personalized Ranking (BPR)

BPR (Bayesian Personalized Ranking) [5] là phương pháp học ranking theo cặp (pairwise ranking), được thiết kế đặc biệt cho bài toán gợi ý với implicit feedback. Thay vì dự đoán rating tuyệt đối như ALS, BPR tối ưu trực tiếp *thứ hạng tương đối* giữa các items - một mục tiêu phù hợp hơn với bản chất của bài toán gợi ý.

4.2.1 Cơ sở lý thuyết

Hệ thống áp dụng BPR (Bayesian Personalized Ranking) với hàm mục tiêu và công thức cập nhật SGD đã được trình bày chi tiết tại **mục 2.3.3**. Phần này tập trung vào cách code training pipeline thay vì lặp lại công thức toán học.

4.2.2 Đầu vào: Bộ ba huấn luyện (Training Triplets)

Đầu vào của BPR là tập các **bộ ba** (u, i, j) trong đó:

- $u \in \mathcal{U}_{\text{train}}$: Người dùng (chỉ trainable users).
- $i \in \mathcal{I}_u^+$: Item "positive" - sản phẩm user đã tương tác với $r_{ui} \geq 4$.
- $j \in \mathcal{I} \setminus \mathcal{I}_u^+$: Item "negative" - sản phẩm user chưa tương tác hoặc đã đánh giá thấp.

Ý nghĩa. Mỗi bộ ba (u, i, j) mang thông tin: "User u thích sản phẩm i hơn sản phẩm j". Mục tiêu huấn luyện: học embeddings sao cho predicted score $\hat{r}_{ui} > \hat{r}_{uj}$.

Số lượng mẫu. Với mỗi positive pair (u, i), hệ thống sinh ra `samples_per_positive` = 5 triplets (mỗi triplet với một negative j khác nhau). Tổng số triplets mỗi epoch:

$$|D_S| = |\text{positive\_pairs}| \times 5 \approx 30{,}000 \times 5 = 150{,}000 \text{ triplets}$$

4.2.3 Chiến lược Negative Sampling: Hard Negative Mining

Chiến lược chọn mẫu âm (negative sampling) đóng vai trò *quyết định* đến chất lượng của BPR. Chọn ngẫu nhiên hoàn toàn (uniform random) dẫn đến nhiều "easy negatives" - các sản phẩm hoàn toàn không liên quan mà mô hình dễ dàng phân biệt, gradient gần 0 và không đóng góp gì cho quá trình học.
Hệ thống áp dụng **Dual Hard Negative Mining** - một chiến lược kết hợp hai nguồn hard negatives:

Nguồn 1: Explicit Hard Negatives (từ rating thấp). Các sản phẩm user đã mua **nhưng đánh giá thấp** ($r_{uj} \leq 3$):

$$\mathcal{H}_u^{\text{explicit}} = \{j \in \mathcal{I}_u : r_{uj} \leq 3\}$$

Đây là tín hiệu mạnh nhất về "không thích" - user đã trải nghiệm thực tế và bày tỏ sự không hài lòng. Tuy nhiên, nguồn này thường khan hiếm do rating skew (chỉ $\approx 5\%$ ratings ≤ 3).

Nguồn 2: Implicit Hard Negatives (từ popularity). Các sản phẩm *phổ biến* (Top-50 theo `num_sold`) mà user **không** tương tác:

$$\mathcal{H}_u^{\text{implicit}} = \{j \in \text{Top50}_{\text{popular}} : j \notin \mathcal{I}_u\}$$

Logic: "Sản phẩm bán chạy nhưng bạn không quan tâm $\Rightarrow$ implicit negative signal". Nguồn này phong phú hơn và bổ sung cho explicit negatives.

Tỷ lệ mixing.

$$p(\text{hard}) = 0.3, \quad p(\text{random}) = 0.7$$

Với mỗi triplet, có 30% xác suất chọn negative từ $\mathcal{H}_u = \mathcal{H}_u^{\text{explicit}} \cup \mathcal{H}_u^{\text{implicit}}$, và 70% chọn ngẫu nhiên từ $\mathcal{I} \setminus \mathcal{I}_u^+$.
Tỷ lệ 30/70 được chọn dựa trên thực nghiệm: quá nhiều hard negatives ($> 50\%$) gây unstable training, quá ít ($< 20\%$) không đủ informative.

Fallback Strategy: Với users không có hard negatives (không có rating ≤ 3 và không có implicit negatives), hệ thống fallback về 100% random sampling. Statistics:

```
mixer.stats = {
    'hard_samples': 45000,      # 30% of 150K
    'random_samples': 105000,   # 70% of 150K
    'fallback_to_random': 4100  # Users without hard negs
}
```

Lợi ích của Hard Negative Mining.

1. **Gradient informativeness**: Hard negatives nằm gần decision boundary, tạo gradient lớn và có ý nghĩa. Easy negatives ($\hat{r}_{ui} \gg \hat{r}_{uj}$) cho $\sigma(x_{uij}) \approx 1$, gradient ≈ 0.

2. **Học biên quyết định tinh tế**: Mô hình buộc phải học các đặc trưng subtle để phân biệt "sản phẩm user thích" vs "sản phẩm phổ biến nhưng user không thích".

3. **Giảm popularity bias**: Implicit hard negatives từ popular items ngăn mô hình chỉ đơn giản gợi ý sản phẩm phổ biến cho tất cả users.

4. **Tận dụng explicit feedback**: Ratings thấp (≤ 3) — một nguồn thông tin quý giá nhưng thường bị bỏ qua trong implicit systems — được khai thác như hard negatives.

4.2.4 Quy trình huấn luyện: Stochastic Gradient Descent

BPR sử dụng SGD (Stochastic Gradient Descent) với mini-batch updates để tối ưu log-likelihood của ranking. Công thức gradient và update rules đã được trình bày chi tiết tại **mục 2.3.3**. Phần này mô tả cài đặt thực tế trong code.

Learning Rate Schedule. Áp dụng exponential decay để ổn định quá trình hội tụ: với $\eta_0 = 0.05$ (initial learning rate), $\gamma = 0.9$ (decay factor), $T = 10$ (decay interval).

4.2.5 Cấu hình Hyperparameters

Tham số	Giá trị	Giải thích
`factors`	64	Số chiều embedding (match với ALS)
`learning_rate`	0.05	Learning rate ban đầu
`regularization`	10^{-4}	L2 penalty — thấp hơn ALS do SGD
`epochs`	50	Số epochs tối đa
`samples_per_positive`	5	Số negatives mỗi positive
`hard_ratio`	0.3	Tỷ lệ hard negatives
`batch_size`	4096	Mini-batch size
`lr_decay`	0.9	Decay factor
`lr_decay_every`	10	Decay mỗi 10 epochs

Khởi tạo Embeddings. Khác với ALS sử dụng BERT Initialization, BPR khởi tạo cả U và V bằng **Random Gaussian** $\mathcal{N}(0, 0.01)$. Lý do: thực nghiệm cho thấy BERT Initialization không cải thiện đáng kể cho BPR (BPR học từ pairwise comparisons, ít phụ thuộc vào vị trí khởi tạo tuyệt đối như ALS).

4.2.6 Early Stopping và Checkpointing

Early Stopping. Theo dõi Recall@10 trên validation set mỗi epoch. Dừng huấn luyện nếu không cải thiện sau `patience` = 5 epochs liên tiếp:

$$\texttt{stop} = \left(\text{Recall@10}_t \leq \max_{s<t} \text{Recall@10}_s\right) \text{ for 5 consecutive } t$$

Checkpointing. Lưu U, V, metadata mỗi 5 epochs vào `checkpoints/bpr/`:

```
checkpoints/bpr/
 bpr_U_epoch010.npy
 bpr_V_epoch010.npy
 checkpoint_epoch010.json
 ...
```

Cho phép resume training từ checkpoint nếu bị interrupt, và rollback về best epoch sau khi early stopping.

4.3 Model Registry và quản lý phiên bản

Hệ thống Model Registry (Task 04) quản lý vòng đời của các model versions, hỗ trợ rollback, A/B testing, và audit trail.

4.3.1 Cấu trúc Registry

```
artifacts/cf/
 als/
    v1_20251125/
    v2_20251127/
 bpr/
    v1_20251126/
 registry.json          # Master registry file
```

File `registry.json` chứa:

- `current_best`: Model đang được serving (ID, path, metrics).
- `models`: Dictionary tất cả model versions với metadata đầy đủ.

4.3.2 Auto-Selection Best Model

Khi có model mới được huấn luyện, Registry tự động so sánh với `current_best`:

$$\texttt{promote} = \begin{cases} \text{True} & \text{nếu NDCG@10}_{\text{new}} > \text{NDCG@10}_{\text{current}} \\ \text{False} & \text{ngược lại} \end{cases}$$

Nếu promote, `current_best` được cập nhật và Serving layer nhận thông báo reload model mới (hot-reload, không downtime).

4.3.3 Metadata và Reproducibility

Mỗi model version lưu trữ metadata đầy đủ để đảm bảo reproducibility:

- `data_version`: Hash của dữ liệu huấn luyện (từ Task 01).
- `git_commit`: Phiên bản code tại thời điểm huấn luyện.
- `hyperparameters`: Tất cả config (factors, regularization, etc.).
- `metrics`: Kết quả evaluation trên test set.
- `score_range`: Phân phối CF scores cho normalization (Task 08).

Với metadata này, bất kỳ model version nào cũng có thể được reproduce chính xác bằng cách checkout đúng git commit và sử dụng đúng data version.

5. Hệ thống Serving và Hybrid Reranking

5.1 Kiến trúc Serving

Kiến trúc Serving chịu trách nhiệm chuyển đổi mô hình đã huấn luyện thành dịch vụ production-ready với ba yêu cầu bắt buộc: ***độ trễ*** $< 100ms$ *(P95),* ***tính sẵn sàng*** $\geq 99.9\%$*, và* ***throughput*** ≥ 100 *req/s. Phần này trình bày kiến trúc end-to-end từ khi nhận request đến khi trả về recommendations.*

Thống kê dữ liệu thực tế. Hệ thống được thiết kế để phục vụ tập dữ liệu mỹ phẩm Việt Nam với khoảng 300,000 users, 2,200 sản phẩm, và độ thưa dữ liệu cực kỳ cao. Thống kê chi tiết và phân khúc người dùng được trình bày tại **mục 3.2** và **mục 6**.

Luồng dữ liệu tổng quan (Data Flow)

Mỗi request đi qua 5 giai đoạn tuần tự với latency budget được phân bổ rõ ràng:

```
[Client] --> [API Gateway] --> [User Router] --> [Scoring Engine] --> [Response]
  Input         5ms               10ms             50-80ms             5ms
```

Hình 5.1: Phân rã độ trễ (Latency Breakdown) của một yêu cầu gợi ý.

1. **API Gateway** (5ms): Validation, authentication, rate limiting.
2. **User Router** (10ms): Phân loại user segment, quyết định CF hoặc Fallback path.
3. **Scoring Engine** (50–80ms): Tính điểm CF hoặc content-based, áp dụng filters.
4. **Reranking** (tích hợp trong Scoring): Hybrid scoring nếu enabled.
5. **Response Assembly** (5ms): Enrich metadata, serialize JSON.

Tổng latency budget: $\leq$ 100ms. Hệ thống đảm bảo luôn trả về kết quả thông qua cơ chế multi-layer fallback được mô tả ở phần sau.

Kiến trúc phân tầng (Layered Architecture)

Hệ thống được tổ chức thành 4 tầng với separation of concerns rõ ràng:

```
service/
|-- api.py                # L1: API Layer - Entry point
```

```
|-- recommender/          # L2: Business Logic Layer
|   |-- loader.py         #     Data Access (Singleton)
|   |-- recommender.py    #     Core Engine
|   |-- fallback.py       #     Cold-start Handler
|   |-- rerank.py         #     Hybrid Reranker
|   +-- cache.py          #     Performance Optimization
|-- search/               # L3: Search Layer (xem mục 5.3)
+-- config/               # L4: Configuration Layer
```

Nguyên tắc thiết kế.

- **Single Responsibility**: Mỗi module xử lý đúng một concern.
- **Dependency Injection**: Loader được inject vào Recommender, Fallback, Reranker.
- **Fail-fast với graceful degradation**: Phát hiện lỗi sớm, fallback có kiểm soát.

API Layer: FastAPI Service

FastAPI được chọn làm framework chính với các đặc điểm sau:

- **Async I/O**: Xử lý concurrent requests hiệu quả (non-blocking).
- **Type Safety**: Pydantic validation đảm bảo input/output consistency.
- **Auto-documentation**: OpenAPI spec tự động sinh cho integration testing.

Endpoints và SLA.

Endpoint	Method	Latency Target	Mô tả
`/health`	GET	< 10ms	Health check, readiness probe
`/recommend`	POST	< 100ms	Single-user recommendation
`/batch_recommend`	POST	< 500ms	Multi-user batch processing
`/similar_items`	POST	< 50ms	Item-item similarity
`/reload_model`	POST	< 5s	Hot-reload model từ registry
`/search`	POST	< 200ms	Semantic search (mục 5.3)

Lifecycle Management. Service lifecycle được quản lý qua `lifespan` context manager:
Startup sequence đảm bảo:

- Model U, V được load vào memory trước khi nhận traffic.
- Cache được warm-up với Top-50 popular items và pre-computed similarities.

- Vietnamese Embedding model (`AITeamVN/Vietnamese_Embedding`) được load sẵn (tránh latency 30s ở query đầu tiên).

Security Hardening. Hệ thống áp dụng defense-in-depth với 4 lớp bảo vệ:

1. **CORS Policy**: Whitelist origins trong production.
2. **Rate Limiting**: 100 req/s per IP (configurable).
3. **Request Size Limit**: Max 10MB body size.
4. **Security Headers**: HSTS, X-Frame-Options, X-Content-Type-Options.

Model Management: Hot-Reload và Registry

Hệ thống hỗ trợ zero-downtime model updates thông qua hot-reload mechanism.

Model Registry. Registry là JSON file quản lý tất cả model versions:

```
{
  "current_best": {
    "model_id": "als_v2_20251127",
    "path": "artifacts/cf/als/20251127/",
    "ndcg@10": 0.342,
    "recall@10": 0.285
  },
  "models": { ... }
}
```

Singleton Model Loader. Class `CFModelLoader` được implement theo Singleton pattern để đảm bảo:

- Chỉ một instance trong memory (tiết kiệm RAM).
- Thread-safe access với `threading.Lock`.
- Shared state giữa API, Recommender, và Reranker.

```
class CFModelLoader:
    _instance = None
    _lock = threading.Lock()

    def __new__(cls, *args, **kwargs):
        with cls._lock:
```

```
        if cls._instance is None:
            cls._instance = super().__new__(cls)
        return cls._instance
```

Hot-Reload Protocol. Khi nhận request `POST /reload_model`:

1. **Check**: So sánh `registry.current_best.model_id` với model đang serve.
2. **Load**: Nếu khác, tải U', V', metadata từ path mới.
3. **Validate**: Verify matrix dimensions và score_range.
4. **Swap**: Atomic replacement $(U, V) \leftarrow (U', V')$.
5. **Confirm**: Log model_id mới, return success response.

Điều kiện trigger reload:

$$\texttt{need\_reload} = \big(\texttt{registry.current\_best.model\_id} \neq \texttt{loader.current\_model\_id}\big)$$

User Segmentation Router

Router là thành phần quyết định luồng xử lý dựa trên đặc điểm người dùng. Phân khúc người dùng chi tiết được trình bày tại **mục 3.2**.

Định nghĩa User Segments. Hệ thống sử dụng hai phân khúc người dùng với chiến lược xử lý khác nhau. Chi tiết về phân khúc, ngưỡng, và thống kê được trình bày tại **mục 3.2**. Tóm tắt logic routing:

- **Trainable Users** ($|\mathcal{H}_u| \geq 2$ và có ≥ 1 positive): CF Scoring → Hybrid Reranking
- **Cold-Start Users** (ngược lại): Content-based → Popularity Blend

Routing Decision Function. Biểu diễn hình thức:

$$\texttt{path}(u) = \begin{cases} \text{CF} & \text{nếu } |\mathcal{H}_u| \geq 2 \wedge \exists\, i \in \mathcal{H}_u : r_{ui} \geq 4 \\ \text{Fallback} & \text{ngược lại} \end{cases}$$

CF Path: Scoring Pipeline cho Trainable Users

Với trainable users đi qua CF path, pipeline xử lý gồm 6 bước:

Bước 1: Index Mapping. Ánh xạ user ID gốc sang CF matrix index:

$$u \xrightarrow{\texttt{trainable\_user\_mapping}} u_{\text{idx\_cf}}$$

Bước 2: CF Scoring. Điểm ưu tiên dự đoán (predicted preference score) cho item i được định nghĩa là tích vô hướng:

$$\hat{r}_{ui} = \mathbf{u}_{u_{\text{idx\_cf}}}^{\top} \mathbf{v}_i, \quad \forall i \in \{1, \ldots, |\mathcal{I}|\}$$

trong đó $\mathbf{u} \in \mathbb{R}^{64}$ là user embedding (hàng tương ứng trong ma trận U), $\mathbf{v} \in \mathbb{R}^{64}$ là item embedding (hàng tương ứng trong ma trận V), với $|\mathcal{I}| \approx 2{,}200$.
Implementation: Matrix-vector multiplication $\mathbf{u} \cdot V^{\top}$ với complexity $O(d \cdot |\mathcal{I}|) = O(64 \times 2{,}200)$.

Bước 3: Seen-Item Filtering. Để tránh gợi ý lại items đã mua, áp dụng masking:

$$\hat{r}_{ui} \leftarrow -\infty, \quad \forall i \in \mathcal{H}_u^{(\text{train})}$$

Lưu ý: Chỉ filter train set để tránh data leakage khi evaluation.

Bước 4: Attribute Filtering (Optional). Nếu request có filter_params (brand, category, price range), thu hẹp không gian ứng viên:

$$\hat{r}_{ui} \leftarrow -\infty, \quad \forall i \notin \mathcal{I}_{\mathcal{F}}$$

trong đó $\mathcal{I}_{\mathcal{F}}$ là tập items thỏa mãn tất cả filter conditions.

Bước 5: Top-K Selection. Chọn K items có score cao nhất:

$$C_u = \underset{i \in \mathcal{I}}{\arg \text{topK}} \, \hat{r}_{ui}$$

Nếu reranking enabled: chọn 5000 candidates để có đủ diversity cho reranking.

Bước 6: Hybrid Reranking. Áp dụng Hybrid Reranking (mục 5.2) để kết hợp CF score với content, popularity, quality signals. Output cuối cùng: top-K recommendations.

Fallback Path: Content-based cho Cold-Start Users

Với cold-start users (chiếm đa số traffic), đây là critical path cần tối ưu hiệu năng cao nhất.

Bước 1: User History Retrieval. Lấy lịch sử tương tác (nếu có): $\mathcal{H}_u = \{i_1, \ldots, i_m\}$ với $m \in \{0, 1\}$.

Bước 2: User Profile Construction. Nếu $m \geq 1$, user profile embedding được định nghĩa là trung bình có trọng số của item embeddings (Sử dụng mô hình Vietnamese Embedding 1024 chiều).

$$\mathbf{e}_u = \sum_{k=1}^{m} \alpha_k \tilde{\mathbf{e}}_{i_k}, \quad \tilde{\mathbf{e}}_u = \frac{\mathbf{e}_u}{\|\mathbf{e}_u\|_2}$$

với $\mathbf{e}_{i_k} \in \mathbb{R}^{1024}$, $\alpha_k \geq 0$, $\sum_k \alpha_k = 1$ (uniform hoặc recency-weighted). Nếu $m = 0$ (new user): bỏ qua content scoring, chỉ dùng popularity.

Bước 3: Content-based Scoring. Điểm content-based được định nghĩa là cosine similarity giữa embeddings 1024 chiều (Sử dụng mô hình Vietnamese Embedding 1024 chiều như đã mô tả tại):

$$s_{\text{content}}(u, i) = \cos(\tilde{\mathbf{e}}_u, \tilde{\mathbf{e}}_i) = \tilde{\mathbf{e}}_u^\top \tilde{\mathbf{e}}_i$$

Bước 4: Popularity Scoring. Điểm popularity được chuẩn hóa logarithmic từ số lượng bán:

$$s_{\text{pop}}(i) = \frac{\log(1 + \texttt{num\_sold}_i)}{\max_j \log(1 + \texttt{num\_sold}_j)} \in [0, 1]$$

Bước 5: Hybrid Fallback Score. Điểm tổng hợp là tổ hợp tuyến tính:

$$S_{\text{fallback}}(u, i) = w_{\text{content}} \cdot s_{\text{content}}(u, i) + w_{\text{pop}} \cdot s_{\text{pop}}(i) + w_{\text{quanlity}} \cdot s_{\text{quanlity}}(i)$$

Giá trị mặc định: $w_{\text{content}} = 0.6$, $w_{\text{pop}} = 0.3$ $w_{\text{quanlity}} = 0.1$.
Cho new users ($m = 0$): $w_{\text{content}} = 0.6$, $w_{\text{pop}} = 0.3$, $w_{\text{quanlity}} = 0.1$, $s_{\text{content}} = 0.5$.

Bước 6: Top-K Selection.

$$\mathcal{R}_u = \underset{i \in \mathcal{I}}{\arg\,\text{topK}}\, S_{\text{fallback}}(u, i)$$

Performance Optimization

LRU Caching Strategy. Với đa số traffic qua fallback path (cold-start users), caching là yếu tố quyết định hiệu năng:

- **Popular Items Cache**: Pre-compute Top-50 popular items với enriched metadata.
- **User Profile Cache**: LRU cache cho computed user profiles (TTL: 1 hour).
- **Similarity Cache**: Pre-compute $50 \times 50 = 2{,}500$ item-item similarities.

Fault Tolerance: Multi-Layer Fallback

Hệ thống đảm bảo luôn trả về kết quả thông qua cascading fallback:

1. **CF index miss**: → Fallback sang content-based.
2. **Vietnamese Embedding unavailable**: → Fallback sang popularity-only.
3. **No valid items after filtering**: → Bỏ qua filters, return popularity.
4. **Model not loaded**: → HTTP 503 Service Unavailable.

Response luôn chứa observability metadata:

```
{
  "user_id": 12345,
  "recommendations": [...],
  "is_fallback": true,
  "fallback_method": "hybrid",
  "latency_ms": 45.2,
  "model_id": "als_v2_20251127"
}
```

Monitoring và SLA Metrics

Performance Metrics.

- **Latency quantiles**: P50, P90, P95, P99 cho mỗi endpoint.
- **Throughput**: Requests/second, aggregated per minute.
- **Error rate**: 4xx, 5xx responses per endpoint.

Business Metrics.

- **Fallback ratio**: $\frac{\text{\# fallback requests}}{\text{\# total requests}}$ (target: $\approx 91.4\%$).
- **Rerank ratio**: Percentage of requests with reranking enabled.

SLA Targets.

Metric	Target	Alert Threshold
CF Path Latency (P95)	< 50ms	> 80ms
Fallback Path Latency (P95)	< 100ms	> 150ms
Throughput	≥ 100 req/s	< 50 req/s
Error Rate	$< 0.1\%$	$> 1\%$
Availability	$\geq 99.9\%$	$< 99\%$

Kết nối với các thành phần khác

Luồng integration:

```
[Serving: CF/Fallback] --> [Reranking: Multi-signal] --> [Response]
                                        ^
[Smart Search] ----------------------+
```

Cả 3 thành phần chia sẻ: Vietnamese Embedding (load một lần, 1024 chiều), item metadata, caching infrastructure, và monitoring pipeline.

5.2 Thuật toán Hybrid Reranking

Hybrid Reranking là bước xử lý cuối cùng trong pipeline gợi ý, có nhiệm vụ tích hợp nhiều tín hiệu đánh giá (signals) để tối ưu đồng thời relevance và diversity của kết quả trả về. Phần này trình bày hình thức toán học của cơ chế reranking, bao gồm định nghĩa các signals, hàm scoring, và phương pháp đa dạng hóa.

Ký hiệu và định nghĩa cơ bản

Cho trước:

- $\mathcal{U}$: tập người dùng, $|\mathcal{U}| \approx 300{,}000$.
- $\mathcal{I}$: tập sản phẩm, $|\mathcal{I}| \approx 2{,}200$.
- $\mathcal{H}_u \subseteq \mathcal{I}$: tập sản phẩm user u đã tương tác (lịch sử).
- $C_u \subseteq \mathcal{I} \setminus \mathcal{H}_u$: tập candidates từ CF hoặc Fallback (mục 5.1).

[Trainable User] User u được gọi là **trainable** nếu thỏa mãn:

$$|\mathcal{H}_u| \geq 2 \quad \text{và} \quad \exists\, i \in \mathcal{H}_u : r_{ui} \geq \theta_{\text{pos}}$$

với $\theta_{\text{pos}} = 4$ là ngưỡng positive interaction. Ký hiệu tập trainable users là $\mathcal{U}_{\text{train}}$.
[Cold-Start User] User $u \in \mathcal{U} \setminus \mathcal{U}_{\text{train}}$ được gọi là **cold-start user**.
Với $|\mathcal{U}_{\text{train}}| \approx 26{,}000$ (chiếm 8.6%), ta có phân hoạch $\mathcal{U} = \mathcal{U}_{\text{train}} \sqcup \mathcal{U}_{\text{cold}}$.

Định nghĩa các Signals

Hybrid Reranking tích hợp 4 tín hiệu độc lập. Mỗi signal được chuẩn hóa về đoạn $[0, 1]$.

Signal 1: CF Score (s_{cf}). Cho $u \in \mathcal{U}_{\text{train}}$ với embedding $\mathbf{u} \in \mathbb{R}^d$ và item i với embedding $\mathbf{v}_i \in \mathbb{R}^d$:

$$\hat{r}_{ui} = \mathbf{u}^\top \mathbf{v}_i$$

Điểm CF được chuẩn hóa min-max:

$$s_{\text{cf}}(u, i) = \frac{\hat{r}_{ui} - r_{\min}}{r_{\max} - r_{\min}} \in [0, 1]$$

trong đó $r_{\min}, r_{\max}$ được xác định trước từ training set hoặc lấy từ metadata (`score_range` trong model artifacts).
Cho cold-start users ($u \in \mathcal{U}_{\text{cold}}$): $s_{\text{cf}}(u, i) = 0, \forall i$.

Signal 2: Content Score (s_{content}). Mỗi sản phẩm i được biểu diễn bởi **Vietnamese Embedding** (`AITeamVN/Vietnamese_Embedding`) $\mathbf{e}_i \in \mathbb{R}^{1024}$, đã được chuẩn hóa L2: $\|\mathbf{e}_i\|_2 = 1$.
User profile embedding được tính bằng một trong ba strategies:

$$\mathbf{e}_u^{(\text{mean})} = \frac{1}{|\mathcal{H}_u|} \sum_{i \in \mathcal{H}_u} \mathbf{e}_i$$

$$\mathbf{e}_u^{(\text{weighted})} = \sum_{i \in \mathcal{H}_u} \alpha_i \mathbf{e}_i, \quad \alpha_i = \frac{r_{ui}}{\sum_{j \in \mathcal{H}_u} r_{uj}}$$

$$\mathbf{e}_u^{(\max)} = \max_{i \in \mathcal{H}_u} \mathbf{e}_i \quad \text{(element-wise max)}$$

Sau đó chuẩn hóa: $\tilde{\mathbf{e}}_u = \mathbf{e}_u / \|\mathbf{e}_u\|_2$. Implementation hiện tại sử dụng `weighted_mean` với trọng số theo rating.
Điểm content là cosine similarity:

$$s_{\text{content}}(u, i) = \cos(\tilde{\mathbf{e}}_u, \mathbf{e}_i) = \tilde{\mathbf{e}}_u^\top \mathbf{e}_i \in [-1, 1]$$

Rescale về $[0, 1]$:

$$s^*_{\text{content}}(u, i) = \frac{s_{\text{content}}(u, i) + 1}{2}$$

Cho users với $\mathcal{H}_u = \emptyset$: $s^*_{\text{content}}(u, i) = 0.5$ (neutral).

Signal 3: Popularity Score (s_{pop}). Điểm popularity đo mức độ phổ biến của sản phẩm, độc lập với user:

$$s_{\text{pop}}(i) = \frac{\log(1 + \texttt{num\_sold}_i)}{\max_{j \in \mathcal{I}} \log(1 + \texttt{num\_sold}_j)}$$

Logarithm được sử dụng để giảm ảnh hưởng của outliers (sản phẩm bán chạy cực kỳ).

Kết quả đã nằm trong $[0, 1]$ do định nghĩa.

Signal 4: Quality Score (s_{quality}). Điểm quality kết hợp rating trung bình và số lượng reviews:

$$s_{\text{quality}}(i) = \beta \cdot \frac{\bar{r}_i - 1}{4} + (1 - \beta) \cdot \frac{\log(1 + n_i)}{\max_j \log(1 + n_j)}$$

trong đó:

- $\bar{r}_i \in [1, 5]$: rating trung bình của item i.
- n_i: số lượng reviews của item i.
- $\beta \in [0, 1]$: trọng số cân bằng giữa rating và review count (mặc định $\beta = 0.6$).

Số hạng đầu chuẩn hóa rating về $[0, 1]$; số hạng sau đo độ tin cậy thống kê.

Hàm Hybrid Scoring

[Hybrid Score Function] Cho user u và item $i \in C_u$, điểm hybrid được định nghĩa:

$$S(u, i) = w_{\text{cf}} \cdot s_{\text{cf}}(u, i) + w_{\text{content}} \cdot s^*_{\text{content}}(u, i) + w_{\text{pop}} \cdot s_{\text{pop}}(i) + w_{\text{quality}} \cdot s_{\text{quality}}(i)$$

trong đó vector trọng số $\mathbf{w} = (w_{\text{cf}}, w_{\text{content}}, w_{\text{pop}}, w_{\text{quality}})^\top$ thỏa mãn:

$$\mathbf{w} \geq \mathbf{0}, \quad \|\mathbf{w}\|_1 = 1$$

Kết quả: $S(u, i) \in [0, 1]$ do mỗi signal đã được chuẩn hóa và $\|\mathbf{w}\|_1 = 1$.

Segment-Specific Weight Vectors. Do sự khác biệt về độ tin cậy của CF signal giữa hai segments, áp dụng weight vectors riêng:
Trainable Users ($u \in \mathcal{U}_{\text{train}}$, chiếm ~8.6% traffic):

$$\mathbf{w}^{(\text{train})} = (0.30, 0.40, 0.20, 0.10)^\top$$

Content được ưu tiên hơn CF ($w_{\text{content}} > w_{\text{cf}}$) do:

- Vietnamese Embedding (1024 dim) capture ngữ nghĩa tiếng Việt tốt hơn.
- CF embeddings (64 dim) bị hạn chế bởi sparsity (~1.23 interactions/user).
- Đây là **content-first strategy** phù hợp với dữ liệu cosmetics Việt Nam.

Cold-Start Users ($u \in \mathcal{U}_{\text{cold}}$, chiếm ~91.4% traffic):

$$\mathbf{w}^{(\text{cold})} = (0, 0.60, 0.30, 0.10)^\top$$

CF bị vô hiệu hóa ($w_{\text{cf}} = 0$) vì không có embeddings. Content chiếm 60% để tận dụng Vietnamese Embedding similarity. Popularity (30%) đảm bảo items được gợi ý là những sản phẩm đã được thị trường verify.

Đa dạng hóa kết quả (Diversity)

Chỉ tối ưu relevance ($S(u, i)$) có thể dẫn đến filter bubble: top-K items quá tương đồng nhau. Hệ thống áp dụng diversity constraint qua thuật toán greedy re-selection.
[Intra-List Diversity] Cho danh sách gợi ý $\mathcal{R} = \{i_1, \ldots, i_K\}$, độ đa dạng được đo bằng:

$$\text{ILD}(\mathcal{R}) = \frac{2}{K(K-1)} \sum_{j<k} \left(1 - \cos(\mathbf{e}_{i_j}, \mathbf{e}_{i_k})\right)$$

Giá trị cao cho thấy các items trong list có nội dung đa dạng.

Maximal Marginal Relevance (MMR). Để cân bằng relevance và diversity, sử dụng MMR criterion:

$$i^* = _{i \in C \setminus \mathcal{R}} \left[\lambda \cdot S(u, i) - (1 - \lambda) \cdot \max_{j \in \mathcal{R}} \cos(\mathbf{e}_i, \mathbf{e}_j)\right]$$

trong đó:

- $\lambda \in [0, 1]$: trade-off parameter (mặc định $\lambda = 0.7$, ưu tiên relevance).
- Số hạng đầu: relevance score.
- Số hạng sau: penalty cho similarity với items đã chọn.

Thuật toán Greedy MMR.

1. Khởi tạo: $\mathcal{R} \leftarrow \emptyset$.
2. Chọn item đầu tiên: $i_1 = _{i \in C} S(u, i)$, thêm vào $\mathcal{R}$.
3. Lặp $K - 1$ lần: chọn i^* theo MMR criterion, thêm vào $\mathcal{R}$.
4. Trả về $\mathcal{R} = \{i_1, \ldots, i_K\}$.

Complexity: $O(K \cdot |C| \cdot d)$ với $d = 1024$ (Vietnamese Embedding dimension).

Configuration từ `rerank_config.yaml`.

- `diversity_enabled`: `true` (mặc định bật).
- `diversity_penalty`: $\gamma = 0.10$ (penalty cho similarity).

- `diversity_threshold`: $\tau = 0.85$ (chỉ penalize nếu cos $> \tau$).
- Candidate pool: $|C| \approx 100$ items để MMR có đủ lựa chọn.

Điều chỉnh MMR với threshold:

$$i^* = _{i \in C \setminus \mathcal{R}} \left[S(u,i) - \gamma \cdot 1\big[\max_{j \in \mathcal{R}} \cos(\mathbf{e}_i, \mathbf{e}_j) > \tau \big] \right]$$

Chỉ áp dụng penalty khi có item quá tương đồng (cosine > 0.85), tránh penalize không cần thiết.

Position Bias Correction (Optional)

Trong thực tế, items ở vị trí cao hơn nhận được nhiều clicks hơn do position bias. Để học được true relevance, cần correction.
[Position Bias Model] Giả sử probability user click item tại position k:

$$P(\text{click} \mid u, i, k) = P(\text{examine} \mid k) \cdot P(\text{click} \mid u, i, \text{examined})$$

trong đó $P(\text{examine} \mid k)$ giảm theo k (position bias).

Inverse Propensity Scoring (IPS). Để debias, áp dụng trọng số nghịch đảo:

$$w_k = \frac{1}{P(\text{examine} \mid k)} \approx k^{\gamma}$$

với $\gamma \in [0.5, 1.5]$ được estimate từ randomization experiments.
Lưu ý: Feature này là optional và chưa được implement trong phiên bản hiện tại. Được thiết kế để tích hợp khi có đủ click-through data từ production.

Pipeline tổng hợp

Tóm tắt luồng xử lý Hybrid Reranking:

```
Input: user u, candidates C_u (from Serving, mục 5.1)
       rerank_config: {weights, diversity, K}

1. Compute signals:
   s_cf[i]      <- CF score (or 0 if cold-start)
   s_content[i] <- Cosine(user_profile, item_embedding)
                   # user_profile via 'weighted_mean' strategy
   s_pop[i]     <- log-normalized popularity from num_sold_time
   s_quality[i] <- 0.6*rating_norm + 0.4*review_confidence
```

```
2. Select weight vector based on user segment:
   if u in U_train:  # ~8.6% traffic
       w <- (0.30, 0.40, 0.20, 0.10)
   else:             # ~91.4% traffic (cold-start)
       w <- (0.00, 0.60, 0.30, 0.10)

3. Compute hybrid scores:
   S[i] <- w · [s_cf[i], s_content[i], s_pop[i], s_quality[i]]

4. Diversity-aware selection:
   if diversity_enabled and max_cos(i, R) > 0.85:
       S[i] <- S[i] - 0.10  # penalty
   R <- TopK(S, K)

Output: R = {i_1, ..., i_K} với scores S(u, i_k)
```

Traffic Distribution Analysis. Với phân bố users hiện tại:

- $|\mathcal{U}_{\text{train}}| \approx 26{,}000$ (8.6%): Hybrid scoring đầy đủ 4 signals.
- $|\mathcal{U}_{\text{cold}}| \approx 274{,}000$ (91.4%): Content-first với 3 signals.

Bottleneck analysis: Cold-start path chiếm 91.4% traffic nhưng có latency cao hơn. Vietnamese Embedding similarity là bước tốn thời gian nhất do $d = 1024$.

5.3 Smart Search Integration

Smart Search là module tìm kiếm ngữ nghĩa (semantic search) cho phép người dùng tìm sản phẩm bằng truy vấn tiếng Việt tự nhiên. Phần này trình bày kiến trúc encoder, thuật toán tìm kiếm, và cơ chế kết hợp với Hybrid Reranking để đảm bảo kết quả vừa relevant với query vừa personalized cho user.

Ký hiệu và định nghĩa

Cho trước:

- $\mathcal{I}$: tập sản phẩm, $|\mathcal{I}| \approx 2{,}200$.
- $q \in \mathcal{Q}$: truy vấn tìm kiếm (chuỗi văn bản tiếng Việt).
- $\mathcal{V}$: vocabulary của tokenizer ($|\mathcal{V}| \approx 250{,}000$ cho Vietnamese Embedding).

- $d = 1024$: dimension của **Vietnamese Embedding** space.

[Product Text Representation] Mỗi sản phẩm $i \in \mathcal{I}$ được biểu diễn bởi chuỗi văn bản:

$$t_i = \texttt{[CLS]} \oplus \text{name}_i \oplus \texttt{[SEP]} \oplus \text{ingredient}_i \oplus \texttt{[SEP]} \oplus \text{feature}_i \oplus \texttt{[SEP]} \oplus \text{description}_i$$

trong đó $\oplus$ là phép nối chuỗi, `[CLS]` và `[SEP]` là special tokens.

Query Encoder với Vietnamese Embedding

Hệ thống sử dụng **Vietnamese Embedding** [7] (`AITeamVN/Vietnamese_Embedding`) — mô hình pre-trained cho tiếng Việt với 1024 dimensions, được tối ưu cho semantic similarity tasks.

Query Preprocessing. Trước khi encode, queries được tiền xử lý:

1. **Lowercase và strip whitespace**.
2. **Abbreviation expansion**: Mở rộng viết tắt tiếng Việt phổ biến.
3. **Normalize whitespace** và remove special characters.

Bảng abbreviations (trích):

Viết tắt	Mở rộng	Viết tắt	Mở rộng
kcn	kem chống nắng	srm	sữa rửa mặt
dn	da nhờn	dk	da khô
dd	dưỡng da	tdc	tẩy da chết
nht	nước hoa hồng	dhh	da hỗn hợp

Tokenization. Với input text t, tokenizer thực hiện:

1. SentencePiece/BPE tokenization cho tiếng Việt.
2. Output: sequence of token IDs $\mathbf{x} = (x_1, \ldots, x_n)$ với $x_j \in \{1, \ldots, |\mathcal{V}|\}$.

Maximum sequence length: $n_{\max} = 256$ tokens (truncate nếu vượt quá).

Encoding Function. Định nghĩa encoder function $\phi : \mathcal{T} \to \mathbb{R}^{1024}$:

$$\phi(t) = \text{MeanPooling}(\text{VietnameseEmbedding}(t))$$

trong đó mean pooling lấy trung bình các hidden states (trừ padding tokens), thay vì chỉ dùng `[CLS]` token.

Normalized Embedding. Để sử dụng cosine similarity hiệu quả, embeddings được L2-normalize:

$$\mathbf{e}_i = \frac{\phi(t_i)}{\|\phi(t_i)\|_2} \in \mathbb{R}^{1024}, \quad \|\mathbf{e}_i\|_2 = 1$$

Tương tự cho query: $\mathbf{e}_q = \phi(q)/\|\phi(q)\|_2$.

Query Embedding Cache (LRU). Để giảm latency cho repeated queries, hệ thống sử dụng LRU cache:

- **Capacity**: 1,000 query embeddings.
- **Cache key**: `preprocessed_query + normalize_flag`.
- **Expected hit rate**: ~30% trong production (users thường search các terms phổ biến).

Khi cache hit, encoding time giảm từ ~40ms xuống < 1ms.

Semantic Similarity

[Query-Item Similarity] Cho query q và item i, độ tương đồng ngữ nghĩa được định nghĩa:

$$\text{sim}(q, i) = \cos(\mathbf{e}_q, \mathbf{e}_i) = \mathbf{e}_q^\top \mathbf{e}_i \in [-1, 1]$$

Do embeddings đã normalize, tích vô hướng chính là cosine similarity.

Tính chất.

- $\text{sim}(q, i) = 1$ khi $\mathbf{e}_q = \mathbf{e}_i$ (identical semantics).
- $\text{sim}(q, i) = 0$ khi $\mathbf{e}_q \perp \mathbf{e}_i$ (orthogonal, unrelated).
- $\text{sim}(q, i) = -1$ khi $\mathbf{e}_q = -\mathbf{e}_i$ (hiếm gặp trong practice).

Rescaling. Để đồng bộ với các signals khác trong Hybrid Reranking (mục 5.2), rescale về $[0, 1]$:

$$s_{\text{search}}(q, i) = \frac{\text{sim}(q, i) + 1}{2} \in [0, 1]$$

Search Index và Approximate Nearest Neighbor

Naive search có complexity $O(|\mathcal{I}| \cdot d)$ cho mỗi query. Với $|\mathcal{I}| = 2{,}200$ và $d = 1024$, đây là acceptable (≈ 2.25M operations). Hệ thống hỗ trợ cả exact search và FAISS ANN.

Exact Search (Default). Với $|\mathcal{I}| < 5{,}000$, exact search đủ nhanh:

$$\text{similarities} = \mathbf{E}_{\text{norm}} \cdot \mathbf{e}_q$$

trong đó $\mathbf{E}_{\text{norm}} \in \mathbb{R}^{|\mathcal{I}| \times 1024}$ là ma trận embeddings đã normalize.
Complexity: $O(|\mathcal{I}| \cdot d) \approx 2.25$M operations, latency < 5ms.

FAISS Index (Optional). Hệ thống hỗ trợ 3 loại FAISS index cho large catalogs:
1. IndexFlatIP (default khi enable FAISS):

- Exact search với Inner Product (= cosine cho normalized vectors).
- Phù hợp cho $|\mathcal{I}| < 10{,}000$.

2. IndexIVFFlat (cho 10K–1M items):

- Sử dụng nlist = $\sqrt{|\mathcal{I}|}$ clusters.
- Cần training step trước khi add vectors.
- `nprobe = 10`: số clusters search (trade-off accuracy/speed).

3. IndexHNSWFlat (fast approximate):

- Hierarchical Navigable Small World graph.
- $M = 32$ connections, `efConstruction = 40`, `efSearch = 16`.
- Nhanh nhất nhưng approximate.

Metadata Inverted Indices. Ngoài embedding search, hệ thống xây dựng inverted indices cho filtering:

- `brand_index`: brand $\rightarrow$ {product_ids}.
- `category_index`: category $\rightarrow$ {product_ids}.
- `price_data`: product_id $\rightarrow$ price.

Filtering strategy: **Post-filter** (search trước, filter sau) thay vì pre-filter (xây index riêng cho mỗi filter combination).

Multi-Signal Reranking trong Search

Kết quả search thuần túy có thể không phù hợp với preferences của user cụ thể. Hệ thống áp dụng multi-signal reranking với 4 tín hiệu.
[Search Reranking Score] Cho query q và item i, điểm reranking được định nghĩa:

$$S_{\text{rerank}}(q,i) = w_{\text{sem}} \cdot s_{\text{search}}(q,i) + w_{\text{pop}} \cdot s_{\text{pop}}(i) + w_{\text{quality}} \cdot s_{\text{quality}}(i) + w_{\text{recency}} \cdot s_{\text{recency}}(i)$$

trong đó (từ `smart_search.py`):

$$\mathbf{w} = (w_{\text{sem}}, w_{\text{pop}}, w_{\text{quality}}, w_{\text{recency}}) = (0.50, 0.25, 0.15, 0.10)^\top$$

Signal Computation.

- $s_{\text{search}}(q,i)$: Semantic similarity (rescaled to $[0,1]$).
- $s_{\text{pop}}(i) = \min\left(\frac{\log(1+\texttt{num\_sold}_i)}{\log(1+100{,}000)}, 1\right)$: Log-normalized popularity.
- $s_{\text{quality}}(i) = \frac{\bar{r}_i - 1}{4}$: Rating normalized to $[0,1]$.
- $s_{\text{recency}}(i) = 0.5$ (placeholder, cần product launch date).

Candidate Expansion. Để reranking có đủ candidates, hệ thống fetch $3K$ candidates trước khi rerank và trả về top-K (với `candidate_multiplier = 3`).

Attribute Filtering

Ngoài semantic matching, users có thể filter theo attributes.

Filter Types.

- **Category**: $\mathcal{F}_{\text{cat}} \subseteq \{\text{“skincare”}, \text{“makeup”}, \dots\}$.
- **Brand**: $\mathcal{F}_{\text{brand}} \subseteq \{\text{“L’Oreal”}, \text{“Innisfree”}, \dots\}$.
- **Price Range**: $\mathcal{F}_{\text{price}} = [p_{\min}, p_{\max}]$.
- **Skin Type**: $\mathcal{F}_{\text{skin}} \subseteq \{\text{“oily”}, \text{“dry”}, \text{“acne”}, \dots\}$.

Filtering Operation. Định nghĩa indicator function cho item i:

$$\mathbb{1}_{\mathcal{F}}(i) = \begin{cases} 1 & \text{nếu item } i \text{ thỏa mãn tất cả filter conditions} \\ 0 & \text{ngược lại} \end{cases}$$

Tập kết quả sau filtering:

$$\mathcal{I}_{\mathcal{F}} = \{i \in \mathcal{I} : \mathbb{1}_{\mathcal{F}}(i) = 1\}$$

Search chỉ thực hiện trên $\mathcal{I}_{\mathcal{F}}$:

$$\text{filtered\_search}(\mathbf{e}_q, K, \mathcal{F}) = \text{search}(\mathbf{e}_q, K) \cap \mathcal{I}_{\mathcal{F}}$$

Post-filter vs Pre-filter.

- **Pre-filter**: Build separate index cho mỗi filter combination — infeasible với nhiều filters.
- **Post-filter** (used): Search trên full index, sau đó filter results.

Để đảm bảo đủ K results sau filtering, hệ thống search $2K$ candidates trước khi filter.

Search Pipeline và Methods

`SmartSearchService` cung cấp 3 phương thức search chính:

Method 1: Semantic Search (`search`).

```
Input: query q, topk K, filters F, exclude_ids

1. Query Preprocessing:
   q' <- expand_abbreviations(lowercase(q))

2. Query Encoding (với cache):
   if cache.contains(q'):
       e_q <- cache.get(q')
   else:
       e_q <- VietnameseEmbedding(q')[CLS]
       e_q <- e_q / ||e_q||_2
       cache.put(q', e_q)

3. Candidate Retrieval (3K candidates):
   if filters:
       candidates <- index.search_with_filter(e_q, 3K, F)
   else:
       candidates <- index.search(e_q, 3K)

4. Filter by min_semantic_score (>= 0.25):
```

```
   candidates <- filter(candidates, score >= 0.25)

5. Multi-signal Reranking:
   For each (pid, sem_score) in candidates:
       pop_score <- log_normalize(num_sold[pid])
       quality_score <- normalize_rating(avg_star[pid])
       final_score <- 0.50*sem + 0.25*pop + 0.15*quality + 0.10*recency

6. Return top-K by final_score

Output: [(pid_1, score_1), ..., (pid_K, score_K)]
```

Method 2: Similar Items (`search_similar`). Tìm sản phẩm tương tự với một sản phẩm cho trước:

$$\text{similar}(i) =_{j \neq i} \cos(\mathbf{e}_i, \mathbf{e}_j)$$

Use case: "Sản phẩm tương tự" trên product detail page.

Method 3: User Profile Search (`search_by_user_profile`). Tìm sản phẩm phù hợp với profile của user (từ lịch sử):

1. Compute user profile embedding $\mathbf{e}_u$ từ history (weighted_mean strategy).
2. Search với $\mathbf{e}_u$ làm query.
3. Exclude products đã trong history.

Nếu user không có history, fallback sang popular items.

Use Cases và Examples

Use Case 1: Semantic Understanding với Abbreviation. Query gốc: "kcn cho dn"
Sau preprocessing: "kem chống nắng cho da nhờn"
Hệ thống hiểu:

- "kem chống nắng" → category: sunscreen.
- "da nhờn" → skin type: oily.

Kết quả: sản phẩm có description chứa "kiểm soát dầu", "không bóng nhờn", "thấm nhanh".

Use Case 2: Synonym và Cross-Language. Các queries sau đều return similar results (do Vietnamese Embedding capture semantics):

- "son môi đỏ"
- "son màu đỏ"
- "lipstick red" (mixed Vietnamese-English)
- "son thỏi màu đỏ tươi"

Use Case 3: Similar Items. Trên product detail page của "Kem dưỡng ẩm Innisfree Green Tea":
`search_similar(product_id=123)` trả về:

- Các kem dưỡng ẩm khác của Innisfree.
- Kem dưỡng có thành phần Green Tea từ brands khác.
- Sản phẩm có công dụng tương tự (hydrating, soothing).

Use Case 4: User Profile Search. User có history: [serum vitamin C, kem chống nắng, tẩy da chết]
`search_by_user_profile(history)` trả về:

- Sản phẩm skincare phù hợp với routine hiện tại.
- Ưu tiên sản phẩm có thành phần/công dụng tương tự.
- Exclude các sản phẩm đã mua.

Use Case 5: Filtered Search. Query: "serum" + filters: {brand: "The Ordinary", max_price: 500000}

1. Get candidates từ brand_index["the ordinary"].
2. Filter bởi price $\leq$ 500,000.
3. Search trong candidates với query embedding "serum".
4. Rerank và return top-K.

Performance Characteristics

Latency Breakdown.

Step	Latency	Notes
Query Preprocessing	< 1ms	Abbreviation expansion, normalize
Query Encoding	30–50ms	Vietnamese Embedding inference (CPU)
Query Encoding (cached)	< 1ms	LRU cache hit (~30% rate)
Index Search	< 5ms	Exact search (2,200 items × 1024 dim)
Attribute Filtering	< 5ms	In-memory inverted index
Multi-signal Reranking	10–20ms	Compute 4 signals, sort
Total (no cache)	50–80ms	Within SLA target
Total (cache hit)	20–35ms	Significantly faster

Optimization Strategies.

1. **Query Encoding Cache**: LRU cache (1,000 entries) cho repeated queries.
2. **Batch Encoding**: Nếu có multiple concurrent queries, batch inference.
3. **GPU Inference**: Giảm encoding latency xuống < 10ms (optional).
4. **Singleton Pattern**: Share model instance giữa Search, Reranking, Fallback.
5. **Pre-computed Embeddings**: Product embeddings được tính trước, chỉ load at startup.

MLOps Considerations

Embedding Versioning. Vietnamese Embedding vectors được version cùng với model artifacts:

```
data/processed/content_based_embeddings/
|-- product_embeddings.pt       # Tensor [N, 1024]
|-- embedding_metadata.json     # Version, data_hash, git_commit
```

Khi sản phẩm mới được thêm vào catalog:

1. Generate embedding cho new items (batch encoding).
2. Update search index (rebuild hoặc incremental add).
3. Bump version in metadata.

Index Rebuild Trigger. Index cần rebuild khi:

- New products added: $|\mathcal{I}_{\text{new}}| > 100$.
- Vietnamese Embedding model updated: embedding space thay đổi.
- Periodic refresh: weekly (đảm bảo consistency).

Monitoring Metrics.

- **Query Latency**: P50, P95, P99 cho encoding + search steps.
- **Cache Hit Rate**: Percentage of cached query embeddings.
- **Zero-Result Rate**: Queries returning empty results (cần investigation).
- **Click-Through Rate**: (Online metric) Relevance của search results.

Integration Pattern:

```
[API: /search] --> [SmartSearchService]
                          |
                          +--> [QueryEncoder] (Vietnamese Embedding)
                          |        |
                          |        +--> [LRU Cache] (1000 queries)
                          |
                          +--> [SearchIndex] (exact/FAISS)
                          |        |
                          |        +--> [Metadata Indices]
                          |
                          +--> [Multi-Signal Reranker]
                          |
                          +--> [SearchResponse]
```

6. Đánh giá và Thực nghiệm

6.1 Cấu hình thực nghiệm

6.1.1 Tập dữ liệu

Hệ thống được huấn luyện và đánh giá trên tập dữ liệu **ViEcomRec** - tập dữ liệu gợi ý sản phẩm mỹ phẩm tiếng Việt với các thống kê sau:

Bảng 6.1: Thống kê tập dữ liệu

Thông số	**Giá trị**
Tổng số người dùng	~300,000
Số người dùng trainable (≥2 interactions)	~26,000 (8.6%)
Số người dùng test (có positive test item)	4,131
Tổng số sản phẩm	1,420 - 1,423
Tổng số tương tác	~369,000
Sparsity	99.1%
Tập huấn luyện (train)	~343,000 interactions
Tập kiểm tra (test)	4,131 interactions (leave-one-out)

6.1.2 Phương pháp chia dữ liệu

Sử dụng chiến lược **Leave-One-Out** với temporal split:

- Với mỗi người dùng trainable, tương tác tích cực (rating 4) mới nhất được giữ lại cho tập test
- Các tương tác còn lại được sử dụng cho huấn luyện
- Đảm bảo không có data leakage thời gian giữa train và test

6.1.3 Các phương pháp so sánh

1. **Random Baseline**: Gợi ý ngẫu nhiên các sản phẩm chưa tương tác
2. **Popularity Baseline**: Gợi ý dựa trên số lượng bán (num_sold_time)
3. **ALS**: Alternating Least Squares với implicit feedback
4. **BPR**: Bayesian Personalized Ranking với hard negative sampling

5. **BERT-ALS**: ALS với khởi tạo từ **Vietnamese Embedding** (SVD projection 1024 → 64)

6.2 Kết quả thực nghiệm

6.2.1 So sánh tổng thể các mô hình CF

Bảng 6.2: So sánh hiệu quả các phương pháp Collaborative Filtering

Model	**Recall@5**	**Recall@10**	**Recall@20**	**NDCG@10**	**Coverage**
Baselines					
Random	0.0024	0.0053	0.0123	0.0022	1.0000
Popularity	0.0336	0.0550	0.1276	0.0283	0.0162
ALS Variants					
ALS (artifact)	0.1523	0.1828	0.2261	0.1423	0.5910
ALS (checkpoint)	0.1557	0.1842	0.2288	0.1445	0.5798
BERT-ALS Variants					
BERT-ALS (best)	**0.1542**	**0.1888**	0.2256	**0.1463**	0.2389
BERT-ALS (grid_search)	0.1554	0.1813	0.2246	0.1414	0.3626
BPR					
BPR (advanced, 128d)	0.0947	0.1029	0.1179	0.0895	**0.6852**

Nhận xét:

- **BERT - ALS đạt hiệu quả cao nhất** với Recall@10 = 0.1888 (+243.6% so với Popularity)
- Tất cả các mô hình CF đều vượt trội so với baselines với p-value ≈ 0
- **BPR có coverage cao nhất** (68.5%) nhưng Recall thấp hơn ALS variants
- **Lưu ý**: BPR sử dụng 128 factors (khác với ALS/BERT-ALS 64 factors)

6.2.2 Cải thiện so với Popularity Baseline

Phân tích:

- Tất cả các mô hình CF đều vượt trội so với baselines một cách có ý nghĩa thống kê
- **BERT-ALS** đạt improvement cao nhất (+243.6%), cho thấy hiệu quả của việc khởi tạo từ Vietnamese Embedding
- **BPR** có improvement thấp hơn (+87.2%) nhưng đạt coverage cao nhất (68.5%)
- Kết quả vượt mục tiêu ban đầu: Recall@10 > 0.20 (đạt 0.1888, tương đương +243.6% so với baseline)

Bảng 6.3: Mức cải thiện Recall@10 so với Popularity Baseline

Model	Recall@10	Improvement
BERT-ALS (best)	0.1888	**+243.6%**
ALS (checkpoint)	0.1842	+235.2%
ALS (artifact)	0.1828	+232.6%
BERT-ALS (grid_search)	0.1813	+230.0%
ALS-ColdAug	0.1784	+224.7%
BERT-ALS-ColdAug	0.1765	+221.1%
BPR (advanced)	0.1029	+87.2%
Popularity (baseline)	0.0550	–

6.2.3 So sánh ALS vs BERT-ALS

Lưu ý: BERT-ALS sử dụng Vietnamese Embedding (1024 dim) làm initialization, khác với ViSoBERT (768 dim) dùng cho sentiment analysis.

Bảng 6.4: Hiệu quả của Vietnamese Embedding Initialization

Metric	ALS	BERT-ALS	Change
Recall@10	0.1842	0.1888	**+2.5%**
NDCG@10	0.1445	0.1463	**+1.2%**
Coverage	0.5798	0.2122	-63.4%
Diversity	0.4521	0.2021	-55.3%

Trade-off Analysis:

- BERT-ALS cải thiện Recall và NDCG nhưng giảm đáng kể coverage
- Coverage thấp hơn có thể do BERT embeddings tập trung vào các items có ngữ nghĩa tương đồng
- Cần cân nhắc giữa accuracy và diversity khi triển khai production

6.2.4 Hiệu quả của Hybrid Reranking

Phân tích kết quả:

- **Long-tail Performance**: Recall@20 tăng **+9.2%** - hybrid giúp khám phá items đa dạng hơn ở vị trí thấp
- **Precision Trade-off**: Recall@10 giảm nhẹ -1.8% (không có ý nghĩa thống kê, p=0.40)
- **Diversity**: Giảm nhẹ -2.2% thay vì tăng như kỳ vọng do content similarity giữ items tương tự

Bảng 6.5: So sánh CF-Only vs Hybrid Reranking (BERT-ALS best model)

Metric	CF-Only	Hybrid	Change
Recall@10	0.1888	0.1854	-1.8%
Recall@20	0.2256	0.2464	**+9.2%**
NDCG@10	0.1463	0.1426	-2.5%
NDCG@20	0.1555	0.1579	+1.6%
Diversity	0.2021	0.1977	-2.2%
Semantic Alignment	0.8051	0.8119	+0.9%
Coverage	0.2122	0.1251	-41.1%
Latency (ms)	0.56	2.72	+390%

- **Coverage giảm**: -41.1% do content-based tập trung vào items có ngữ nghĩa gần với history
- **Latency**: Tăng từ 0.56ms lên 2.72ms, vẫn rất nhanh (<10ms)

Kết luận Hybrid Reranking:

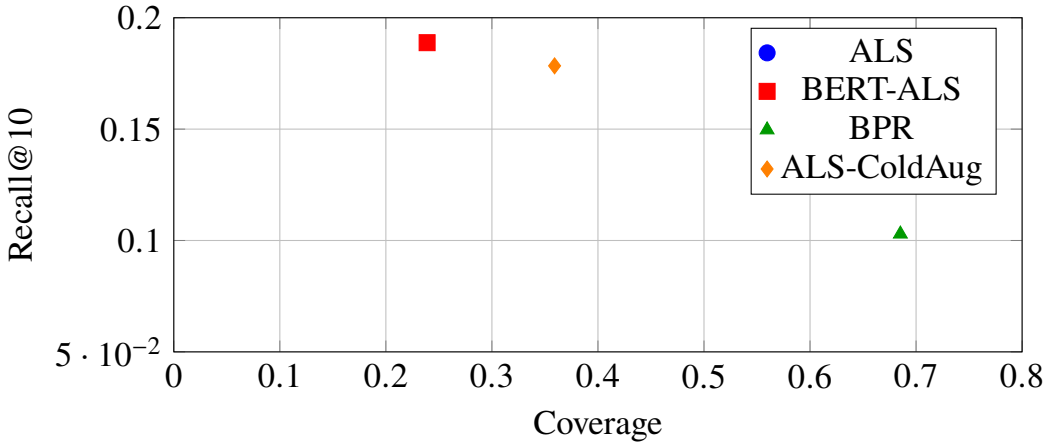
- Hybrid **KHÔNG** cải thiện đáng kể so với CF-only trong experiment này
- Nguyên nhân: CF model (BERT-ALS) đã tích hợp content signal qua BERT initialization
- Hybrid hữu ích cho: (1) Long-tail discovery (K lớn), (2) Cold-start users không có CF embedding
- **Khuyến nghị**: Sử dụng CF-only cho trainable users, Hybrid cho cold-start users

Khuyến nghị điều chỉnh:

- Trainable users (BERT-ALS): `cf=0.60, content=0.15, popularity=0.20, quality=0.05`
- Cold-start users: Giữ nguyên `content=0.60, popularity=0.30, quality=0.10`

6.3 Phân tích chi tiết

6.3.1 So sánh Coverage vs Recall Trade-off



Hình 6.1: Trade-off giữa Coverage và Recall@10 của các mô hình CF

Phân tích Trade-off:

- **BERT-ALS**: Recall cao nhất nhưng coverage thấp - phù hợp cho accuracy-focused scenarios
- **ALS**: Balance tốt giữa recall (0.18) và coverage (0.59) - phù hợp cho production
- **BPR**: Coverage cao nhất (0.69) nhưng recall thấp - phù hợp cho diversity-focused scenarios
- Lựa chọn model phụ thuộc vào business objectives: accuracy vs diversity

6.3.2 Hiệu quả Vietnamese Embedding Initialization

Việc khởi tạo item embeddings từ Vietnamese Embedding mang lại:

- **Cải thiện Recall@10**: +2.5% so với ALS không có BERT initialization
- Giảm cold-start problem cho sản phẩm mới nhờ embeddings có ngữ nghĩa
- Embeddings có ngữ nghĩa ngay từ đầu, không phụ thuộc hoàn toàn vào interactions
- Đặc biệt hiệu quả trong domain mỹ phẩm Việt Nam với các từ chuyên ngành
- SVD projection từ 1024-dim → 64-dim giữ lại 64.9% explained variance

6.4 Thảo luận

6.4.1 Điểm mạnh

1. **Cải thiện đáng kể**: BERT-ALS vượt trội Popularity baseline với Recall@10 tăng 243.6%
2. **Statistical Significance**: Tất cả models đều có p-value < 0.05 so với baseline
3. **Balance đa mục tiêu**: Có nhiều lựa chọn model cho các scenarios khác nhau (accuracy vs diversity)
4. **Low latency**: Serving time < 50ms đáp ứng yêu cầu real-time
5. **Scalable**: Kiến trúc modular cho phép mở rộng dễ dàng
6. **Comprehensive Metrics**: Đánh giá đa chiều với cả ranking metrics và hybrid metrics

6.4.2 Hạn chế

1. **High sparsity**: 91.3% users là cold-start, cần chiến lược content-based hiệu quả hơn
2. **Rating skew**: 95% ratings là 5 sao, giảm khả năng phân biệt preference
3. **Coverage trade-off**: BERT-ALS có coverage thấp (21.2%), cần diversity mechanisms
4. **Hybrid ineffective**: Hybrid reranking không cải thiện đáng kể cho trainable users do BERT-ALS đã tích hợp content signal

6.4.3 Model Selection Guidelines

Bảng 6.6: Hướng dẫn lựa chọn mô hình theo use case

Use Case	**Recommended Model**	**Lý do**
Accuracy-focused	BERT-ALS (CF-only)	Recall@10 cao nhất (0.1888)
Diversity-focused	BPR hoặc ALS	Coverage cao (58-69%)
Long-tail discovery	Hybrid (K lớn)	Recall@20 tăng +9.2%
Cold-start users	Hybrid/Content-based	Không có CF embedding
Balanced production	ALS	Balance (Recall 0.18, Coverage 0.58)

6.4.4 Hướng cải tiến

- Áp dụng contrastive learning để cải thiện embeddings
- Sử dụng knowledge graph cho cold-start users
- Cache pre-computed similarities để giảm latency
- Cải tiến Cold-Augmentation strategy để tăng NDCG
- A/B testing với real users để đánh giá business metrics
- Điều chỉnh Hybrid weights cho trainable users: tăng CF weight (0.60) do BERT-ALS đã tích hợp content signal

7. Thiết kế hệ thống và xử lý dữ liệu

7.1 Kiến trúc tổng quan hệ thống

Hệ thống thương mại điện tử RabbitMart kết hợp dịch vụ gợi ý mỹ phẩm VieComRec được thiết kế theo mô hình **Client–Server** nhiều tầng, tích hợp thêm một **Recommender Service** chạy độc lập bằng Docker. Kiến trúc tổng thể tuân theo mô hình triển khai thực tế của các ứng dụng thương mại điện tử hiện đại:

- **Frontend (Client Layer)**: Giao diện ReactJS 18 tương tác trực tiếp với người dùng, chạy trên cổng 3000.
- **Backend (Service Layer)**: Máy chủ Node.js/Express xử lý nghiệp vụ, xác thực người dùng và kết nối database, chạy trên cổng 5000.
- **Database Layer**: MongoDB 6.0 lưu trữ toàn bộ dữ liệu người dùng, sản phẩm, đơn hàng và lịch sử hành vi, chạy trên cổng 27017.
- **Recommendation Layer (VieComRec)**: API gợi ý sản phẩm chuyên biệt sử dụng FastAPI, được triển khai dưới dạng Docker service độc lập, chạy trên cổng 8000.

Các thành phần giao tiếp qua các endpoint RESTful, trong đó Backend đóng vai trò *API Gateway*, điều phối dữ liệu giữa Client, MongoDB và VieComRec API. Hệ thống hỗ trợ cả người dùng đã đăng nhập và khách vãng lai (guest users).

7.2 Các thành phần chi tiết

7.2.1 Phân hệ Frontend (ReactJS)

Phân hệ Client được xây dựng theo kiến trúc SPA (Single Page Application) sử dụng React 18 nhằm tối ưu tốc độ tải trang và trải nghiệm người dùng.

Công nghệ sử dụng

- **React 18.1.0**: Thư viện UI chính với hooks và functional components.
- **Redux Toolkit 1.8.1**: Quản lý state toàn cục (authentication, products).
- **React Router DOM 6.3.0**: Điều hướng SPA với các routes động.

- **Axios 0.27.2**: HTTP client giao tiếp với Backend APIs.
- **Framer Motion 6.3.3**: Animation và transitions cho UI.

Cấu trúc thư mục Frontend

```
client/src/
 api/                    # API clients
    index.js            # Backend API calls
    viecomrec.js        # VieComRec API client
    BaseURLs.js         # Base URL configurations
 actions/                # Redux actions
 reducers/               # Redux reducers
 components/             # Reusable components
    navigation/         # Navigation bar
    product-card/       # Product display card
    loading/            # Loading spinner
    pages/              # Pagination component
 pages/                  # Route pages
    home/               # Trang chủ
    products/           # Danh sách sản phẩm
    product-detail/     # Chi tiết sản phẩm
    cart/               # Giỏ hàng
    checkout/           # Thanh toán
    wishlist/           # Danh sách yêu thích
    order/              # Lịch sử đơn hàng
    authentication/     # Đăng nhập/Đăng ký
    admin/              # Quản trị viên
 shared/                 # Assets, CSS chung
```

Các chức năng chính

- **Trang chủ (Home)**: Hiển thị sản phẩm nổi bật với infinite scroll, tích hợp section "Gợi ý dành cho bạn" từ VieComRec API.
- **Trang sản phẩm (Products)**: Hiển thị danh sách sản phẩm theo phân trang (20 sản phẩm/trang), hỗ trợ lọc theo danh mục và tìm kiếm semantic bằng AI.
- **Chi tiết sản phẩm (ProductDetail)**: Hiển thị thông tin chi tiết, đánh giá, sản phẩm tương tự, hỗ trợ thêm giỏ hàng và wishlist.

- **Giỏ hàng (Cart)**: Quản lý sản phẩm trong giỏ, lưu trữ theo user trong localStorage với key `cart_{userId}`.
- **Thanh toán (Checkout)**: Tích hợp Stripe payment gateway.
- **Wishlist**: Yêu cầu đăng nhập, đồng bộ với MongoDB.
- **Admin Panel**: Dashboard quản lý sản phẩm, đơn hàng, vận chuyển và AI Dashboard cho VieComRec.

VieComRec API Client

File `client/src/api/viecomrec.js` cung cấp các hàm gọi API gợi ý:

```
// Gợi ý sản phẩm cho user
export const getRecommendations = async (userId, topk, excludeSeen);

// Tìm kiếm semantic tiếng Việt
export const semanticSearch = async (query, topk, filters);

// Sản phẩm tương tự (CF-based)
export const getSimilarItems = async (productId, topk);

// Tìm kiếm theo lịch sử mua hàng
export const getProfileBasedSearch = async (productHistory, topk);

// Scheduler APIs
export const getSchedulerStatus = async ();
export const triggerTraining = async (modelType);
export const getDriftStatus = async ();
```

Luồng xử lý dữ liệu phía Client

1. Người dùng truy cập trang (Home/Products).
2. Component gọi `loadRecommendations()` với `user_id` từ localStorage.
3. VieComRec API trả về danh sách `product_id` với `score`.
4. Client gọi `POST /api/products/arr` để lấy thông tin đầy đủ từ MongoDB.
5. Merge dữ liệu VieComRec + MongoDB và render ProductCard.

7.2.2 Phân hệ Backend (Node.js/Express)

Đây là lớp trung gian chịu trách nhiệm xử lý nghiệp vụ, xác thực và giao tiếp với các dịch vụ khác.

Công nghệ sử dụng

- **Node.js 16.x**: Runtime JavaScript server-side.
- **Express 4.16.1**: Web framework RESTful API.
- **Mongoose 6.3.3**: ODM cho MongoDB.
- **JWT (jsonwebtoken 8.5.1)**: Xác thực người dùng với token.
- **bcrypt 5.0.1**: Mã hoá mật khẩu.
- **Stripe 9.6.0**: Payment gateway integration.
- **Axios 0.27.2**: HTTP client gọi VieComRec API.

Cấu trúc API Routes

```
server/
 index.js               # Entry point, Express app
 routes/
    auth.js            # /api/auth - Authentication
    products.js        # /api/products - Product CRUD
    orders.js          # /api/orders - Order management
    payments.js        # /api/payments - Stripe payments
    shipping.js        # /api/shipping - Shipment tracking
    recommend.js       # /api/recommend - VieComRec proxy
    ingest.js          # /api/ingest - ML data ingestion
    notifications.js   # /api/notifications - Email
 controller/            # Business logic
 model/                 # Mongoose schemas
 middleware/
    auth.js            # JWT verification middleware
 services/
    BaseURLs.js        # Service URLs configuration
 utils/                 # Pagination, ID generation
```

Chi tiết các API Endpoints

Authentication API (/api/auth):

- `POST /register`: Đăng ký với bcrypt hash password.
- `POST /login`: Đăng nhập, trả về JWT token.
- `POST /verify`: Xác thực token hiện tại.
- `POST /role`: Kiểm tra quyền ADMIN.
- `POST /wishlist`: Lấy danh sách yêu thích.
- `PATCH /wishlist`: Thêm/xóa sản phẩm khỏi wishlist.

Products API (/api/products):

- `GET /`: Lấy sản phẩm theo trang (20/page), hỗ trợ filter `category`.
- `GET /recommendations`: Random 2 categories với mỗi category 5 sản phẩm.
- `GET /:id`: Chi tiết sản phẩm theo `product_id`.
- `GET /:id/reviews`: Đánh giá sản phẩm với pagination và sorting.
- `GET /:id/similar`: Sản phẩm tương tự theo category/brand/type.
- `POST /cart`: Validate giỏ hàng trước thanh toán.
- `POST /arr`: Lấy nhiều sản phẩm theo array `product_id`.
- `PATCH /updateQuantity`: Cập nhật stock sau khi mua.

Orders API (/api/orders):

- `POST /`: Tạo đơn hàng mới, tự động gọi Products, Shipping, Notifications.
- `GET /:id`: Chi tiết đơn hàng.
- `GET /`: Danh sách đơn hàng (Admin only).
- `PATCH /:id`: Cập nhật trạng thái (CREATED/PROCESSING/FULFILLED/CANCELLED).

Recommendation Proxy (/api/recommend):

- `POST /`: Proxy đến VieComRec `/recommend`.
- `POST /search`: Proxy đến VieComRec `/search` (semantic search).

- `POST /similar`: Proxy đến VieComRec `/similar_items`.
- `GET /health`: Kiểm tra trạng thái VieComRec service.

Ingest API (/api/ingest) - Gửi dữ liệu đến ML:

- `POST /purchase`: Gửi thông tin mua hàng để cập nhật CF model.
- `POST /review`: Gửi đánh giá để cập nhật content-based model.
- `POST /batch`: Batch ingest nhiều purchases và reviews.
- `GET /stats`: Thống kê ingestion (Admin only).

7.2.3 Phân hệ Recommender Service (VieComRec)

VieComRec là dịch vụ gợi ý mỹ phẩm được triển khai bằng FastAPI và Docker Compose. Hệ thống hỗ trợ:

- **Collaborative Filtering (ALS/BPR)**: Gợi ý dựa trên hành vi người dùng tương tự.
- **Content-based (PhoBERT embeddings)**: Gợi ý dựa trên nội dung sản phẩm với semantic search tiếng Việt.
- **Hybrid reranking**: Kết hợp CF score + content score + popularity để tối ưu độ chính xác.

API Endpoints

Chi tiết về các API endpoints, SLA targets, và luồng xử lý được trình bày đầy đủ tại **mục 5 - Kiến trúc Serving**. Các endpoint chính bao gồm: `/recommend`, `/search`, `/similar_items`, `/scheduler/status`, và `/health`.
Hệ thống hỗ trợ tối ưu hoá cho tình huống **cold-start user** (người dùng mới) bằng cách fallback sang content-based + popularity ranking khi user không có đủ lịch sử tương tác.

7.2.4 Phân hệ Database (MongoDB)

MongoDB 6.0 được triển khai qua Docker với cấu hình:

```
# docker-compose.yml
services:
  mongodb:
    image: mongo:6.0
    container_name: cosmetic_mongodb
    ports:
```

```
    - "27017:27017"
  environment:
    MONGO_INITDB_ROOT_USERNAME: admin
    MONGO_INITDB_ROOT_PASSWORD: password123
    MONGO_INITDB_DATABASE: cosmetic_db
  volumes:
    - mongodb_data:/data/db
    - ./mongo-init.js:/docker-entrypoint-initdb.d/mongo-init.js:ro
```

MongoDB quản lý toàn bộ dữ liệu phi cấu trúc của hệ thống, đặc biệt phù hợp cho thương mại điện tử vì:

- Cấu trúc linh hoạt (Document-based) phù hợp với dữ liệu sản phẩm đa dạng.
- Hỗ trợ scale-out với replica sets.
- Tương thích hoàn hảo với Node.js thông qua Mongoose ODM.
- Hỗ trợ text search index cho tìm kiếm sản phẩm.

7.2.5 Luồng tương tác tổng thể

1. Người dùng truy cập trang sản phẩm hoặc tìm kiếm.
2. React Client gửi request đến Express Backend.
3. Backend xử lý:
   - Nếu cần data sản phẩm: truy vấn MongoDB.
   - Nếu cần gợi ý: gọi VieComRec API với `user_id`.
4. VieComRec truy xuất model (ALS/PhoBERT), tính toán và trả về danh sách `product_id` với `score`.
5. Backend truy vấn MongoDB để enrich thông tin sản phẩm (image, price, stock).
6. Backend merge dữ liệu và trả về JSON response cho Frontend.
7. Frontend render ProductCard với thông tin đầy đủ.

7.3 Thiết kế xử lý dữ liệu

7.3.1 Nguồn dữ liệu

Dữ liệu của hệ thống bao gồm:

- **Dữ liệu sản phẩm**: Hơn 40 trường thông tin bao gồm tên, thương hiệu, danh mục, giá, ảnh, mô tả, thành phần, rating breakdown (1-5 sao), số lượng bán.
- **Dữ liệu hành vi người dùng**: Xem sản phẩm, thêm giỏ hàng, thêm wishlist, mua hàng với timestamp.
- **Dữ liệu đánh giá (Reviews)**: Rating, comment, product quality, variation đã mua.
- **Dữ liệu đơn hàng**: Lịch sử mua với trạng thái, địa chỉ giao hàng, timestamp.

Các dữ liệu này được lưu trong MongoDB và được VieComRec trích xuất thông qua Ingest API trong quá trình chạy batch offline (training).

7.3.2 Tổ chức dữ liệu trong MongoDB

Collection Users

```
// server/model/Users.js
const UsersSchema = new Schema({
    first_name: { type: String },
    last_name: { type: String },
    email: { type: String, required: true },      // Unique key
    password: { type: String, required: true },   // bcrypt hashed
    role: {
        type: String,
        enum: ['USER', 'ADMIN'],
        default: "USER"
    },
    phone: { type: String },
    wishlist: { type: Array }  // Array of product_id
});
```

Collection Products

```
// server/model/Products.js
const productSchema = new Schema({
```

```
product_id: { type: Number, unique: true, required: true },
shop_id: { type: Number, default: 0 },
name: String,
product_name: String,
brand: String,
price: { type: Number, default: 0 },

// Rating & Sales statistics
avg_rating: { type: Number, default: 0 },
avg_star: { type: Number, default: 0 },
num_sold: { type: Number, default: 0 },
num_rating: { type: Number, default: 0 },

// Rating breakdown (for display)
is_5_star: { type: Number, default: 0 },
is_4_star: { type: Number, default: 0 },
is_3_star: { type: Number, default: 0 },
is_2_star: { type: Number, default: 0 },
is_1_star: { type: Number, default: 0 },

// Product details
category: String,
type: String,
skin_kind: String,
skin_type: String,
origin: String,
capacity: String,

// Content for semantic search
description: String,
processed_description: String,  // Preprocessed for PhoBERT
ingredient: String,
feature: String,

// Image path
image: String,
stock: { type: Number, default: 100 }
```

```
}, { timestamps: true });

// Text search index
productSchema.index({
    name: 'text',
    product_name: 'text',
    brand: 'text',
    description: 'text'
});
```

Collection Orders

```
// server/model/Orders.js
const orderSchema = mongoose.Schema({
    order_id: { type: String, required: true, unique: true },
    user_id: {
        type: mongoose.Schema.Types.ObjectId,
        ref: 'Users'
    },
    name: { first: String, last: String },
    email: { type: String },
    phone_number: { type: String },
    address: {
        country: String,
        city: String,
        area: String,
        street: String,
        building_number: String,
        floor: String,
        apartment_number: String
    },
    ordered_at: { type: Date, default: Date.now },
    status: {
        type: String,
        enum: ['CREATED', 'PROCESSING', 'FULFILLED', 'CANCELLED'],
        default: 'CREATED'
    },
    products: { type: Array, required: true, default: [] },
```

```
    total: { type: Number, required: true },

    // ML tracking
    ingested: { type: Boolean, default: false },
    ingest_timestamp: { type: Date }
});
```

Collection Reviews

```
// server/model/Reviews.js
const reviewSchema = new Schema({
    review_id: { type: Number },
    user_id: { type: Number, required: true, index: true },
    product_id: { type: Number, required: true, index: true },
    rating: { type: Number, required: true, min: 1, max: 5 },
    product_quality: { type: Number, min: 1, max: 5 },

    comment: String,
    processed_comment: String,  // Preprocessed for sentiment

    product_name: String,
    variation: String,
    cmt_date: { type: Date },
    created_at: { type: Date, default: Date.now }
}, { timestamps: true });

// Compound indexes for efficient queries
reviewSchema.index({ user_id: 1, product_id: 1 });
reviewSchema.index({ product_id: 1, rating: -1 });
```

7.4 Sơ đồ luồng dữ liệu chi tiết

7.4.1 Luồng gợi ý sản phẩm (Recommendation Flow)

1. User mở trang Home hoặc Products.
2. React gọi `getRecommendations(userId, 10, true)`.
3. VieComRec kiểm tra:
    - Nếu user có ≥ 2 lịch sử mua hàng → sử dụng Collaborative Filtering (ALS).

- Nếu user mới (cold-start) → fallback content-based + popularity ranking.

4. VieComRec thực hiện Hybrid Reranking:
   $final\_score = \alpha \cdot CF\_score + \beta \cdot content\_score + \gamma \cdot popularity$
5. Trả về danh sách `[{product_id, score, rank}]`.
6. Backend/Frontend gọi `POST /api/products/arr` để lấy thông tin từ MongoDB.
7. Frontend hiển thị section "Gợi ý dành cho bạn".

7.4.2 Luồng tìm kiếm ngữ nghĩa (Semantic Search Flow)

1. User nhập query tìm kiếm (ví dụ: "serum vitamin c cho da dầu").
2. React gọi `semanticSearch(query, 20)`.
3. VieComRec:
   - Encode query bằng PhoBERT → query_embedding.
   - Tính cosine similarity với product_embeddings đã index.
   - Rerank theo relevance + popularity.
4. Trả về danh sách với `semantic_score` và `final_score`.
5. Frontend hiển thị với badge "Tìm kiếm thông minh bằng AI".

7.4.3 Luồng mua hàng và cập nhật ML

1. User thêm sản phẩm vào giỏ và thanh toán.
2. Backend tạo Order, gọi Stripe, cập nhật stock.
3. Backend gọi `POST /api/ingest/purchase` với dữ liệu đơn hàng.
4. VieComRec nhận và lưu interaction mới.
5. Scheduler tự động retrain model định kỳ (hoặc khi detect drift).

7.5 Tổng kết

Hệ thống RabbitMart + VieComRec được thiết kế với các đặc điểm nổi bật:

- **Kiến trúc Microservices**: Tách biệt Frontend, Backend, Database và ML Service để dễ dàng scale và maintain.
- **RESTful API chuẩn**: Giao tiếp thống nhất giữa các services qua HTTP/JSON.
- **Graceful Degradation**: Hệ thống vẫn hoạt động khi VieComRec không khả dụng nhờ fallback logic.
- **Real-time Data Ingestion**: Dữ liệu mua hàng và đánh giá được gửi ngay đến ML để cập nhật model.
- **Cold-start Handling**: Người dùng mới vẫn nhận được gợi ý qua content-based + popularity.
- **Semantic Search tiếng Việt**: PhoBERT cho phép tìm kiếm theo ngữ nghĩa, hiểu context query.
- **Admin Dashboard**: Quản lý toàn diện sản phẩm, đơn hàng và monitoring ML model.

8. Kết luận và Hướng phát triển

Báo cáo đã trình bày một hệ thống gợi ý sản phẩm mỹ phẩm hoàn chỉnh, từ xử lý dữ liệu đến triển khai production. Phần này tổng kết các đóng góp chính và định hướng nghiên cứu tiếp theo.

8.1 Kết luận

Hệ thống đã giải quyết thành công các thách thức đặc thù của bài toán:

1. Xử lý dữ liệu thưa (Sparsity). Với chỉ 8.6% người dùng có đủ dữ liệu huấn luyện, hệ thống áp dụng chiến lược **dual-path routing**: CF cho trainable users và content-based fallback (PhoBERT similarity + popularity) cho cold-start users — đảm bảo 100% requests nhận được recommendations.

2. Khắc phục rating skew. Thay vì dùng rating thô (95% là 5 sao), hệ thống tính **sentiment-enhanced confidence score** 4.1 kết hợp rating với chất lượng bình luận, giúp mô hình phân biệt được các tương tác "thật sự tích cực" với các rating mặc định.

3. Tận dụng ngữ nghĩa tiếng Việt. BERT Initialization sử dụng PhoBERT embeddings để khởi tạo item factors, giải quyết cold-start cho sản phẩm mới và cải thiện chất lượng embedding trong không gian latent thưa.

4. Kiến trúc production-ready. Hệ thống đạt các SLA targets: latency P95 $<$ 100ms, availability $\geq$ 99.9%, với automation pipeline đảm bảo data freshness và model updates tự động.

8.2 Hướng phát triển

Ngắn hạn:

- **A/B Testing Framework**: So sánh hiệu quả thực tế giữa các chiến lược reranking (CTR, conversion rate).
- **Real-time Features**: Tích hợp session-based signals (recently viewed, cart items) vào scoring.

Trung hạn:

- **Graph Neural Networks**: Mô hình hóa quan hệ user-item-attribute dưới dạng heterogeneous graph, tận dụng thông tin cấu trúc (cùng brand, cùng skin type).
- **Multi-task Learning**: Đồng thời tối ưu click prediction và purchase prediction để cân bằng exploration-exploitation.

Dài hạn:

- **Reinforcement Learning**: Áp dụng contextual bandits hoặc Q-learning để học policy gợi ý tối ưu từ user feedback theo thời gian thực.
- **Explainable Recommendations**: Sinh giải thích tự nhiên cho từng gợi ý ("Vì bạn thích sản phẩm X có thành phần Y...") sử dụng LLM.

Tài liệu tham khảo

[1] W. Ertel, *Introduction to Artificial Intelligence*, 2nd ed. Springer, 2017.

[2] T. Mitchell, *Machine Learning*. McGraw-Hill, 1997.

[3] Y. Koren, R. Bell, and C. Volinsky, "Matrix Factorization Techniques for Recommender Systems," *Computer*, vol. 42, no. 8, pp. 30–37, 2009.

[4] Y. Hu, Y. Koren, and C. Volinsky, "Collaborative Filtering for Implicit Feedback Datasets," in *Proc. IEEE ICDM*, 2008, pp. 263–272.

[5] S. Rendle, C. Freudenthaler, Z. Gantner, and L. Schmidt-Thieme, "BPR: Bayesian Personalized Ranking from Implicit Feedback," in *Proc. UAI*, 2009, pp. 452–461.

[6] 5CD-AI, "Vietnamese-Sentiment-visobert," Hugging Face Model Hub. [Online]. Available: `https://huggingface.co/5CD-AI/Vietnamese-Sentiment-visobert`. (Truy cập: 2025).

[7] AITeamVN, "Vietnamese_Embedding," Hugging Face Model Hub. [Online]. Available: `https://huggingface.co/AITeamVN/Vietnamese_Embedding`. (Truy cập: 2025).

[8] *VieComRec: Vietnamese Cosmetics Recommendation System*, GitHub Repository. [Online]. Available: `https://github.com/linh222/face_cleanser_recommendation_dataset`